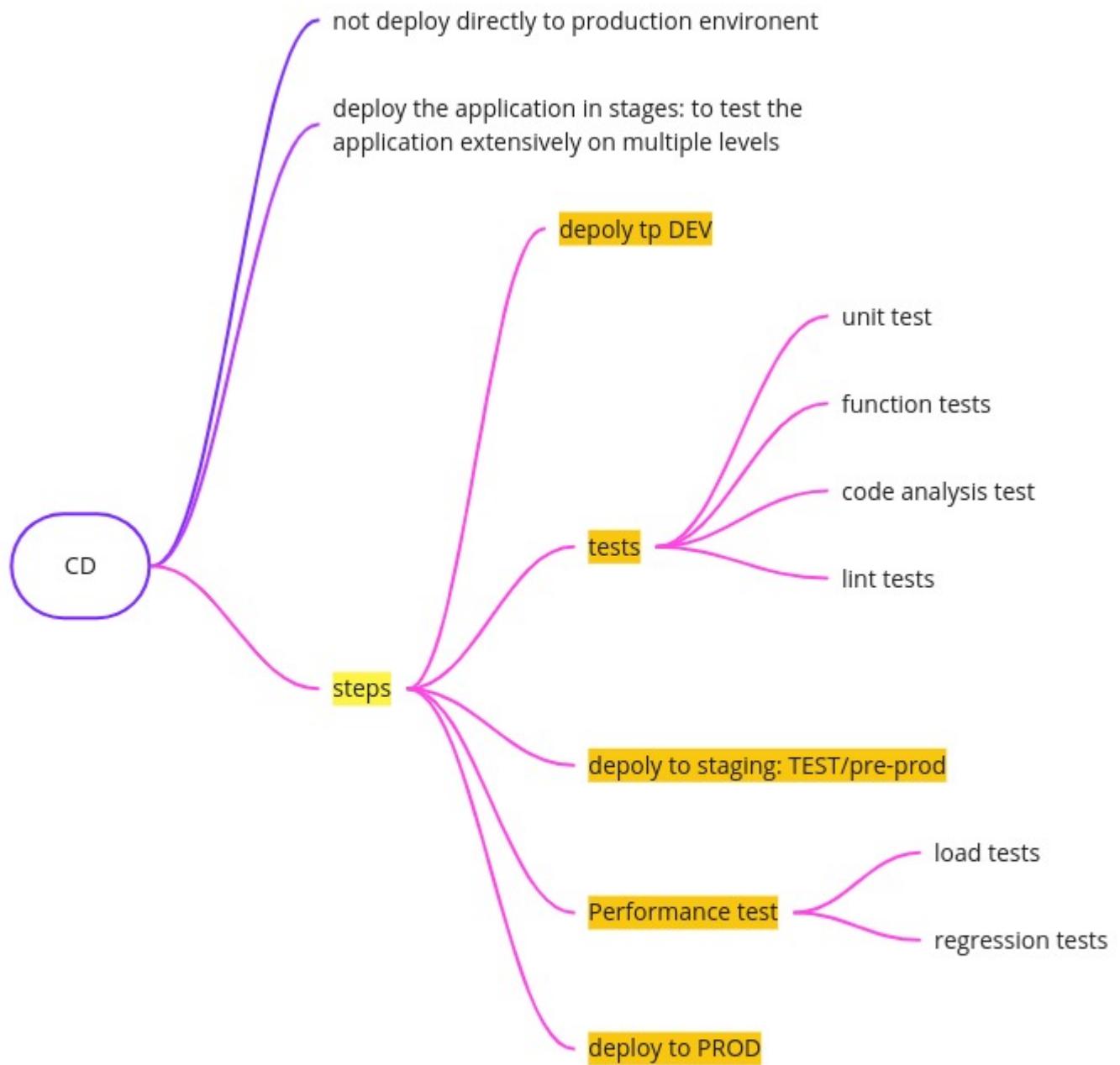


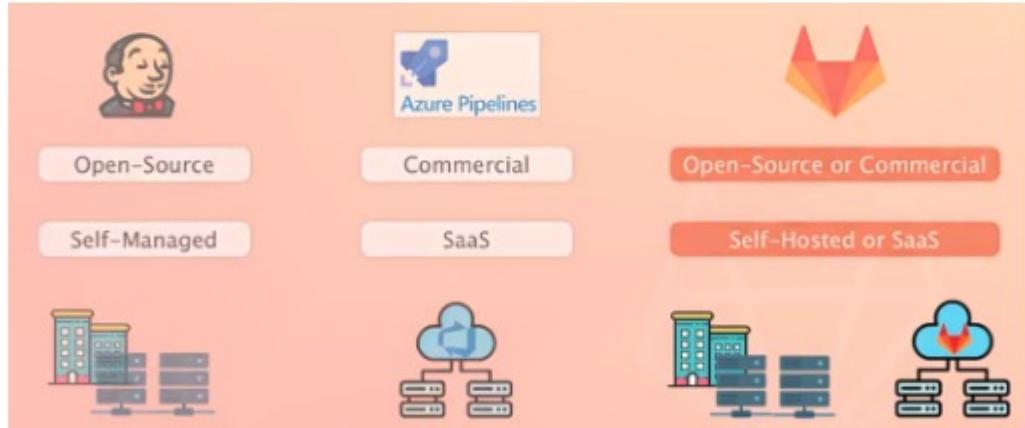
CI/CD: frequently deliver changes to custom in an automated way





Gitlab CI/CD platform:

- built-in CI/CD to streamline testing and delivery
- **Built-in features:**
 - built-in security capabilities
 - 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 arbitrary names for your jobs
- must contain at least the script clause
- script specifies the commands to execute

```
home > gu > Documents > DevOps > gitlabCI/CD > .gitlab-ci.yml
1  run_tests:
2  # prepare env datasets
3  | before_script:
4  |   - echo "Preparing test data..."
5  | script:
6  |   # script needs a list, using hyphen
7  |   - echo "Running tests...."
8  | after_script:
9  |   - echo "Clean up temporary files..."
10
11
12 build_image:
13 | script:
14 |   - echo "Building docker image...."
15 |   - echo "Tagging the docker image"
16
17
18 push_image:
19 | script:
20 |   - echo "Logging into docker registry...."
21 |   - echo "Push docker image to registry..."
```

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

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

- GitLab detects the `.yml` file and builds it.
- 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 in 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 when all jobs are successful, next stage will be started

CI/CD
Pipelines
Editor
Jobs
Schedules
Security & Compliance
Deployments
Packages & Registries
Infrastructure
Monitor
Analytics
Wiki

Status	Pipeline	Triggerer	Stages
passed	Update .gitlab-ci.yml file 8574745976 P main => db7d2eed		
failed	Update .gitlab-ci.yml file 8574745908 P main => 274e0958		
passed	Update .gitlab-ci.yml file 8574740739 P main => ef4592b0		
failed	Add new file 8574683308 P main => 5397668a		

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:

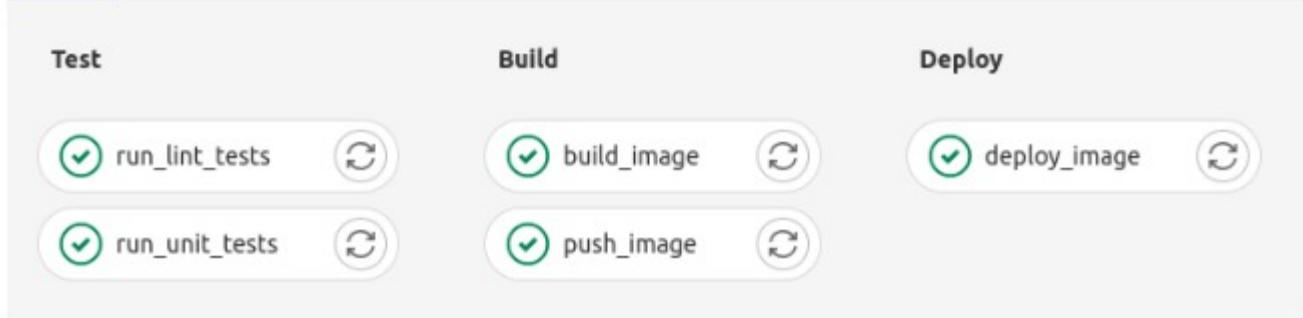
```

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


run_lint_tests:
| stage: test
needs:
| | - run_unit_tests

```

Pipeline Needs Jobs 5 Tests 0



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

```
1 pwd
2 ls
3 mkdir test-data
4 ls
```

Each pipeline is run on a fresh environment

```
stages:
- test
- build
- deploy

run_unit_tests:
stage: test
# prepare env datasets
before_script:
- echo "Preparing tet data..."
script:
# script neCeds a list, using hyphen
- 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 hyphen
- 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 to validate code before merging it into the main branch

=> tell gitlab pipeline logic just run for main branch

keywords: **only / except**

```
18 | | - rm -r test-data
19
20 run_lint_tests:
21   stage: test
22   # prepare env datasets
23   before_script:
24     - echo "Preparing test data..."
25   script:
26     # script needs a list, using hyphen
27     - echo "Running tests...."
28   after_script:
29     - echo "Clean up temporary files..."
30
31
32 build_image:
方法1 33   only:
34     - main
35   stage: build
36   script:
```

New branches still execute test, but not build anymore

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

Situation 2

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

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

How does gitlab know whether we are in main or branch?

-> In every pipeline GitLab provides predefined **environment variables**, is available in GitLab docs

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

GitLab Predefined variables:

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

```
workflow:
  rules:
方法2 1   if: $CI_COMMIT_BRANCH != "main"
    | when: never
    - when: always

stages:
  - test
  - build

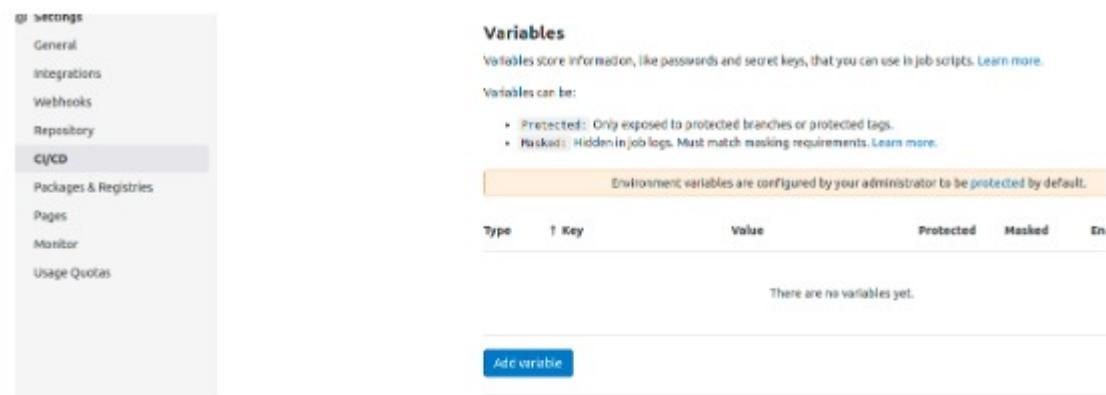
workflow:
  rules:
    - if: $CI_COMMIT_BRANCH != "main" && $CI_PIPELINE_SOURCE != "merge_request-event"
      | when: never
      - when: always

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

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

use cases: run pipeline for every microservice

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



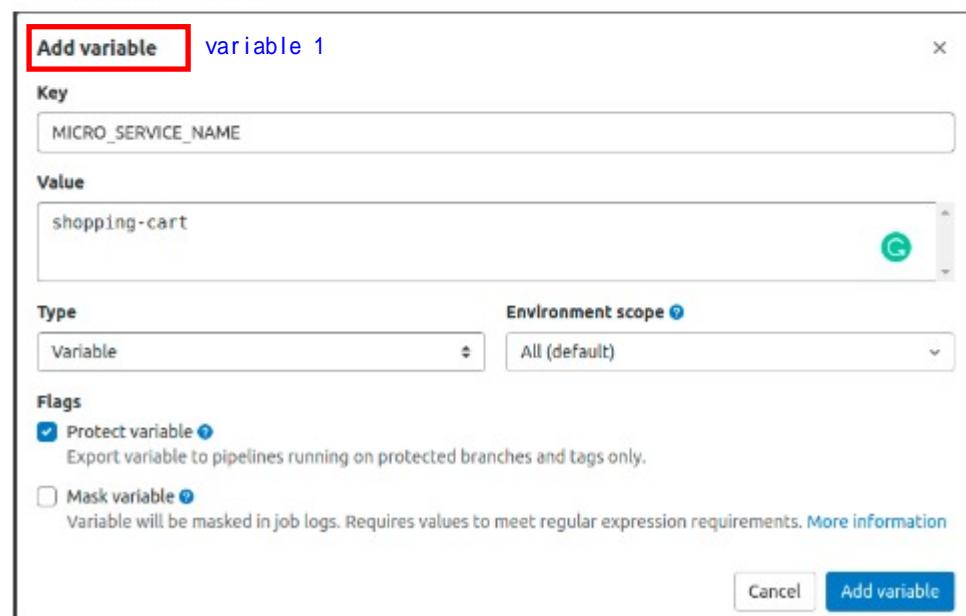
The screenshot shows the 'Variables' section in the GitLab CI/CD settings. On the left, there's a sidebar with options like General, Integrations, Webhooks, Repository, CI/CD (which is selected), Packages & Registries, Pages, Monitor, and Usage Quotas. The main area has a heading 'Variables' with a sub-instruction: 'Variables store information, like passwords and secret keys, that you can use in job scripts. Learn more.' Below this, it says 'Variables can be:' with two items: 'Protected' (Only exposed to protected branches or protected tags) and 'Masked' (Hidden in job logs. Must match masking requirements. Learn more.). A note at the top right states: 'Environment variables are configured by your administrator to be **protected** by default.' A table below lists variables, showing one entry: 'There are no variables yet.' A blue 'Add variable' button is located at the bottom left of the table area.

Add project variables 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



The dialog box for adding a new variable 'variable 1'. It has fields for 'Key' (MICRO_SERVICE_NAME) and 'Value' (shopping-cart). Under 'Type', 'Variable' is selected. Under 'Environment scope', 'All (default)' is chosen. In the 'Flags' section, 'Protect variable' is checked, with a note: 'Export variable to pipelines running on protected branches and tags only.' The 'Mask variable' option is also present with a note: 'Variable will be masked in job logs. Requires values to meet regular expression requirements. [More information](#)'. At the bottom are 'Cancel' and 'Add variable' buttons.

Type	Environment scope
Variable	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](#)

```

before_script:
  - echo "Preparing test data..."
script:
# script needs a list, using hyphen
  - 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

File Type Variables

"Variable" type

- ▶ Consists of key-value pair



'File' variable type

- ▶ Consists of a key, value and file

database-url: localhost
database-user: admin
database-password: secret



```
$ 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/VinodTremendous/mynodeapp/ci/cd/project/tmp/PROPERTIESFILE...
```

```

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

```
0 | - test  
1 | - build  
2 | - deploy      globally  
3 variables:  
4 | image_repository: docker.io/my-docker-id/myapp  
5 | image_tag: v1.0  
6 run_unit_tests:  
7 | stage: test
```

Where are the tasks executed?

