Technological foundations of software development

Integrate and deploy your software continuously

ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

Objectives of the session

This session is designed to get you familiar with methods and tools for continuous software integration and deployment.
In particular, we will cover Gitlab CI/CD, and Github Actions

Technological foundations of software development

Integrate and deploy your software continuously

Part 1: DevOps

ICM – Computer Science Major – Course unit on Technological foundations of computer scienceopment
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

DevOps