Gitlab architecture:

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

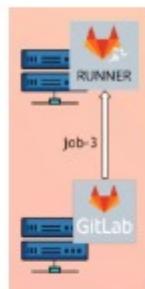
- main component
- pipeline configuration
- manages the pipeline execution
- stores the pipeline results

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-managed

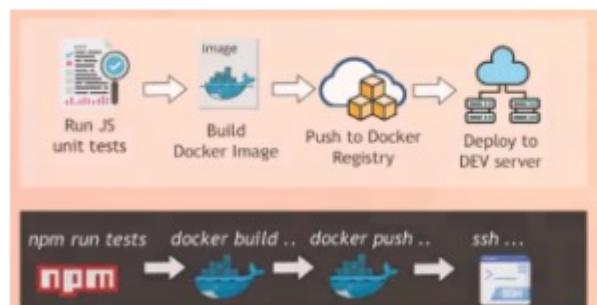


RUNNER:

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 platform



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 to be 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 difficulties

----> Limitation 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 separate and isolated container**

SO :

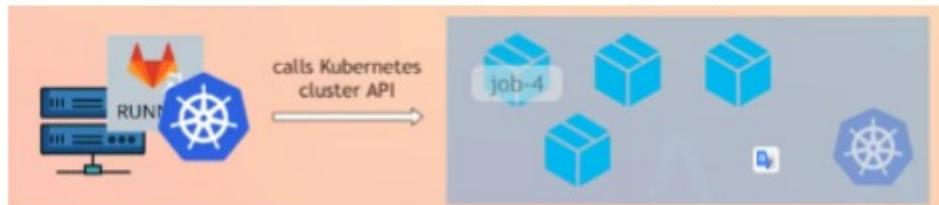
-> All tools need can be put in the docker image, no version conflicts. clean state of each job

executor 3

Kubernetes Executor

Allows you to use an existing kubernetes cluster for your builds: utilize your high availability setup.

Kubernetes executor will create a new **pod** for each gitlab job



Docker machine executor: [Managing layer over Docker]

special version of docker executor: support auto-scaling

lets you create docker hosts on your computer, on cloud providers on demand, dynamically

creates servers, installs Docker on them and configures the client to talk to them

Docker machine executor works like the normal docker executor

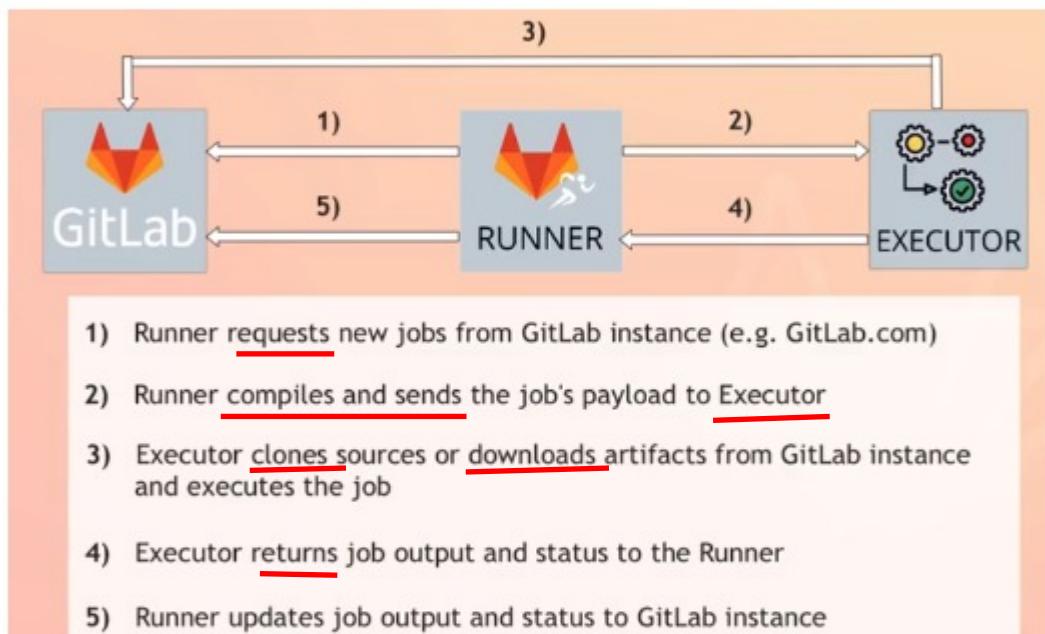
How to select:

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 pipeline logic
- easy

Globally specify image

方法1

The screenshot shows a pipeline editor interface with a toolbar at the top. Below the toolbar, there is a code block representing a pipeline configuration. The first line of the configuration is 'image: node:18-alpine', which is highlighted with a red box.

```
1 image: node:18-alpine
2
3 workflow:
4   rules:
```

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 status
 - 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
```

```
WARNING: The 'user' mode is not supported for non-root users.
(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/): ← job run in the gitlab-runner
https://gitlab.com/
Enter the registration token: ←
GR1348941TPCAFF4E5HT62rt677M8
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!
```

Runners

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

Register as many runners as you want. You can register runners as specific to a project or global.

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

AWS:

- default root user und unlimited privileges
- -> create an admin user

 IAM dashboard to create an adminuser, which can manage the users

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 **Zugriffsschlüssel – Programmgesteuerte Zugriff**

auswählen*

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 könnenKonsolenpasswort* Automatisch generiertes Passwort Benutzerdefiniertes Passwort

Zurücksetzen des Passworts erforderlich 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](#)[Filtrerrichtlinien](#) ▾ Suchen

755 Erg

	Richtlinienname ▾	Typ	Verwendet
<input checked="" type="checkbox"/>	AdministratorAccess	Auftragsfunktion	Keine
<input type="checkbox"/>	AdministratorAccess-Amplify	AWS-verwaltet	Keine
<input type="checkbox"/>	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 networking and content delivery.
 - VPC for each region
 - Region has multiple availability zones
 - AZ: 1 or more discrete data
 - private network is isolated from others
- virtual **representation** of network infrastructure
 - setup of servers network configuration moved to cloud
 - your components have to run in an VPC
 -

Bigger corporation: -> multiple VPCs

Subnet: subnet for each availability 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

- Internet Gateway connects the VPC to the outside internet



Controlling access

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

External access:

- configure external access such as SSH
- configuration access on subnet level => NACL
- configure firewall rules in the virtual server level using security groups

This screenshot shows the AWS Management Console with the 'Services' dropdown open, highlighting 'VPC'. The left sidebar shows 'New VPC Experience' and various VPC-related services like Egress Only Internet Gateways, DHCP Options Sets, and Elastic IPs. The main content area displays a list of subnets under an 'Availability Zone'.

Security

- Configure access on **subnet level => NACL**
- Configure access on **instance level => Security Group**
- Create on **VPC level** and assign to subnet and instance

Region

Your VPC

Security Group

Network Access Control List (NACL)



Instance -> Verbinden

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

```
(base) gu@gu-GE60-2PC:~/Downloads$ 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

```
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 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)

 * 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/repositories/gitlab-runner/main/ubuntu/amd64.deb"
% Total    % Received % Xferd  Average Speed   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
```

```
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
The following NEW packages will be installed:
```

Check the version

```
INFO: DOCKER_CLIENT_VERSION not found, skipping check
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:~$
```

Available specific runners

 #16218104 (Nudsg1ms)  	 Remove runner
ec2-runner	
  	
 #16216999 (86v619NM)  	 Remove runner
local-runner	
 	

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

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

```
1 Running with gitlab-runner 15.1.0 (76984217)  
2 on ec2-runner Nudsg1ms  
3 Preparing the "shell" executor  
4 Using Shell executor...  
5 Preparing environment  
6 Running on ip-172-31-46-28...  
7 Getting source from Git repository  
8 Fetching changes with git depth set to 20...  
9 Initialized empty Git repository in /home/gitlab-runner/builds/N  
s/mynodeapp-cicd-project/.git/  
10 Created fresh repository.  
11 Checking out 7bfcc480f as main...  
12 Skipping Git submodules setup  
13 Executing "step_script" stage of the job script  
14 $ echo "Preparing tet data..."  
15 Preparing tet data...  
16 $ echo "Running tests for micro service $MICRO_SERVICE_NAME"  
17 Running tests for micro service shopping-cart  
18 $ npm version  
19 bash: line 133: npm: command not found  
20 Running after_script  
21 $ echo "Clean up temporary files..."  
22 Clean up temporary files...  
23 $ rm -r test-data  
24 rm: cannot remove 'test-data': No such file or directory  
25 Cleaning up project directory and file based variables
```

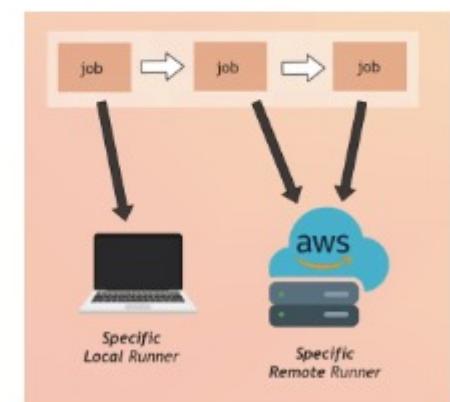
connect ec2

```
(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
```

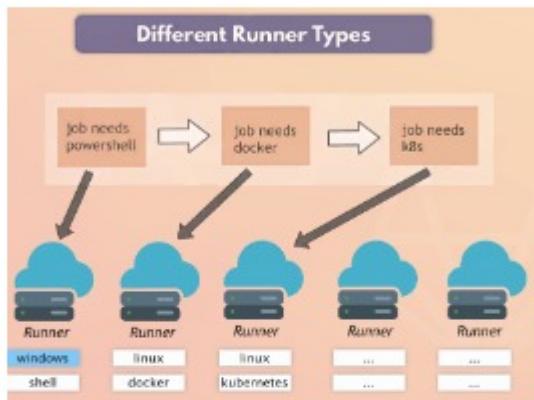
Distribute jobs in different runners

```
run_lint_tests:  
  tags:  
    - ec2  
    - remote  
    - aws  
  stage: test  
  # prepare env datasets  
  before_script:  
    - echo "Preparing tet data..."  
  script:  
    # script needs a list, using hyphen  
    - 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"
```



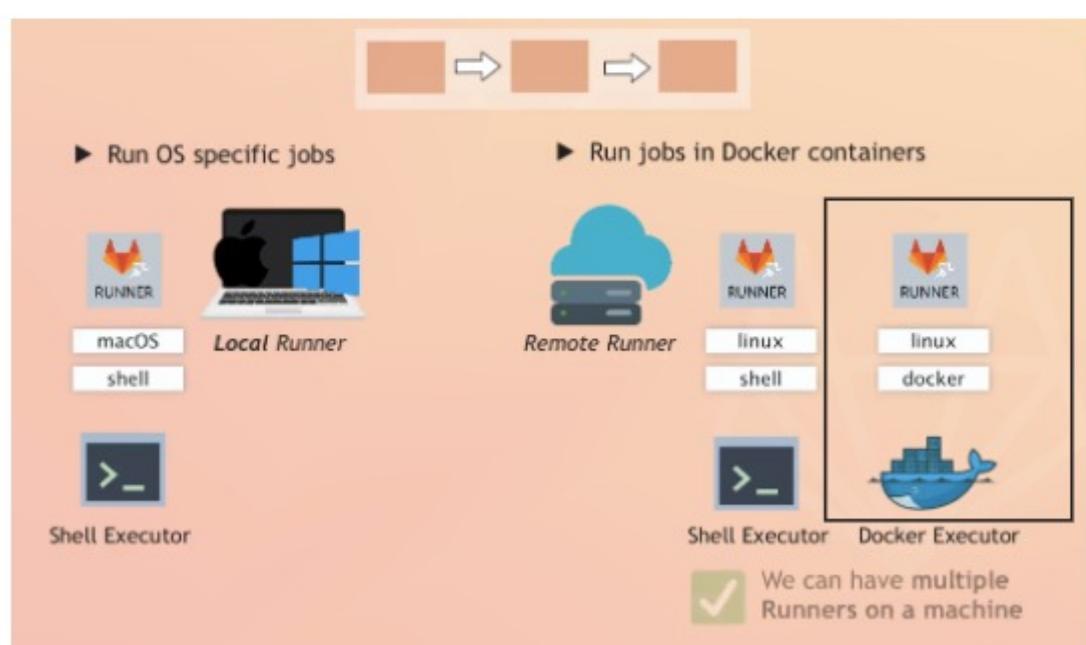


Usually you have two or more runners of the same type with same tags and job that references those tags will be executed on one of the runners



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

the docker is running

Add ubuntu user to docker group:

```
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
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, docker-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 already the config should be automatically reloaded!
ubuntu@ip-172-31-46-28:~$
```

The screenshot shows the GitLab interface with several sections:

- Available specific runners:** A list of registered runners:
 - #16264592 (WcFGaXsg) - ec2-docker-runner (tags: docker, ec2, remote)
 - #16218104 (Nudsg1ms) - ec2-runner (tags: aws, ec2, remote)
 - #16216999 (B6v619NM) - local-runner (tags: linux, local)
- Job Log:** A detailed log of a runner's activity, with steps numbered 1 through 17. Red arrows point from specific lines in the log back to the runner configuration in the UI:
 - Step 2: "Running with gitlab-runner 15.1.0 (76984217) on ec2-docker-runner WcFGaXsg"
 - Step 4: "Preparing the 'docker' executor"
 - Step 5: "Using Docker executor with image node:17-alpine3.14 ..."
 - Step 6: "Pulling docker image node:17-alpine3.14 ..."
 - Step 7: "Preparing environment"
 - Step 8: "Running on runner-wcfgaxsg-project-37391294-concurrent-0 via ip-172-31-46-28"
 - Step 9: "Getting source from Git repository"
 - Step 10: "Fetching changes with git depth set to 20..."
 - Step 11: "Initialized empty Git repository in /builds/Yingtremendous/mynod...
 - Step 12: "Created fresh repository."
 - Step 13: "Checking out 112c55f4 as main..."
 - Step 14: "Skipping Git submodules setup"
 - Step 15: "Executing 'step_script' stage of the job script"
 - Step 16: "Using docker image sha256:b20b24e39dda538a41dfa3e9fc7d70479cad945 for node:17-alpine3.14 with digest node@sha256:0d8276c8e82fa7115f9d04acbe8dd16a850f783c ..."
 - Step 17: "echo \"Preparing test data...\""
- Job Script:** A snippet of a job script with annotations:

```
run_unit_tests:
  image: node:17-alpine3.14
  tags:
    - ec2
    - docker
    - remote
  stage: test
  # prepare env datasets
  before_script:
    - echo "Preparing test data..."
```

Annotations highlight the 'image' field and the 'tags' section.
- Text Overlay:** The text "using docker executor and need image for node" is overlaid on the UI.

runners assigned to that specific project

You can unlock and assign runner to other projects as well

Runner #16264592 specific

This runner is associated with specific projects.
You can set up a specific runner to be used by multiple projects but you cannot make this a shared runner. Learn more.

Active	<input checked="" type="checkbox"/> Paused runners don't accept new jobs
Protected	<input type="checkbox"/> This runner will only run on pipelines triggered on protected branches
Run untagged jobs	<input type="checkbox"/> Indicates whether this runner can pick jobs without tags
Lock to current projects	<input checked="" type="checkbox"/> When a runner is locked, it cannot be assigned to other projects
IP Address	3.127.152.163
Description	ec2-docker-runner
Maximum job timeout	

Use case: online shop, a lot of microservices (shipping-app, billing-app...), separate git project for each microservice . 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 use groups to manage permission: if someone has access to the group, they get access to all the projects in the group

The screenshot shows the process of creating a new group named "my nice online shop". A success message says "Group 'my nice online shop' was successfully created." Below the message, there's a summary card for the group "my nice online shop" with a Group ID of 51217954. It includes buttons for "New subgroup" and "New project". At the bottom, there are sections for "Subgroups and projects", "Shared projects", "Archived projects", and search fields for "Search by name" and "Name".

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(**specific** runner, **group** runner, **shared** runner) -> You can install and register a shared runner which is available to every project in the gitlab instance

Self-Managed GitLab Instance

- E.g. for security reasons
- Have unused servers you want to utilize
- Have full control over your infrastructure
- Save infrastructure costs

The diagram illustrates a self-managed GitLab instance. It features a central "GitLab" logo with the URL "www.gitlab.mycompany.com". To the left, a "Self-Managed GitLab Instance" box contains a building icon. To the right, a shield icon with a lock symbol represents security. In the center, two application icons labeled "app-1" and "app-2" are connected by arrows, indicating a network or dependency structure between them.

Gitlab runner versions



? sync and compatible ?



2 different GitLab Setups

SaaS - GitLab's Instance



Self-Managed GitLab Instance

www.gitlab.mycompany.com



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

- ▶ 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 security testing: analyzes code itself without actually executing the code
It scans your code for any security 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
1 workflow:
2   rules:
3     - if: $CI_COMMIT_BRANCH != "main" && $CI_PIPELINE_SOURCE != "merge_request_event"
4       | when: never
5       - when: always
6
7
8
9
10 run_unit_tests:
11   image: node:17-alpine3.14
12   tags:
13     - ec2
14     - docker
15     - remote
16   before_script:
17     - cd app
18     - npm install
19   script:
20     - npm test

```

```
1 Running with gitlab-runner 15.1.0 (7098421)
2 on ec2-docker-runner WcFGaXsg
3 Preparing the "docker" executor
4 Using Docker executor with image node:17-alpine3.14 ...
5 Pulling docker image node:17-alpine3.14 ...
6 Using docker image sha256:b20b24e39dda538a41dfa3e9fcf7d70479cad96e3aa7324a0fc7fd1eacd8de
7 45 for node:17-alpine3.14 with digest node@sha256:0d8276c8e82fa717a9a88b8734bbad60ac29a0f
8 15f9d04acbe8dd16a850f783c ...
9 Preparing environment
10 Running on runner-wcfgaxsg-project-37391294-concurrent-0 via ip-172-31-46-28...
11 Getting source from Git repository
12 Fetching changes with git depth set to 20...
13 Reinitialized existing Git repository in /builds/Yingtremendous/mynodeapp-cicd-project/...
14 git/
15 Checking out 3e2afdd4 as main...
16 Skipping Git submodules setup
17 Executing "step_script" stage of the job script
18 Using docker image sha256:b20b24e39dda538a41dfa3e9fcf7d70479cad96e3aa7324a0fc7fd1eacd8de
19 45 for node:17-alpine3.14 with digest node@sha256:0d8276c8e82fa717a9a88b8734bbad60ac29a0f
20 15f9d04acbe8dd16a850f783c ...
21 $ cd app
22 $ npm install
23 npm WARN deprecated resolve-url@0.2.1: https://github.com/lydell/resolve-url#deprecated
24 npm WARN deprecated urix@0.1.0: Please see https://github.com/lydell/urix#deprecated
25 added 557 packages, and audited 558 packages in 24s
26 24 packages are looking for funding
27   run `npm fund` for details
28 4 vulnerabilities (2 moderate, 1 high, 1 critical)
29 To address all issues, run:
30   npm audit fix
31 Run `npm audit` for details.
32 $ npm test
33 > bootcamp-node-project@1.0 test
34 > jest --ci --reporters=default --reporters=jest-junit
35 Browserslist: caniuse-lite is outdated. Please run:
36 npx browserslist@latest --update-db
37 Why you should do it regularly:
38 https://github.com/browserslist/browserslist#browsers-data-updating
39 PASS ./server.test.js
40   ✓ main index.html file exists (4 ms)
41   ✓ Dockerfile exists
42   ✓ .gitignore file exists (1 ms)
43 Test Suites: 1 passed, 1 total
44 Tests:       3 passed, 3 total
45 Snapshots:   0 total
46 Time:        1.707 s
47 Ran all test suites.
48 Cleaning up project directory and file based variables
49 Job succeeded
```

Configure test results

No test results and no artifacts

The screenshot shows a pipeline interface for a project named 'main'. A single job named 'main' has failed after 35 seconds (queued for 51 seconds). The status bar at the bottom indicates 'Pipeline Needs Jobs 1 Tests 0'. A large central icon features a clipboard with several checkmarks and a thumbs-up icon. Below the icon, the text reads: 'There are no test reports for this pipeline'. A link 'Learn how to upload pipeline test reports' is provided.

Job can output an archive of files and directories. -> It called a job artifact
JUnit was originally 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
```

### Update .gitlab-ci.yml file

1 job for main in 37 seconds (queued for 51 seconds)

latest

d2b4544d

No related merge requests found.

Pipeline Needs Jobs 1 Tests 3

### Summary

3 tests

0 Failures

0 errors

### Jobs

| Job            | Duration | Failed |
|----------------|----------|--------|
| run_unit_tests | 5.00ms   | 0      |

Organize your test results use paths properties

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

The screenshot shows a CI pipeline interface. At the top, there's a header with a profile picture, a search bar, and some navigation links. Below the header, the main title is "Update README.md". Under this, there's a summary section with the following details:

- 1 job for feature/refactor in 41 seconds (queued for 46 seconds)
- latest merge request
- 4f3a5c0e
- 1 related merge request: 13 Did refactor

Below the summary, there are tabs for Pipeline, Needs, Jobs (1), Failed Jobs (1), Tests (3), and a selected tab labeled Tests 3. The Tests 3 tab shows a summary of the test results:

| 3 tests        | 2 Failures | 0 errors |
|----------------|------------|----------|
| run_unit_tests | 7.00ms     | 2        |

At the bottom of the interface, there are several large, light-gray buttons with text that is mostly illegible due to the image quality.

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

#### Container Registry

#### Infrastructure Registry

## 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 **authenticate** first

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

```
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
```

## Authentication via GitLab CI/CD



Pre-defined CI/CD variables

CI\_REGISTRY\_USER

CI\_REGISTRY\_PASSWORD

- GitLab provides **temporary credentials** for the Container Registry in your CI/CD pipeline
- Read-write access to the Container Registry

CI\_REGISTRY\_USER

CI\_REGISTRY\_PASSWORD The values of these are only valid for 1 job only

```
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
```

必须写的非常的准确

lowercase u and p for user and password

```
tags:
 - ec2
 - shell
 - remote
before_script:
 - docker login -u $CI_REGISTRY_USER -P $CI_REGISTRY_PASSWORD registry.gitlab.com
script:
 - docker push registry.gitlab.com/yingtremendous/mynodeapp-cicd-project:1.0
```

## Multiple image repositories

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



## Give images name

```
38
39 build_image:
40 stage: build
41 tags:
42 - ec2
43 - shell
44 - remote
45 script:
46 - docker build -t $CI_REGISTRY_IMAGE/microservice/payment:1.0 .
47
48 push_image:
49 stage: build
50 needs:
51 - build_image
52 tags:
53 - ec2
54 - shell
55 - remote
56 before_script:
57 - echo "$CI_REGISTER_USER $CI_REGISTRY_PASSWORD"
58 - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD registry.gitlab.com
59 script:
60 - docker push $CI_REGISTRY_IMAGE/microservice/payment:1.0
```

## Parametrize 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

X

#### Key

#### Value

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAQCAQEA61ccpCuSLmrjvpHyNFgAgY0SEW2XCDJLhZy0PigNkVeZc0Ho
gDvdfpKpH1f5EPjQQn6Bz3znKK+fNbZ017k6jlEUArIU99jDFgXECEKPIytdH6aI
tRqIISYBv1vWFORWYb41ADNpqVx82aN05/WYMOwW7e6pGD6/PwlCfmIgighbjvv6
WEenFCi/iCpKL0YwjULAFc08jWa/oSmcVa6ix1djbyXylKP5+JQmFRHbnRq5uF6vW
Iz07+reXTob8k3RoFEaY11eAR02DA8Br73VRDrWPkywqCAND4x/xG+i/d8Pi6hnz
7RR4aaxc14ToSiRRcfuGrDhzplck/RUuHylQWwTDA0APRAcTRADLUNEWY1fils0nH
```



#### Type

 Variable 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](#)

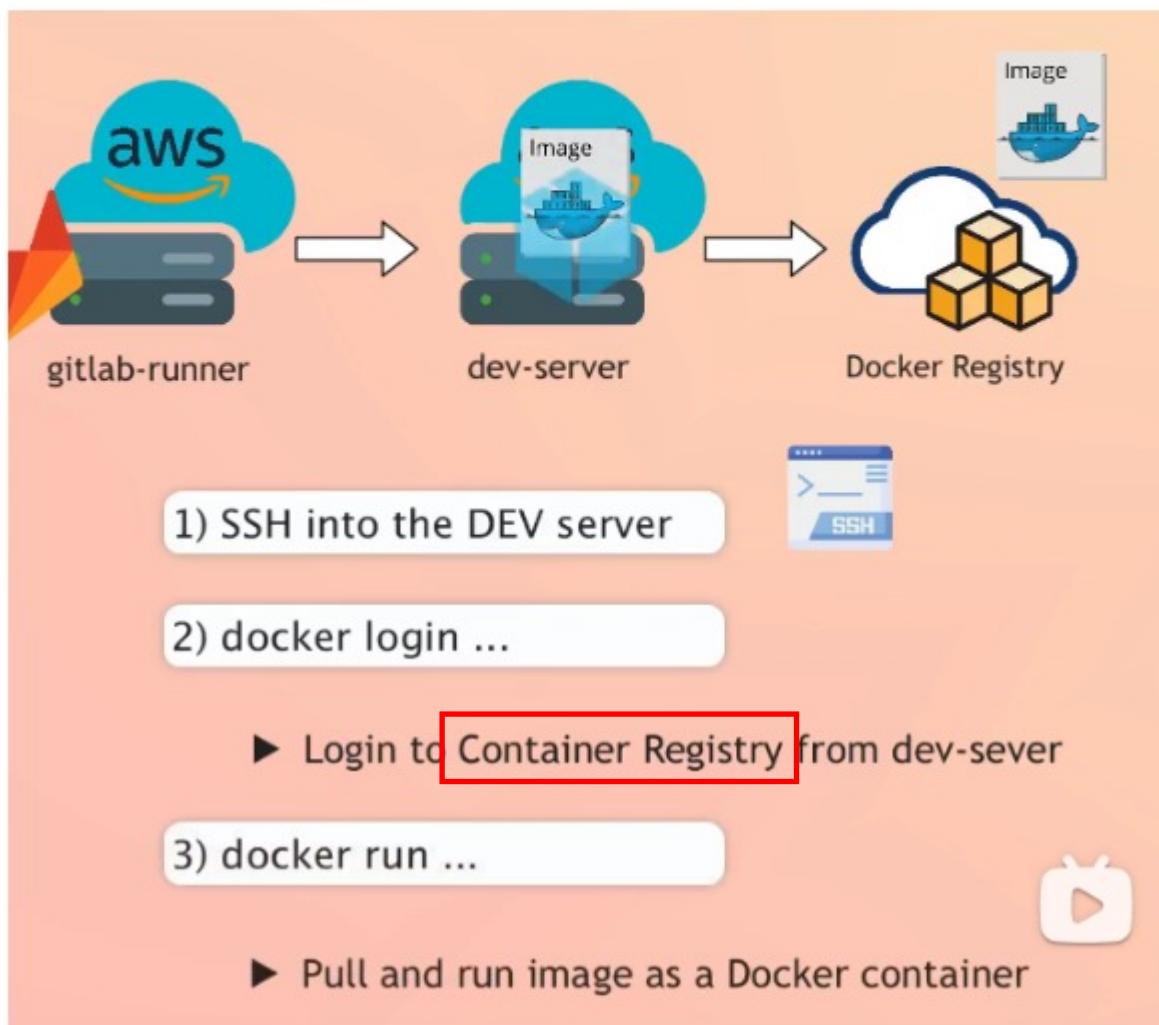
#### Environment scope ?

 All (default)

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:
 deploy
 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: Andreea Efti

Title: User Experience

Project: Staff migration, Travel Automation, GIGI



Name: Ari Balter

Title: Software developer

Project: Dallas Shop, KEXP Volunteer

## Deploy a microservice to my backend

gitlab.com/microservices/microservice-movieapi

| Microservice      | Path              | Code                           | Tests                          | Metrics                        | Package                        | Service                        | Release                        | Logs                           | Activity                       | Details                        |
|-------------------|-------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|
| Movie API         | movie-api         | microservice-movieapi          |
| Auth Service      | auth-service      | microservice-auth-service      |
| Rating Service    | rating-service    | microservice-rating-service    |
| Review Service    | review-service    | microservice-review-service    |
| Profile Service   | profile-service   | microservice-profile-service   |
| Task Service      | task-service      | microservice-task-service      |
| Rating Migration  | rating-migration  | microservice-rating-migration  |
| Review Migration  | review-migration  | microservice-review-migration  |
| Profile Migration | profile-migration | microservice-profile-migration |
| Task Migration    | task-migration    | microservice-task-migration    |

go to deployment -> Environment

```
DOCUMENTATION > 3. DEPLOYMENT > ENVIRONMENTS
ENVIRONMENT:
NAME: development
URL: $DEV_DOMAIN
```

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

## CD with Docker-Compose

Docker run will always work 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

- Use Docker Compose to run multiple containers



docker-compose.yml

- Docker Compose is used to define and run multiple containers

- Everything is defined in one YAML file

- You can spin up, bring up and tear it all down with just a single command



create and write a new file docker-compose

```
{ } docker-compose.... ④
1 version: "3.3"
2 services:
3 app:
4 images: registry.gitlab.com/yingtremendous/mynodeapp-cicd-project
5 ports:
6 - 3000:3000
7
```

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

```
deploy to dev:
stage:
| depoly
tags:
| - ec2
| - shell
| - remote
before_script:
| - chmod 400 $SSH_PRIVATE_KEY
script:
| - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./docker-compose.yaml ubuntu@$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-compose.yaml down && |
| docker-compose -f docker-compose.yaml up -d"
environment:
| name: development
url: $DEV_ENDPOINT
```

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

lsudo 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 optimierun
```

```
8 before_script:
9 - chmod 400 $SSH_PRIVATE_KEY
10 script:
11 - scp -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./docker-compose.yaml ubuntu@$DEV
12 ubuntu
13 - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubuntu@$DEV_SERVER_HOST "
14 docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY &&
15 docker-compose down &&
16 docker-compose up -d"
17 environment:
18 name: development
19 url: $DEV_ENDPOINT
20
```

## locker-compose.yaml

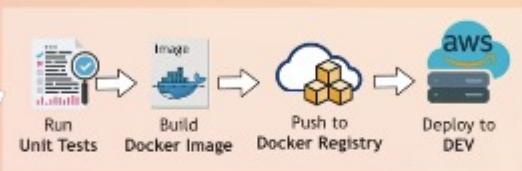


123 bytes

```
1 version: "3.3"
2 services:
3 app:
4 image: ${DC_IMAGE_NAME}:${DC_IMAGE_TAG}
5 ports:
6 - 3000:3000
7
8 --
9 - shell
10 - remote
11 before_script:
12 - chmod 400 $SSH_PRIVATE_KEY
13 script:
14 - scp -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./docker-compose.yaml ubuntu@$DEV_SERVER_HOST:/home/
15 ubuntu
16 - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubuntu@$DEV_SERVER_HOST "
17 docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY &&
18 export DC_IMAGE_NAME=$IMAGE_NAME
19 export DC_IMAGE_TAG=$IMAGE_TAG
20 docker-compose down &&
21 docker-compose up -d"
22 environment:
23 name: development
24 url: $DEV_ENDPOINT
25
```

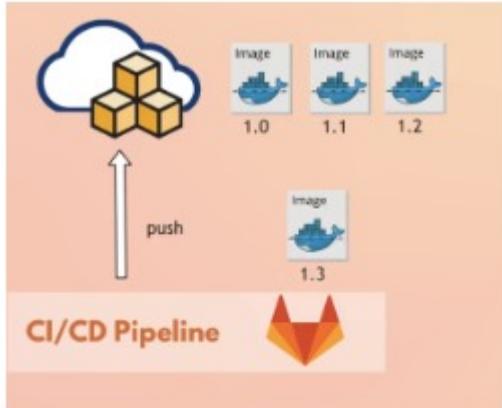
## Extend & Optimize Pipeline to a Real Life Pipeline

node.js



- **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



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
Building dependency tree
```

test jq command line

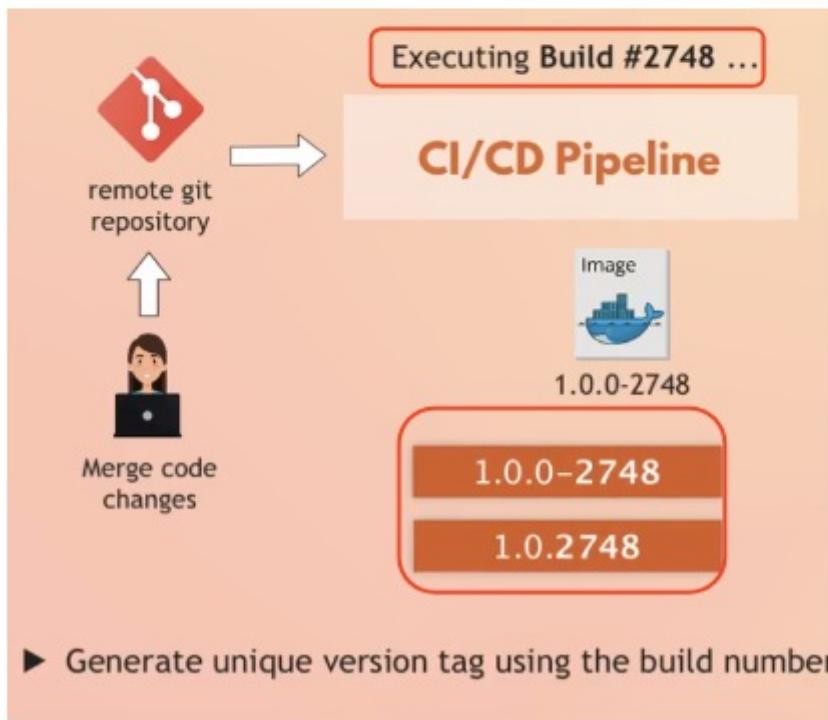
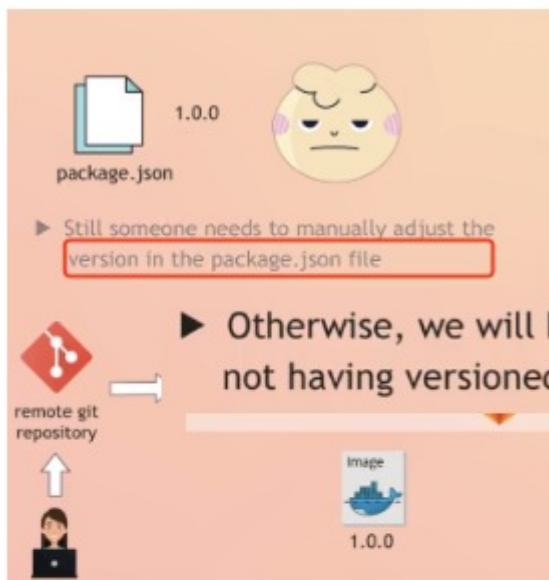
```
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.9) ...
ubuntu@ip-172-31-46-28:~$ vim package.json
ubuntu@ip-172-31-46-28:~$ cat package.json | jq -r .version
1.0
```

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



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

# Use artifacts in the other jobs

## CI/CD Pipeline

build\_image



1.0.0-2748

VERSION

push\_image



get version also for push

VERSION

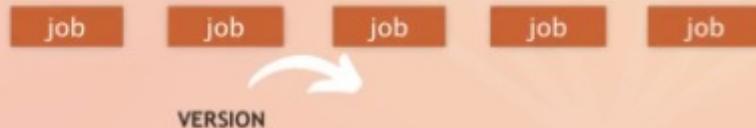


- ▶ Variable is only available in the job it was defined = **job environment**
- ▶ Every job gets executed in its own new environment

How do we pass data  
between jobs?



## CI/CD Pipeline



## 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



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



## version-file.txt persists by artifacts paths properties

```
script:
 - remote
before_script:
 - export PACKAGE_JSON_VERSION=$(cat app/package.json | jq -r .version)
 - export VERSION=$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

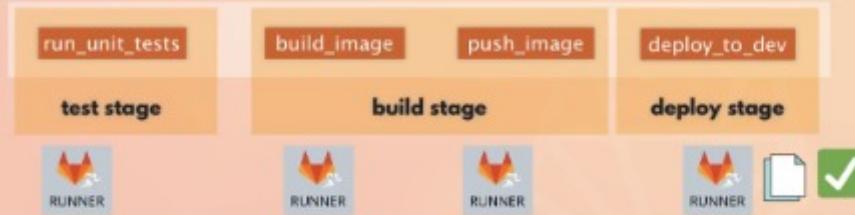
```
push_image:
 stage: build
 needs:
 - build_image
 tags:
 - ec2
 - shell
 - remote
 before_script:
 - export VERSION=$(cat version-file.txt)
 - echo "$CI_REGISTER_USER $CI_REGISTRY_PASSWORD"
 - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD
 script:
```

- ▶ 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

### dependencies



for same stage

### dependencies



artifact from  
build\_image

dependencies:  
- build\_image

build\_image

push\_image

build stage



### Same Stage

- ▶ Use "dependencies" attribute to define a list of jobs to fetch artifacts from

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



GitLab

version-file.txt

build\_image



version-file.txt

► Artifact will be uploaded & persisted on the GitLab instance



version-file.txt



RUNNER

push\_image



version-file.txt

# Difference between need and dependencies

## needs vs dependencies



build\_image



push\_image

**build stage**

### 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 **not need dependencies if you have needs**
- ▶ No need to define "*dependencies*" separately

## 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

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

```
lab-ci.yml > {} test_dev > [] depende
 docker-compose d
 docker-compose u
environment:
 name: development
 url: $DEV_ENDPOINT

test_dev:
 stage: deploy
 dependencies:
 - run_unit_tests
 script:
 - echo "testing"
```

## Restrict artifact download

 Download only the artifacts, you actually need in the job

- Repository
- Issues 0
- Merge requests 1
- CI/CD
- Security & Compliance
- Deployments
- Monitor
- Infrastructure
- Packages & Registries
  - Package Registry
  - Container Registry
  - Infrastructure Registry
- Analytics

Filter results

5 tags

1.0

56.51 MiB

1.0.121

56.52 MiB

1.0.122

56.52 MiB

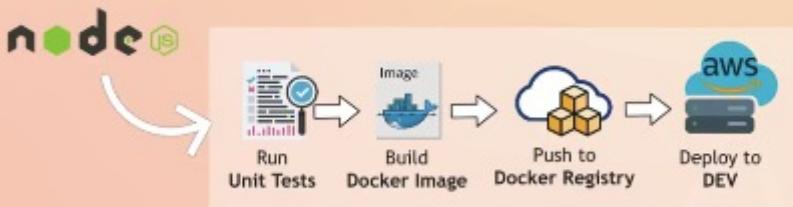
1.1

56.51 MiB

1.2



## Extend & Optimize Pipeline to a Real Life Pipeline



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



v1.0



v1.1



v2



## 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**

```
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_PASSWORD"
 - 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



- ▶ Each job runs in its own isolated environment



job\_B

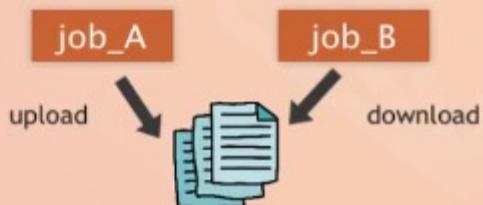
- ▶ Starting from a fresh new environment

- Jobs don't affect each other
- No unexpected side-effects

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

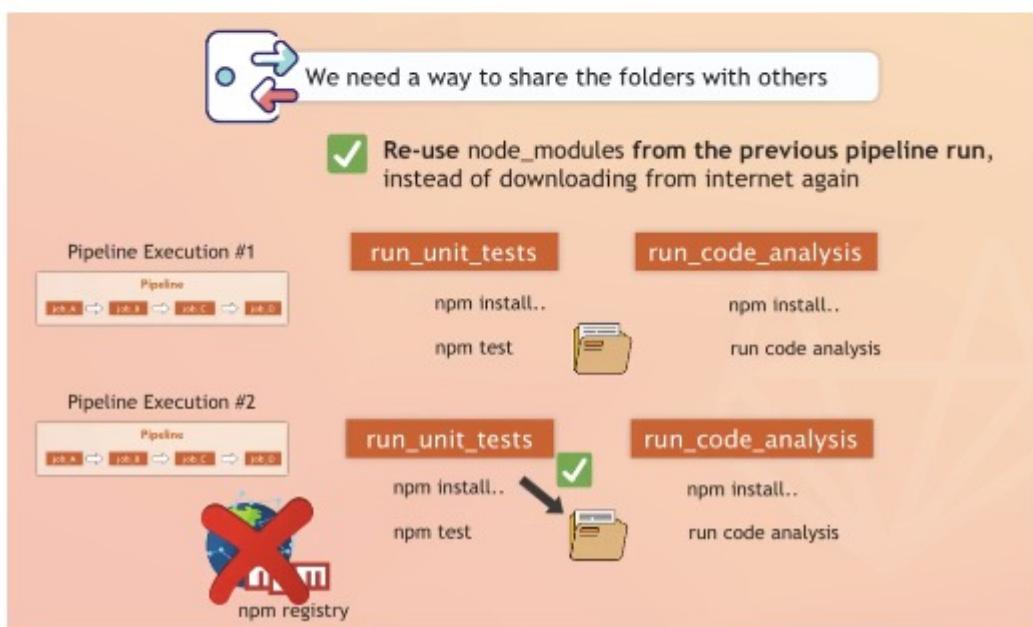
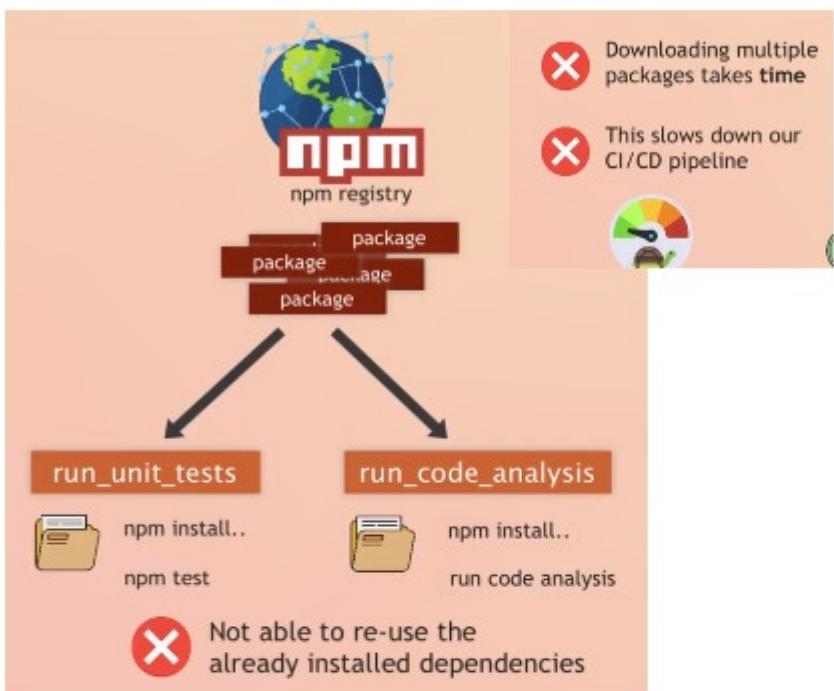
#### Artifacts

- 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



```
MYNODE... E+ E- ⌂ ⌂ @ 7
 app 8
 > Images 9
 > node_modules 10
 > index.html 11
 > jest.xml 12
 package-lock.json 13
 package.json 14
 server.js 15
 server.test.js 16
 .gitignore }
```

"test": "jest --ci --reporters=default --reporters=jest-junit",  
"start": "node server.js"  
},  
"author": "NJ",  
"license": "ISC",  
"dependencies": {

Uploading/Downloading all dependencies from  
GitLab server wouldn't be a big improvement

reuse between pipeline runs

## Caching to re-use between pipeline runs

Pipeline Execution #1



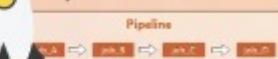
Pipeline Execution #2



Pipeline Execution #3



Pipeline Execution #4



- Speed up pipeline execution
- Saves CI costs

git cache

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

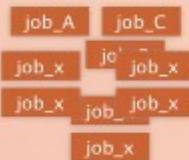
- ▶ Use cache for dependencies, like packages you download from the internet
- ▶ Cache is stored on the GitLab Runner!



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

## Distributed Cache

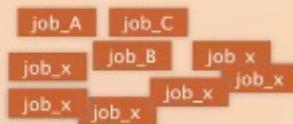
### Distributed Cache



cache ZZZ



cache ZZZ



solution distributed cache

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



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 pipeline

## Configure a Cache

Configure in the .gitlab\_ci.yml file



Using "cache" attribute

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

You can define a cache for each job

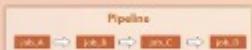
# Cache Key

- ▶ Give each cache a **unique identifying 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



main



dev



cache\_main

cache\_dev



- ▶ 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

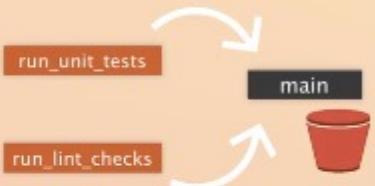
```
paths:
 - app/node_modules
policy: pull-push
```

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



- ▶ Jobs executing in parallel using the same cache
- ▶ Updating the same cache in 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 or pass the cache between jobs

configure volume for docker executor

job cached in the docker container with docker executor

### Cache with Docker Runner



▶ Job gets executed inside a new Docker container

▶ Cache gets created inside the container

Docker Runner



▶ Docker: file system of the container



▶ Shell: file system of the server

## 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



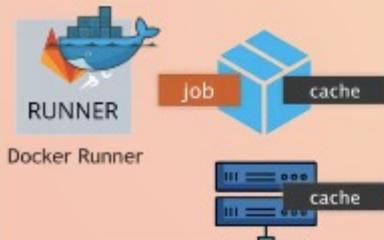
- ▶ Files are only stored inside the container, but not on the host (server)



### Configure a Docker Volume

- ▶ Volumes persist data generated by containers on the host machine

## Cache with Docker Runner



```
ubuntu@ip-172-31-22-114:~$ sudo ls /etc/gitlab-runner/
config.toml
ubuntu@ip-172-31-22-114:~$ sudo vim /etc/gitlab-runner/config
```

### config.toml

- Can be used to change the behavior of GitLab Runner

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
```

```
[[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
```

```
 skipping Git submodules setup
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/lydtknudsen/urix
28 npm WARN deprecated resolve-url@0.2.1: https://github.com/lydtknudsen/resolve-url
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
58 Uploading artifacts for successful job
59 Uploading artifacts...
60 app/junit.xml: found 1 matching files and directories
```

```
0 Checking cache for main-3...
1 No URL provided, cache will not be downloaded from share
local version of cache will be extracted.
2 Successfully extracted cache
4 Executing "step_script" stage of the job script
5 Using docker image sha256:af611d49779a0d9371d6a4101d750
18a50d699 for node:17-alpine3.14 with digest node@sha256:
513b0e591c89e08a0b1d0c9bf23f9284c7611603 ...
6 $ cd app in lint not download anymore
7 $ npm install
8 up to date, audited 558 packages in 3s
9 24 packages are looking for funding
```

```
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 cache

> mynodeapp-cicd > Pipelines

Finished Branches Tags

Clear runner caches

CI lim

Pipelines



Show

Pipeline

Triggerer

Stages

### 2 ways to clear the cache

- ▶ Manually with this button
- ▶ Change the value for the cache key in yml file



cache-1



cache-2



GitLab Runner

### Clearing the cache manually

- ▶ The old cache is not deleted
- ▶ Instead it cache name will be updated and used instead
- ▶ You can manually delete the files from the Runner storage

## 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

```
4 Checking out 23clf581 as main...
5 Removing app/node_modules/
6 Skipping Git submodules setup
8 Restoring cache
9 Checking cache for main-2-protected...
0 No URL provided, cache will not be downloaded from s
version of cache will be extracted.
1 Successfully extracted cache
3 Executing "step_script" stage of the job script
4 Using docker image sha256:b20b24e39dda538a41dfa3e9fc
45 for node:17-alpine3.14 with direct nodeCache256MB
```

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
ubuntu@ip-172-31-22-114:~$
```

```
runner-qhm8c6bn-project-34530931-concurrent-0-cache-c33bca
172-31-22-114:~$ docker volume inspect runner-qhm8c6bn-pro
-3c3f060a0374fc8bc39395164f415a70
```

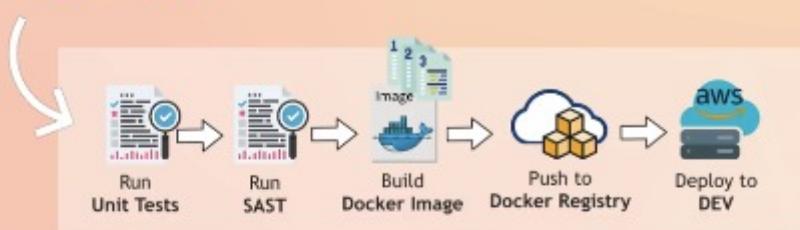
```
"com.gitlab.gitlab-runner.runner.local_id": "0",
"com.gitlab.gitlab-runner.type": "cache"
},
"Mountpoint": "/var/lib/docker/volumes/runner-qhm8c6bn-project-34530931-concurrent-0-cache-3c3f060a0374fc8bc39395164f415a70/_data",
"Name": "runner-qhm8c6bn-project-34530931-concurrent-0-cache-3c3f060a0374fc8bc39395164f415a70",
"Options": null.
```

```
ubuntu@ip-172-31-22-114:~$ sudo ls /var/lib/docker/volumes/runner-qhm8c6bn-project-34530931-concurrent-0-cache-3c3f060a0374fc8bc39395164f415a70/_data
nanuchi
ubuntu@ip-172-31-22-114:~$ sudo ls /var/lib/docker/volumes/runner-qhm8c6bn-project-34530931-concurrent-0-cache-3c3f060a0374fc8bc39395164f415a70/_data/nanuchi
mynodeapp-cicd-project
ubuntu@ip-172-31-22-114:~$ sudo ls /var/lib/docker/volumes/runner-qhm8c6bn-project-34530931-concurrent-0-cache-3c3f060a0374fc8bc39395164f415a70/_data/nanuchi/mynodeapp-cicd-project
main-3 main-4
ubuntu@ip-172-31-22-114:~$
```

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

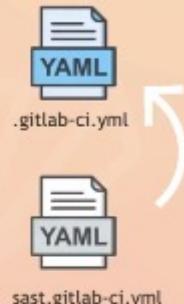
read gitlab ci/cd cache and artifact

## Include SAST job with job template



### Job Templates

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



### Auto DevOps



enough automated test unit test:

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

### functional testing

- validate the functionality of the software

### 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, ...

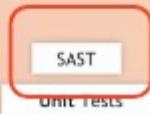
## DAST - Dynamic application security testing



- ▶ 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



Earlier

✗ More expensive to fix vulnerabilities

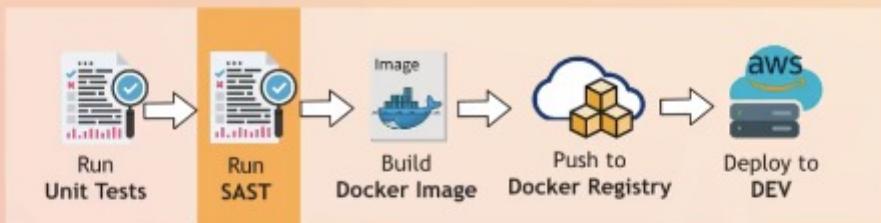


Later

- ▶ 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?



xxx.gitlab-ci.yml

- ▶ Job templates for different tasks and for different programming languages & technologies



SAST.gitlab-ci.yml



.gitlab-ci.yml

Out-of-the-box functionality!



## What is a CI/CD template?



\*.gitlab-ci.yml

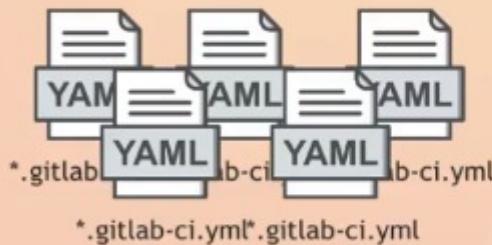


.gitlab-ci.yml

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 engineers wrote these job templates
- ▶ You can check out the available templates on [gitlab.com](https://gitlab.com)

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

python ML Y News TUM AI Handelsblatt... GitLab CI/CD... 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

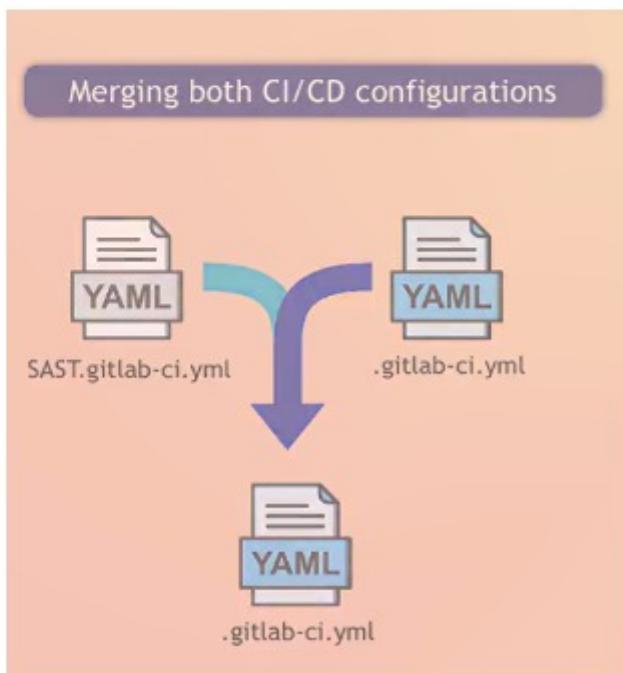
Snippets

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 |



Add latest changes from gitlab-org/gitlab@master  
GitLab Bot authored 2 months ago

SAST.gitlab-ci.yml 7.55 KiB

Edit

Lock

```
1 # Read more about this feature here: https://docs.gitlab.com/ee/user/application_security/sast/
2 #
3 # Configure SAST with CI/CD variables (https://docs.gitlab.com/ee/ci/variables/index.html).
4 # List of available variables: https://docs.gitlab.com/ee/user/application_security/sast/index.ht
5
6 variables:
7 # Setting this variable will affect all Security templates
8 # (SAST, Dependency Scanning, ...)
9 SECURE_ANALYZERS_PREFIX: "registry.gitlab.com/security-products"
10 SAST_IMAGE_SUFFIX: ""
11
12 SAST_EXCLUDED_ANALYZERS: ""
13 SAST_EXCLUDED_PATHS: "spec, test, tests, tmp"
14 SCAN_KUBERNETES_MANIFESTS: "false"
15
16 sast:
17 stage: test
18 artifacts:
19 reports:
20 sast: gl-sast-report.json
21 rules:
22 - when: never
23 variables:
24 SEARCH_MAX_DEPTH: 4
25 script:
26 - echo "$CI_JOB_NAME is used for configuration only, and its script should not be executed"
27 - exit 1
28
29 .sast-analyzer:
30 extends: sast
~~~
```

include sast template into our pipeline code

use feature include

## "include"

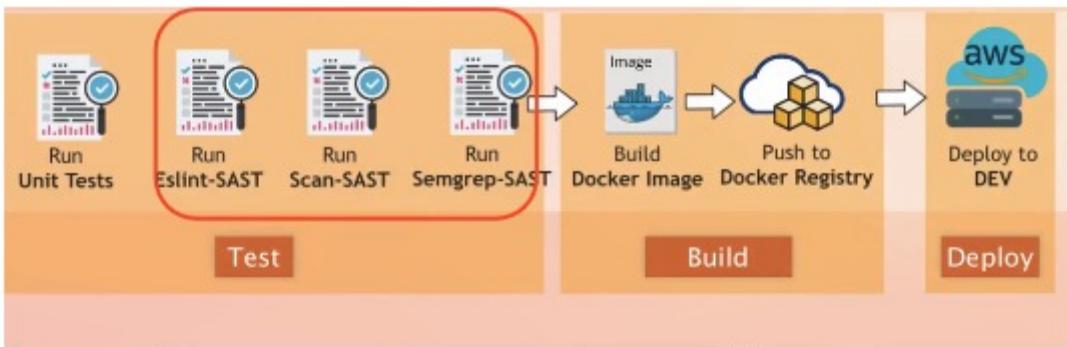
- ▶ Possible subkeys are:  
template, local, file, remote
- ▶ E.g. you can split one long .gitlab-ci.yml  
file into multiple files and "include" them

# "include"

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

```
90      docker-compose down &&
91      docker-compose up -d"
92      environment:
93          name: development
94          url: $DEV_ENDPOINT
95
96      include:
97          - template: Jobs/SAST.gitlab-ci.yml
98
99
```

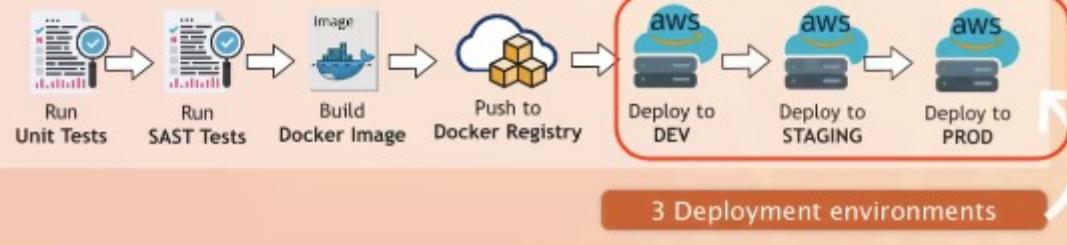
```
37      paths:
38          - app/junit.xml
39      reports:
40          junit: app/junit.xml
41
42      sast:
43          stage: test
44
45      build image:
```



## 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



## Deployment Environments

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



Release new app changes...



## Promote to Staging and Production



Release in Stages



- ▶ Functional tests
- ▶ Integration tests
- ▶ SAST tests
- ▶ Performance tests
- ▶ DAST tests
- ▶ not attackt

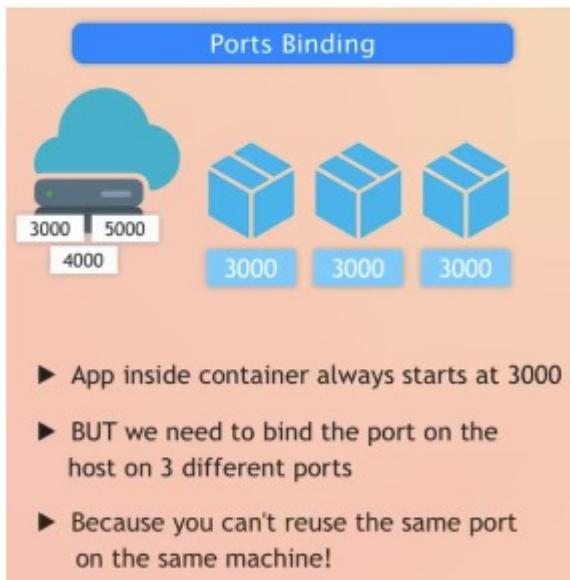




dev-staging-prod

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

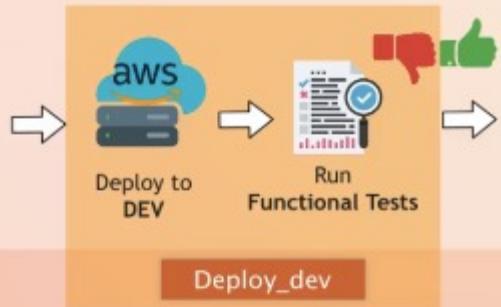
make host port configurable



- ▶ App inside container always starts at 3000
- ▶ BUT we need to bind the port on the host on 3 different ports
- ▶ Because you can't reuse the same port on the same machine!

er/compose.yaml / services / app / ports /

```
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

```

run_functional_tests:
  stage: deploy_dev
  needs:
    - deploy_to_dev
  script:
    - echo "running test"

deploy_to_staging:
  stage:
    deploy_staging
  tags:
    - ec2
    - shell
    - remote
  before_script:
    - chmod 400 $SSH_PRIVATE_KEY
  script:
    - scp -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ./docker-compose.yml ubuntu@$STAGING_ENDPOINT:~/app
    - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubuntu@$STAGING_ENDPOINT "cd ~/app && docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY && export DC_IMAGE_NAME=$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

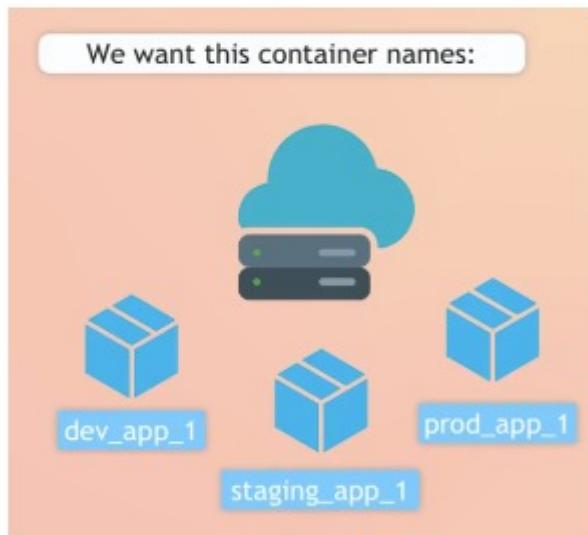
```
ubuntu@ip-172-31-36-31:~$ docker ps
CONTAINER ID IMAGE
CREATED STATUS PORTS
S
629c569218f3 registry.gitlab.com/nanuchi/mynodeapp-cicd-project:1.0.144 "doint.s."
3 minutes ago Up 3 minutes 0.0.0.0:3000->3000/tcp, :::3000->3000
tu_app_1
ubuntu@ip-172-31-36-31:~$ ls
docker-compose.yaml
ubuntu@ip-172-31-36-31:~$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

```
before_script:
  - chmod 400 $SSH_PRIVATE_KEY
script:
  - scp -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 && staging_app_1
    docker-compose up -d"
```

environment:

```
ubuntu@ip-172-31-43-110:~$ docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
91b3ee524944      registry.gitlab.com/yingtremendous/mynodeapp-ctcd-project:1.0.91   "docker-entrypoint.s..."   2 s
97ea21155915      registry.gitlab.com/yingtremendous/mynodeapp-ctcd-project:1.0.91   "docker-entrypoint.s..."   Abo
ut a minute ago
ubuntu@ip-172-31-43-110:~$
```

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

## 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**

## Regeln für eingehenden Datenverkehr bearbeiten [Info](#)

Regeln für eingehenden Datenverkehr steuern den eingehenden Datenverkehr, der die Instance erreichen darf.



### Regeln für eingehenden Datenverkehr [Info](#)

| ID der Sicherheitsgruppenregeln | Typ <a href="#">Info</a>              | Protokoll <a href="#">Info</a> | Portbereich <a href="#">Info</a> | Quelle <a href="#">Info</a>           | Beschreibung – optional <a href="#">Info</a> | Löschen                 |
|---------------------------------|---------------------------------------|--------------------------------|----------------------------------|---------------------------------------|----------------------------------------------|-------------------------|
| 59f-020924333cd437c43           | SSH                                   | TCP                            | 22                               | Benutzerdefinier... <a href="#">Q</a> | 0.0.0.0/0 <a href="#">X</a>                  | <a href="#">Löschen</a> |
| sgr-04fbefc02165e8515a          | Benutzerdefinier... <a href="#">Q</a> | TCP                            | 3000                             | Benutzerdefinier... <a href="#">Q</a> | 0.0.0.0/0 <a href="#">X</a>                  | <a href="#">Löschen</a> |
| -                               | Benutzerdefinier... <a href="#">Q</a> | TCP                            | 4000                             | Anywh... <a href="#">Q</a>            | 0.0.0.0/0 <a href="#">X</a>                  | <a href="#">Löschen</a> |
| -                               | Benutzerdefinier... <a href="#">Q</a> | TCP                            | 5000                             | Anywh... <a href="#">Q</a>            | 0.0.0.0/0 <a href="#">X</a>                  | <a href="#">Löschen</a> |

[Regel hinzufügen](#)

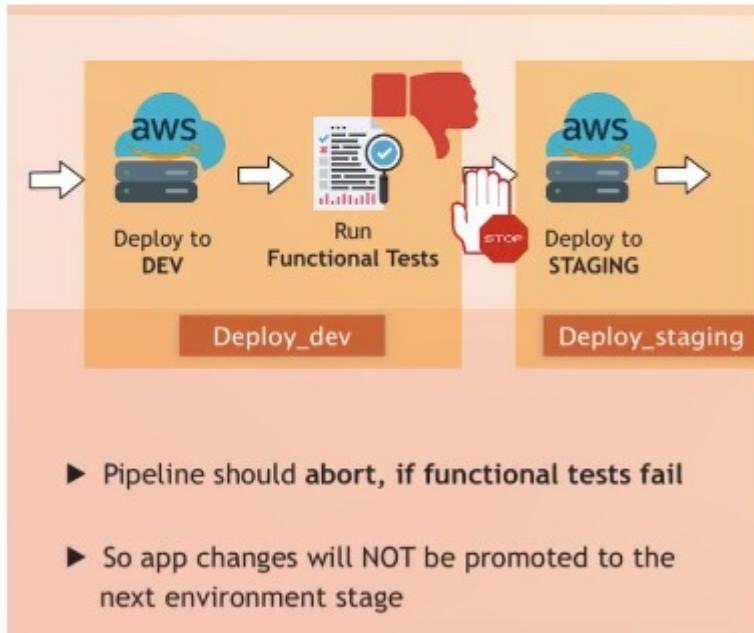
[Abbrechen](#)

[Änderungen in der Vorschau anzeigen](#)

[Regeln speichern](#)

need to set port security-> security group-> inbound rules

Fail functional tests to skip promoting to staging

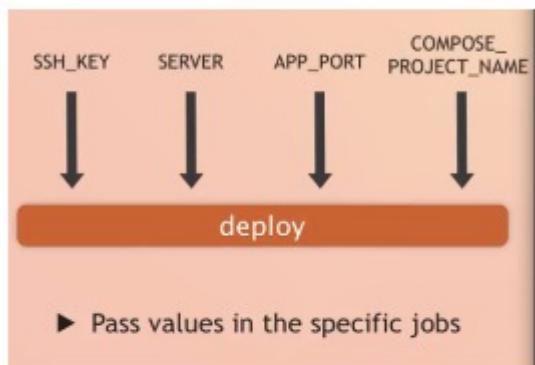


deploy to production:

- deploy\_dev and deploy\_staging have similar structure
- avoid code duplication. reuse configuration sections
- ? how to reuse code in gitlab cicd.
- concept: extends

### "extends"

- ▶ Is used to reuse configuration sections
- ▶ Job inherits configuration from the other job

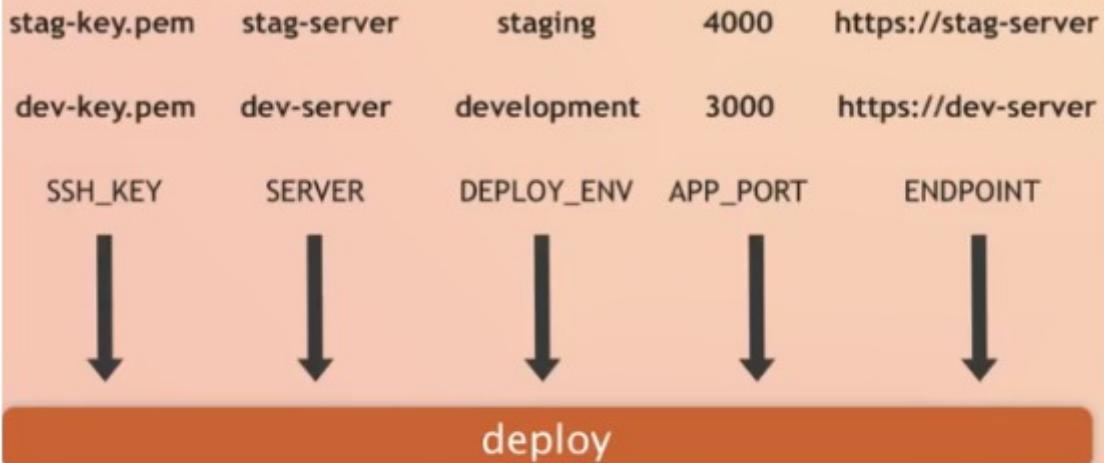


Use base code, using keyword extends in the specific jobs

```
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-con...  
    - ssh -o StrictHostKeyChecking=no -i $SSH_PRIVATE_KEY ubuntu@$SERV...  
    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
```



Extracted duplicate logic into an generic job



Implement the custom job behavior by setting the variables

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

Hide jobs

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

- ▶ If you start the job with a dot it's not processed by GitLab CI/CD

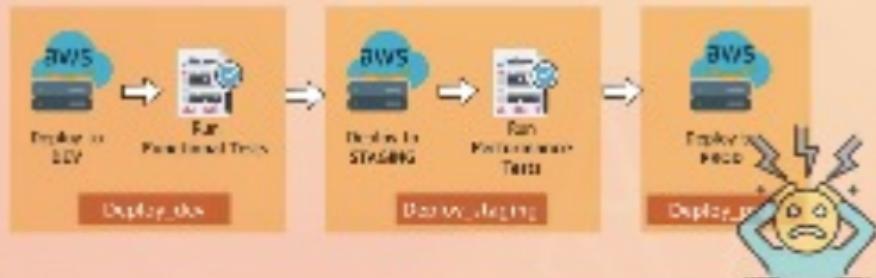
- ▶ You can also comment out jobs with "#" to hide a job



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!

## Manual Approval



- In real life, many teams don't feel comfortable deploying to production automatically
- Without any manual human review



Pipeline deploys to development and staging environment automatically



Manually trigger the deployment to production environment

## 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

```
variables:
  - env: "DEV"
  - host_ip: "1.1.1.1"
  - dev_server_host: "54.195.229.194"
  - dev_endpoint: "http://ec2-54-195-229-194.us-east-1.compute.amazonaws.com:3000"
  - stage_host: "54.195.229.194"
  - stage_endpoint: "http://ec2-54-195-229-194.us-east-1.compute.amazonaws.com:4000"
  - prod_server_host: "54.195.229.194"
  - prod_endpoint: "http://ec2-54-195-229-194.us-east-1.compute.amazonaws.com:5000"
```

```
deploy:
  hosts:
    - ec2
    - docker
    - remote
  before_script:
    - echo $SSH_PRIVATEKEY | sed -e 's/-----BEGIN RSA PRIVATE KEY-----\n-----END RSA PRIVATE KEY-----/-----END RSA PRIVATE KEY-----/g' > ./display-key.pem
    - chmod 400 display-key.pem
    - curl -s https://$CONTAINER_IP:8080/_version/file.txt
  variables:
    SSO_KEY: ""
    SERVER_HOST: ""
    DEPLOY_ENV: ""
    APP_PORT: 5000
    DEPLOYPOINT: ""
  stages:
    - dev:
        - ssh -o StrictHostKeyChecking=no -i display-key.pem ./docker-compose.yml down
        - ssh -o StrictHostKeyChecking=no -i display-key.pem docker login $CONTAINER_HOST
        - docker login -u $CONTAINER_REGISTRY_USER -p $CONTAINER_REGISTRY_PASSWORD $CONTAINER_REGISTRY
        - export DEPLOY_ENV=$APP_ENV-$DEPLOYPOINT-$ENV
        - export DC_IMAGE_NAME=$IMAGE_NAME-$ENV
        - export DC_IMAGE_TAG=$VERSION-$ENV
        - export DC_APP_PORT=$APP_PORT-$ENV
        - docker-compose down $ENV
        - docker-compose up -$ENV
    environment:
      name: DEPLOY_ENV
      url: DEPLOYPOINT
```

```
deploy_to_staging:
  extends: .deploy
  stage: deploy_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: deploy_staging
  needs:
    - deploy_to_staging
  script:
    - echo "running performance test"

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

## Communication between microservices



Communication via API Calls

- ▶ Each service has its own API
- ▶ By calling the respective API endpoint, they can talk to each other

```

image: gcr.io/google-samples/react-native-microservice:latest
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



payment

shopping-cart



- ▶ 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



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



**Configure the communication between services**



**More difficult to monitor with multiple instances of each service distributed across servers**

CICD pipeline for microservices

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 management 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
    - additional logic necessary 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:



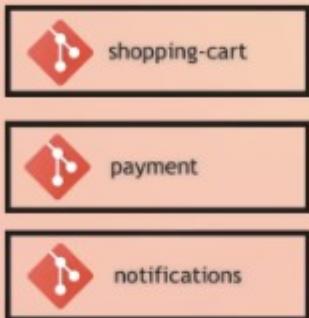
### Downsides of Polyrepo

- ✖ 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.



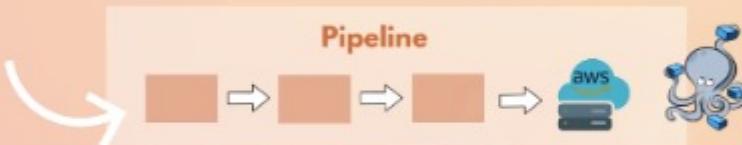
For smaller microservice applications



## Microservices - Demo Overview



MONOREPO



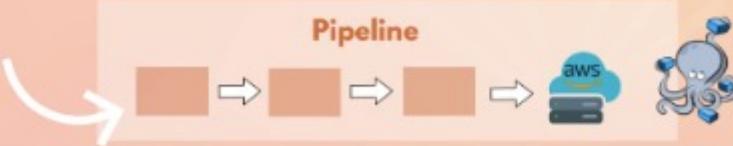
See comparison!



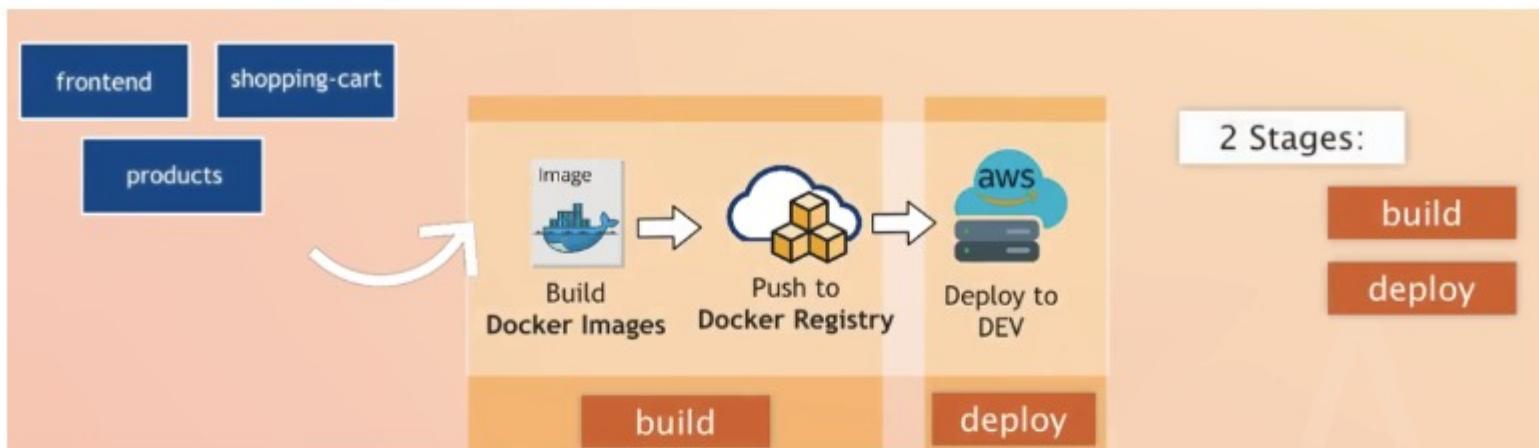
Learn new Gitlab CI/CD concepts, like  
reusing Pipeline Configurations



POLYREPO



1. use runner from other project
2. prepare deployment environment deployment 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

```
before_script:
  - cd frontend
    Run build_frontend job only when code
    changed in /frontend folder
script:
  - docker login -u $CI_REGISTRY_USER -p
  - docker build -t $IMAGE_NAME:$IMAGE_TAG
  - docker push $IMAGE_NAME:$IMAGE_TAG
only:
  changes:
    - "frontend/**/*"
```

```
.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/**/*"
```

frontend

YAML

docker-compose.yml

shopping-cart

YAML

docker-compose.yml

products

YAML

docker-compose.yml

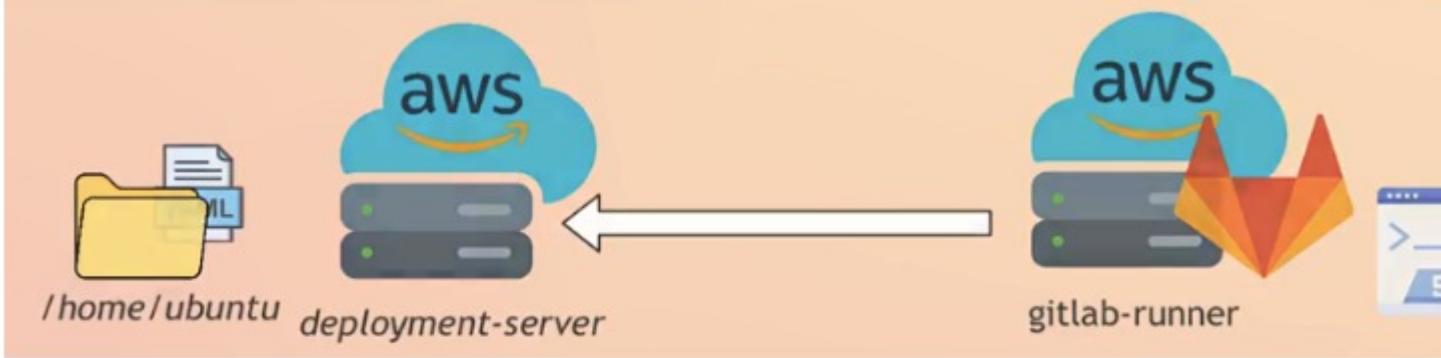
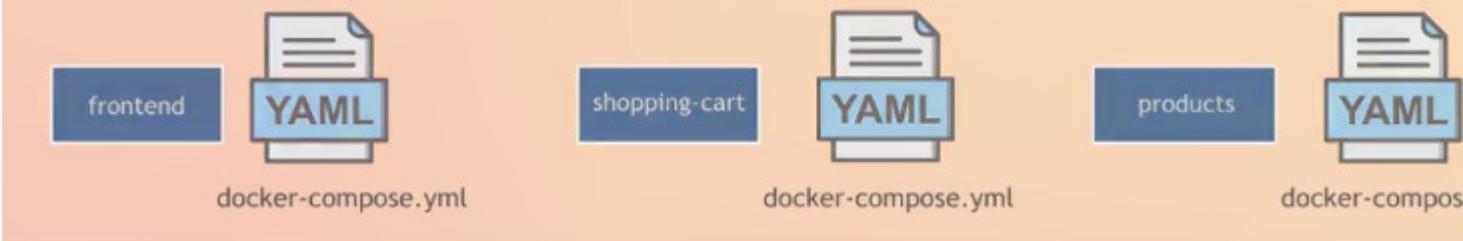
/home/ubuntu

deployment-server

gitlab-runner

```
.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/**/*"
```



```
.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_SERVER_HOST
    - 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-compose file

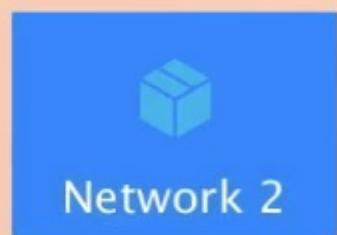
## Networking in Docker Compose

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



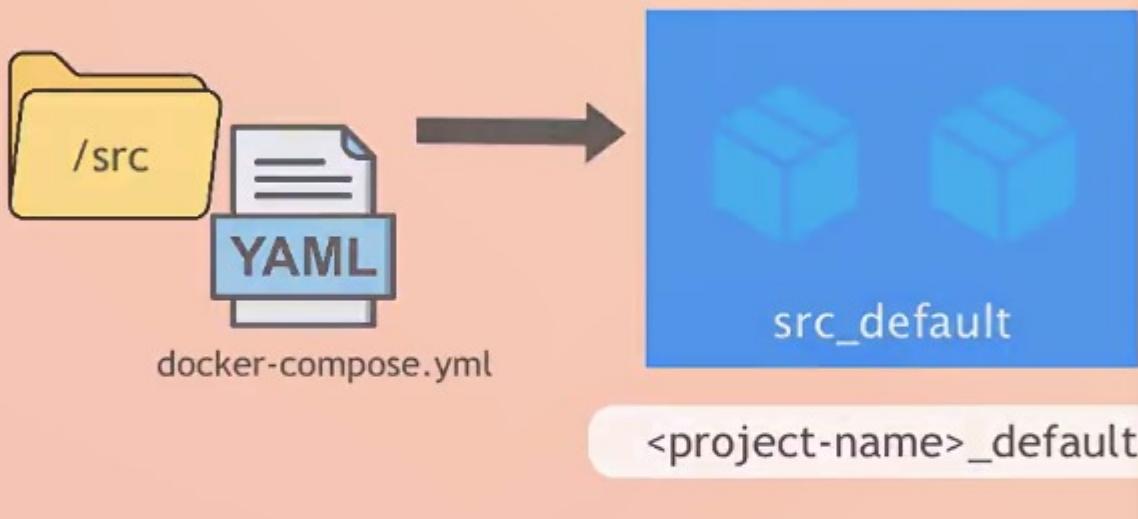
docker network is

Each network is like an isolated virtual network



## Networking in Docker Compose

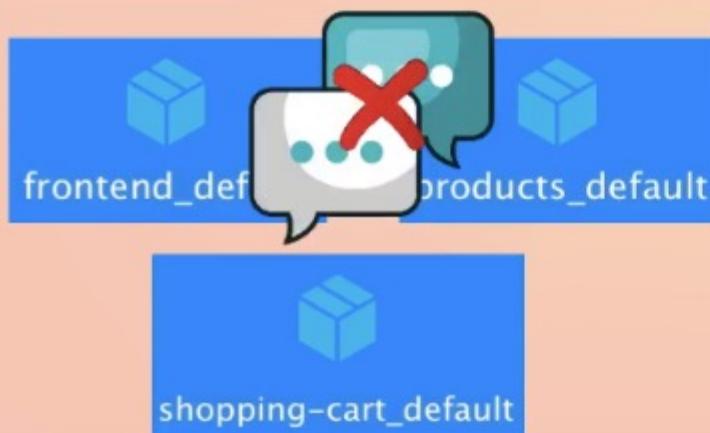
- ▶ Compose creates a new Docker network on every docker-compose up
- ▶ **By default:** the network name is based on the name of the directory the compose file resides



## Networking in Docker Compose

### Problem:

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



create one network all microservices inside

```
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"]
```

```
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
```

### 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

```
- export IMAGE_NAME=$SERVICE_VERSION
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

```
1 FROM node:17-alpine
2
3 WORKDIR /usr/src/app
4
5 COPY package*.json .
6 RUN npm install
7 COPY index.html .
8 COPY server.js .
9
10 ENV PRODUCTS_SERVICE="products_app_1"
11 ENV SHOPPING_CART_SERVICE="shopping-cart_app_1"
12
13 EXPOSE 3000
14 CMD ["npm", "start"]
15
```

```
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/**/*"
```

```
.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/**/*"
```

```
1 version: "3.3"
2 services:
3   app:
4     image: ${DC_IMAGE_NAME}:${DC_IMAGE_TAG}
5     ports:
6       - ${DC_APP_PORT}:${DC_APP_PORT}
7     networks:
8       - micro_service
9
10 networks:
11   micro_service:
12     external:
13       name: micro_service
14
```

## See the CI/CD pipeline with a Polyrepo



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

A screenshot of the GitLab web interface. The sidebar shows navigation links: Group information, Issues, Merge requests, Security, CI/CD, Runners (selected), Packages & Registries, and Settings. The main area shows a group named 'mymicroservice-cld'. A banner at the top says 'Runners' and 'Configure Runner for whole group, instead of each project'. Below this, there's a table with columns: Status, Runner, Version, Jobs, Tags, and Last contact. One runner is listed: 'active' (runner ID 81638759, IP address 98.197.15.194, version 15.2.3, 0 jobs, tags 'gitlab-runner', last contact just now).

## 2 Prepare deployment environment

We will reuse the deployment server as well

Stop all Docker containers and Images

delete all images, docker and network

create SSH\_PRIVATE\_KEY

3 first create docker- compose file

### Why Job Templates?



Each team needs to write and maintain own file

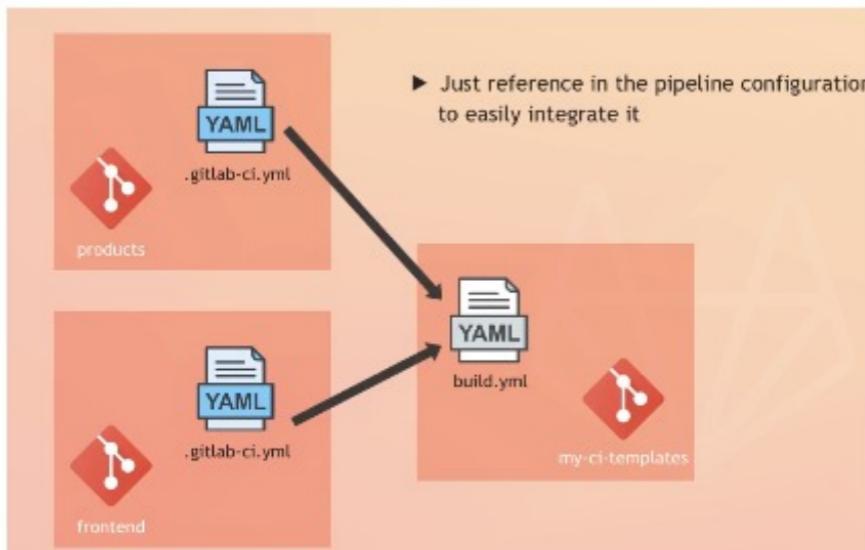
- ▶ Pipeline configurations may grow apart
- ▶ Instead of having a common configuration that all microservices can use



## Job Templating

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

Another project can **include** it in its own CI/CD configuration



template: own or other teams  
extract configuration into 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 paste build job there

```
b-ci.yml ① .build-template.y... ②
{
  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
}
```



Configuration is already generic & customizable, no hardcoded values

```

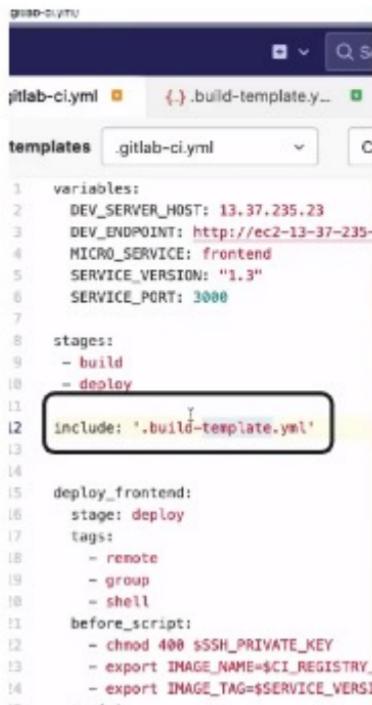
variables:
  MICRO_SERVICE: ""
  SERVICE_VERSION: ""

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-template in the pipeline -> include



```

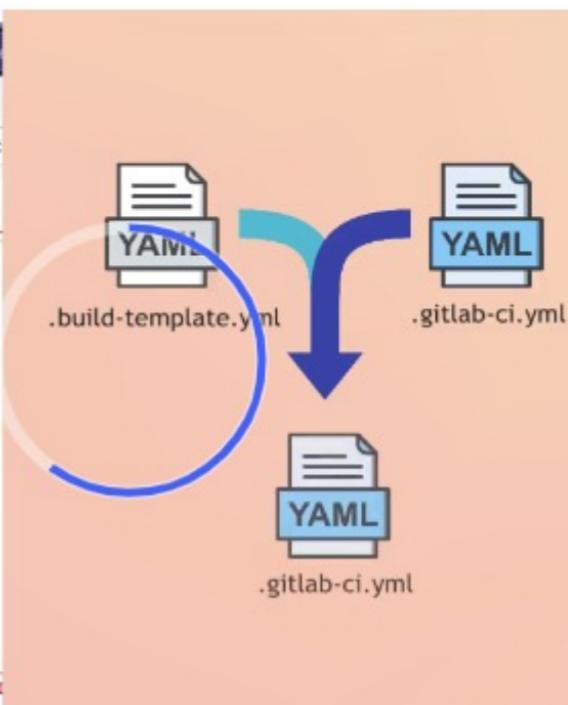
variables:
  DEV_SERVER_HOST: 13.37.235.23
  DEV_ENDPOINT: http://ec2-13-37-235-
  MICRO_SERVICE: frontend
  SERVICE_VERSION: "1.3"
  SERVICE_PORT: 3000

stages:
- build
- deploy

include:
  - '.build-template.yml'

deploy_frontend:
  stage: deploy
  tags:
    - remote
    - group
    - shell
  before_script:
    - chmod 400 $SSH_PRIVATE_KEY
    - export IMAGE_NAME=$CI_REGISTRY_
    - export IMAGE_TAG=$SERVICE_VERSI
  script:

```



Then in the pipeline is very easy:

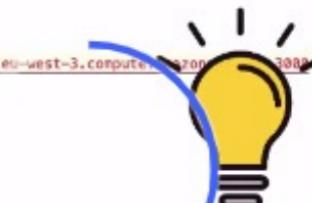
```
when: always
- run_performance_tests
variables:
  SSH_KEY: $SSH_PRIVATE_KEY
  SERVER_HOST: $PROD_SERVER_HOST
  DEPLOY_ENV: production
  APP_PORT: 5000
  ENDPOINT: $PROD_ENDPOINT
when: manual
```

## "include"

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

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

```
1 include:
2   local: ".build-template.yml"
3
4 variables:
5   DEV_SERVER_HOST: 13.37.235.23
6   DEV_ENDPOINT: http://ec2-13-37-235-23.eu-west-3.compute.amazonaws.com:3002
7   MICRO_SERVICE: frontend
8   SERVICE_VERSION: "1.3"
9   SERVICE_PORT: 3008
10
11 stages:
12   - build
13   - deploy
14
15 build:
16   tags:
17     - remote
18     - group
19     - shell
```



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

如果template里面也有before\_script，将会被build pipeline里面的before\_script移除

netx create deploy template

```
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.yml and deploy.yaml

```
File templates | gitlab-ci.yml | Choose a template... ▾  
1 include:  
2   - projects: ''  
3  
4  
5 variables:  
6   DEV_SERVER_HOST: 13.37.235.23  
7   DEV_ENDPOINT: http://ec2-13-37-235-23.eu-west-3.compute.amazonaws.com:3888  
8   MICRO_SERVICE: frontend  
9   SERVICE_VERSION: "1.3"  
10  SERVICE_PORT: 3008  
11  
12 stages:  
13   - build  
14   - deploy  
15  
16 build:  
17   tags:  
18     - remote  
19     - group  
20     - shell  
21  
22 deploy:
```

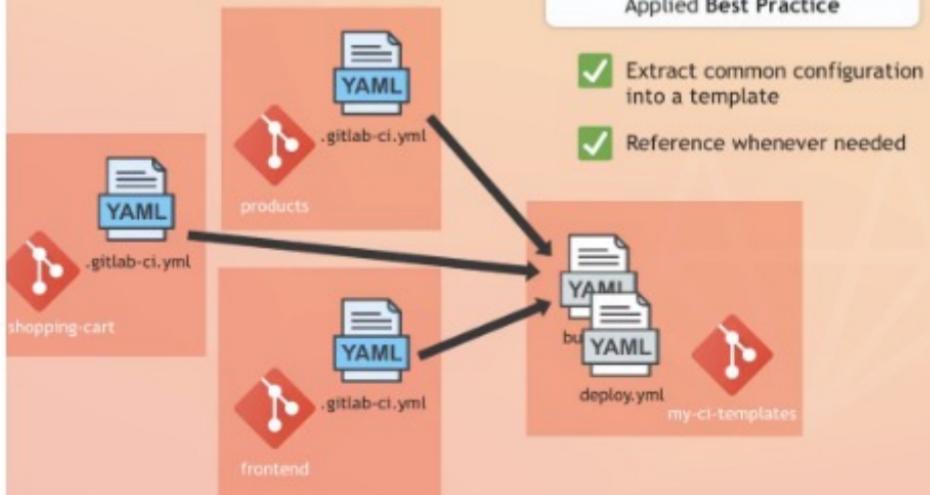
### "include:file" & "include:project"

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

Use another keyword projects

```
1 include:  
2   - project: mymicroservice-cicd/ci-templates  
3     ref: main  
4     file:  
5       - build.yml  
6       - deploy.yml  
7
```

## Applied Best Practice



✓ Extract common configuration into a template

✓ Reference whenever needed

Pipeline #527254787 passed for 054f340a: Update .gitlab-ci.yml

⚠ This GitLab CI configuration is invalid: Remote file 'https://gitlab.com/awto-devops.gitlab-ci.yml' does not exist.

Edit Visualize List View merged YAML

Browse templates Help

```
1 include:
2   - project: mymicroservice-ci/ci-templates
3     ref: main
4   file:
5     - build.yml
6     - deploy.yml
7   remote: "https://gitlab.com/awto-devops.gitlab-ci.yml"
8   template: Auto-DevOps.gitlab-ci.yml
9
10 variables:
11   MICRO_SERVICE: shopping-cart
12   SERVICE_VERSION: "2.1"
13   SERVICE_PORT: 3002
14
15 stages:
16   - build
17   - deploy
```

## include

### To include external YAML files

#### - local

► Reference from same repository

#### - file

► Reference from another private project (same GitLab Instance)

#### - remote

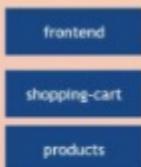
► Include from a different location (full URL necessary)

#### - template

► Include GitLab's templates

deploy from gitlab ci/cd pipeline to a k8s cluster

### Demo Overview



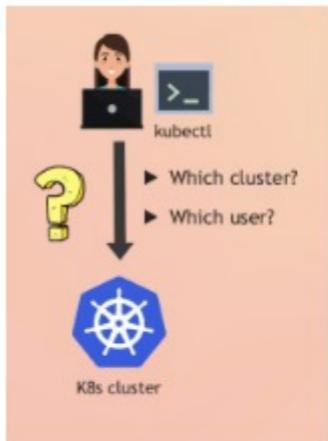
### CI/CD Pipeline



- 1) Create a Kubernetes cluster
- 2) Create a dedicated gitlab user (security best practice)
- 3) Create K8s manifests (Deployment and Service) for microservices
- 4) Adjust GitLab pipeline configurations

create a managed k8s platform

tell kubectl



**kubeconfig =**  
File used to configure access to K8s

- Contains all the information needed to connect to the cluster

K8s API Endpoint

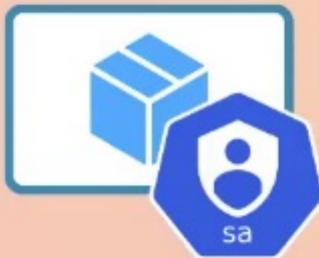
Username/Password

Or secure token

Kubeconfig:

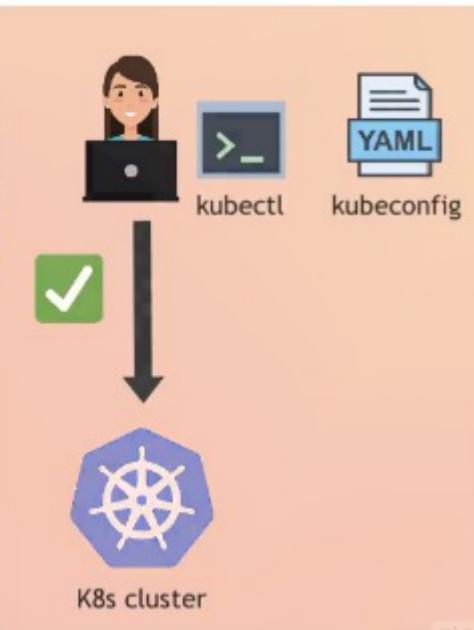
↓ my-micro-service-kubeconfig.yaml |

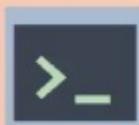
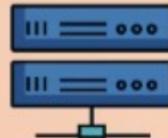
# ServiceAccount = Represent non-human user



chmod 400 to configuration 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
```



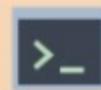


kubectl

- ▶ Kubernetes CLI that you can install on every machine



admin-kubeconfig



kubectl



Should GitLab have admin permissions?

Create Deployment



Create Service

in a specific namespace

Create ConfigMap

Create Secret



K8s cluster



Doesn't need cluster wide permissions



## Security Best Practice

1) Create a dedicated User (ServiceAccount) for GitLab



2) Create restricted permissions (Role)

3) Generate kubeconfig file for ServiceAccount



kubectl



gitlab-  
kubeconfig



K8s cluster



## Security Best Practice



K8s administrator



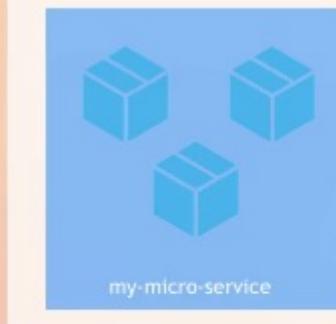
Gitlab

Admin User

GitLab User

- ▶ Deploy our services in a dedicated namespace

- Restrict GitLab access to that namespace



Namespace = Isolated space for your K8s resources within your cluster

```
kube-system          Active   tom
[\W]$ kubectl create namespace my-micro-service
```

```
[\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]$
```

```
serviceaccount/cicd-sa created
[\W]$ vim cicd-role.yaml
permissions for role
```

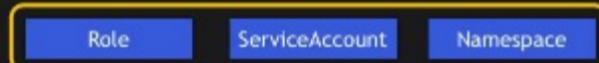
```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: my-micro-service
  name: cicd
rules:           add secret after "services"
- apiGroups: [""]
  resources: ["pods", "services"]
  verbs: ["get", "list", "watch", "create", "update", "patch", "delete"]
- apiGroups: ["extensions",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

because you will create secret  
for container docker

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



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



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



## Create kube config file



ServiceAccount created



gitlab-kubeconfig

► Create kubeconfig access file

► Copy admin kubeconfig file and replace values

```
[\\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]$ 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: eyJhbGciOiJSUzIiNiIsImtpZCI6IlYMDZrdkIBWF1jY1lGbDYzU2h2ZDVvMURCDElXY1RTZFpLV31
CNU2iV1kfQ.eyJpc3MiOiJrdW1lc3NlcnZpY2VhY2NvdW50Iiwia3ViZXJuZXRLcy5pbby9zZXJ2aWNlYnjb3Vudc9z
Njb3VudC9uYW1lc3BhY2UiOjteSitaWNyby1zzXJ2aWNlIiwiia3ViZXJuZXRLcy5pbby9zZXJ2aWNlYnjb3Vudc9z
ZwNyZXQubmFtZSI6ImPy2Qtc2EtdG9rZW4tNGZrZ24iLCJrdW1lc3NlcnZpY2VhY2NvdW50L3Nlcn
ZpY2UtYWNjb3VudC5uYW1ljoiy2ljZC1zYSIsImti1YmVybmV0ZXMuaw8vc2VydmljZWfjY29ibonQvc2VydmljZS1h
Y2NvdW50LnVoZCI6IjFiMzE2YzQ1LTEwZDQtNDQxOSihYTk3LWVkoGNkNNUUyZjg4YSIsInN1YiI6InN5c3R1bTpZx
J2aWNlYnjb3VudDptesitaWNyby1zZXJ2aWh10mNpY2Qtc2Eif0.DylKvx_UDQonijPw413f1HJDYNCr1xvnjzy3
Pnfs5okEZ3oHfcyyvSuWc27mtqTD91yQA1CtFktRy65LJX-jIglhsFPdApwb8HsaeJkrJ94_VM-iCc2uRRvSo9LHT
4RFxMSxueyejn3fmr5Fio3ots9_KYWLMMH1StaR93ZBJ_yFH91U4E0ZLeNf5uhp5gleJAKS7-40sAXIwbfK8xCnp
vG-7FPNkZKkJE-nr1KV4zq-gL8VGKqW_vMwI8uQrlCT9ZC5sS45g7PBZXVUHEZwJJneCPAJ9UQJriPnVUHzNb1NXid
2nVuTFJeR35Q81_31ONS7jYB6Toaa22whsYg

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

Context = Used to group access parameters under a name

Each context has 3 parameters: cluster, namespace, user

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

test:

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

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

certificate-authority-data:  
LS0tLjCrUdJ1bDRVJlJSUzJ3J0BFUR50tL50tCk115UlvakN0D0WvhZ0F35UJBz01CQURBTkjna3fpa21H0Xc  
wQkFRc0ZBREFwTVJNd0VRwJRMUUVFERXdwcrRXSmwY201bGRHVnpnQjRYRFRJe01EVXdNVEE1TkRneE1sb1  
hHEVE1STURre09e0TV0R6d4Twxvd8ZURVRNQikVHQTFVR0p0eE1LYTMNaVpYSnVwFjsY3pdQBFTSxdEUw1K5  
29a5Nh29b5BUUVCQ1FBRGdnRVBBREND0VfQ2dnRUJBTUZYCcFXU024ySHJ1NFVV0nLZ0XvPeKZuVdAvnBE  
KBZJekd5DXYrwDY2SzFd4gW5YnAxZHE3YjF0VHRUicWovMEQzMNgkCS3dYY3VpTVJvhzdDUzR5b1A0dGRJYmN  
pwInhdVFKaWJUM1NLefJ33MTLQb3A0YWHM2VINjNMU3dPcUo r0GdPYgoYMriwvcjZEVGhxNnZTVWkzlwPb0

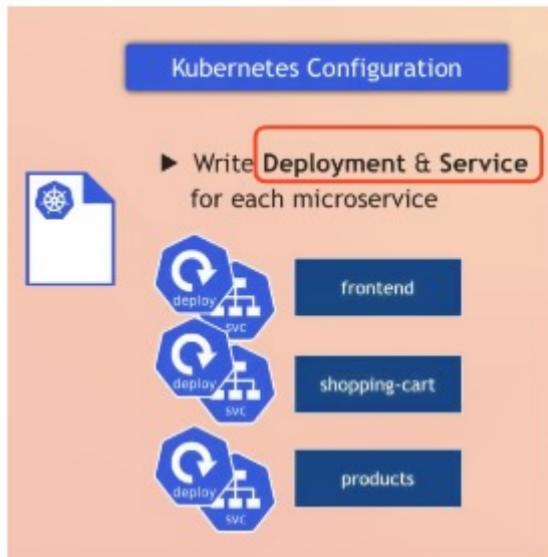
Type: File      Environment scope: All (default)

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](#)

configurations we need to write to deploy our service into the k8s cluster



- 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 deployment and service.
- for the docker inside the pod you need docker registry secret
- in the gitlab runner you need to execute the k8s so you need to install kubectl
- we have k8s so we change docker-impostor with k8s

## New file

```
' main / deployment.yaml Select a template type
```

```
1 apiVersion: apps/v1
2 kind: Deployment
3 metadata:
4   name: frontend
5   namespace: my-micro-service
6 spec:
7   selector:
8     matchLabels:
9       app: frontend
10  replicas: 1
11  template:
12    metadata:
13      labels:
14        app: frontend
15    spec:
16      containers:
17        - name: frontend
18          image: ...
19          ports:
20            - containerPort: 30808
```

## 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
```

```
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      containers:
17        - name: $MICRO_SERVICE
18          image: $IMAGE_NAME:$IMAGE_TAG
19          ports:
20            - containerPort: $SERVICE_PORT
```

## deployment.yaml service.yaml

```
apiVersion: v1
kind: Service
metadata:
  name: frontend
  namespace: my-micro-service
spec:
  selector:
    app: frontend
  ports:
    - protocol: TCP
      port: 3808
      targetPort: 3000
```

### Service Configuration

**name**

► Can be any name

**selector**

► Defines which Pods  
(created by the Deployment) are  
connected to the Service

```
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**

3000

**Pod**



► Service forwards the request to the  
container's targetPort



Have a separate git repository  
for K8s configurations



frontend



ms-4



ms-7



ms-7



products



ms-5



ms-8



ms-8



ms-6

ms-9

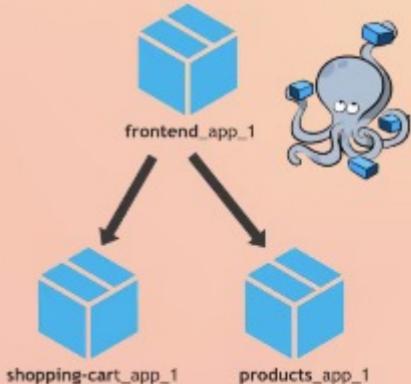


ms-9

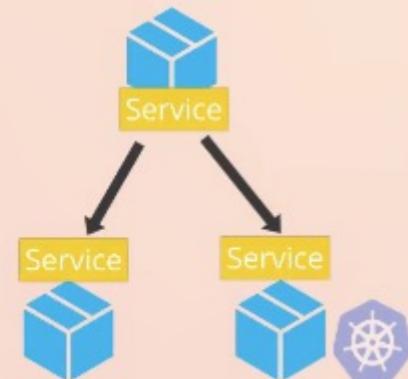
## fox microservice endpoints

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

Before K8s, Docker Compose deployment and communication via Docker Network

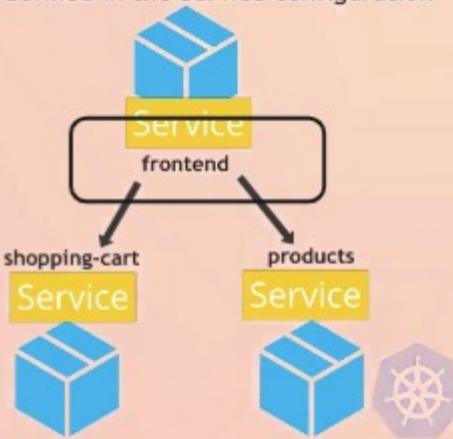


With K8s, we have Service endpoints for communication



With K8s, we have Service endpoints for communication

- ▶ Point to the Service name, defined in the Service configuration



```
1 FROM node:17-alpine
2
3 WORKDIR /usr/src/app
4
5 COPY package*.json ./
6 RUN npm install
7 COPY index.html .
8 COPY server.js .
9
10 ENV PRODUCTS_SERVICE="products"
11 ENV SHOPPING_CART_SERVICE="shopping-cart_app_1"
12
13 EXPOSE 3000
14 CMD ["npm", "start"]
15
```

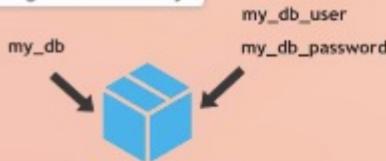
Service = enables communication



Set endpoints and other configuration from outside instead of hardcoding it in Dockerfile

- ▶ Docker Compose
- ▶ ConfigMap (for non-confidential data)  
or Secret (for sensitive data) in K8s

#### Configure externally

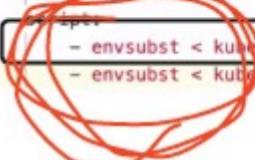


mymicroservice-cicd > ci-templates > Repository

#### New file

这个文件将会被include到每个service里面，

```
# main / deploy-k8s.yml Select a template type ▾  
1 deploy:  
2   stage: deploy  
3   before_script:  
4     - export IMAGE_NAME=$CI_REGISTRY_IMAGE/microservice/$MICRO_SERVICE  
5     - export IMAGE_TAG=$SERVICE_VERSION  
6     - export MICRO_SERVICE=$MICRO_SERVICE  
7     - export SERVICE_PORT=$SERVICE_PORT  
8     - export REPLICAS=$REPLICAS  
9     - export KUBECONFIG=$KUBE_CONFIG  
10    after_script:  
11      - envsubst < kubernetes/deployment.yaml | kubectl apply -f -  
12      - envsubst < kubernetes/service.yaml | kubectl apply -f - I
```



Private GitLab  
container registry



Dedicated Secret type: Docker config Secrets

- ▶ Stores the credentials for accessing a container image registry



container registry



## Variables

### Update variable

X

#### 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

```
--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
  - envsubst < kubernetes/deployment.yaml | kubectl apply -f -
  - envsubst < kubernetes/service.yaml | kubectl apply -f -
```

kubectl commands are imperative

Imperative = Tell what to do

Apply YAML files, which are declarative

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 kubectl

in the gitlab-runner check also envsubst command. It is a linux command.

summary 任务1 deployment and service for each service. Create it in the kubernetes file and 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 kubectl create docekr registry secret important, which used in the deployment.yaml image

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

How to change the pipeline code in the service

## Pre-defined GitLab variables

The file has been successfully created.

Add new file  
None Janeisha authored just now

deploy-k8s.yaml 488 bytes

```
1 deploy:
2   stage: deploy
3   before_scripts:
4     - export IMAGE_NAME=$CI_REGISTRY_IMAGE/microservice/$MICRO_SERVICE
5     - export IMAGE_TAG=$SERVICE_VERSION
6     - export MICRO_SERVICE=$MICRO_SERVICE
7     - export SERVICE_PORT=$SERVICE_PORT
8     - export REPLICAS=$REPLICAS
9     - export KUBECONFIG=$KUBE_CONFIG
10  script:
11    - kubectl create secret docker-registry my-registry-key --dry-run=client --docker-username=$GITLAB_USER --docker-password=$GITLAB_PASSWORD --service --dry-run=client -o yaml | kubectl apply -f -
12    - envsubst < kubernetes/deployment.yaml | kubectl apply -f -
13    - envsubst < kubernetes/service.yaml | kubectl apply -f -
```

\$CI\_REGISTRY\_NAME

Variables, we need to define

\$MICRO\_SERVICE

\$SERVICE\_VERSION

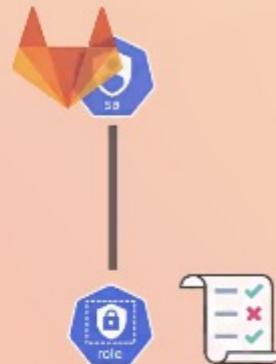
\$SERVICE\_PORT

\$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-runner/microservice-cicd/frontend/.git/  
Checking out d9cd003 as main...  
Skipping Git submodules setup  
Executing "step\_script" stage of the job 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  
\$ kubectl create secret docker-registry my-registry-key --dry-run=client --docker-username=\$GITLAB\_USER --docker-password=\$GITLAB\_PASSWORD --service --dry-run=client -o yaml | kubectl apply -f -  
Error from server (Forbidden): error when retrieving current resource: "v1, Resource: secrets", GroupVersionKind: "v1, Name: "my-registry-key", Namespace: "my-micro-service"  
from server for: "STDIN": secrets "my-registry-key" is forbidden: serviceaccount: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



Role with defined permissions

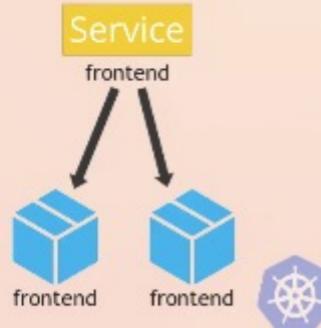


Not allowed to create a Secret

```
[\\W]$ kubectl get service -n my-micro-service
NAME      TYPE      CLUSTER-IP   EXTERNA
frontend  ClusterIP  10.128.160.92 <none>
[\\W]$ kubectl get deployment -n my-micro-service
NAME        READY   UP-TO-DATE   AVAILABLE
frontend   2/2     2           2
[\\W]$ kubectl get pod -n my-micro-service
NAME          READY   STATUS    AGE
frontend-6d5cb78bc5-4sm56  1/1    Running  10m
frontend-6d5cb78bc5-j2zjm  1/1    Running  10m
[\\W]$
```



2 replicas deployed



Access the K8s cluster with localhost  
using kubectl proxy



Service

```
[\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 just build 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.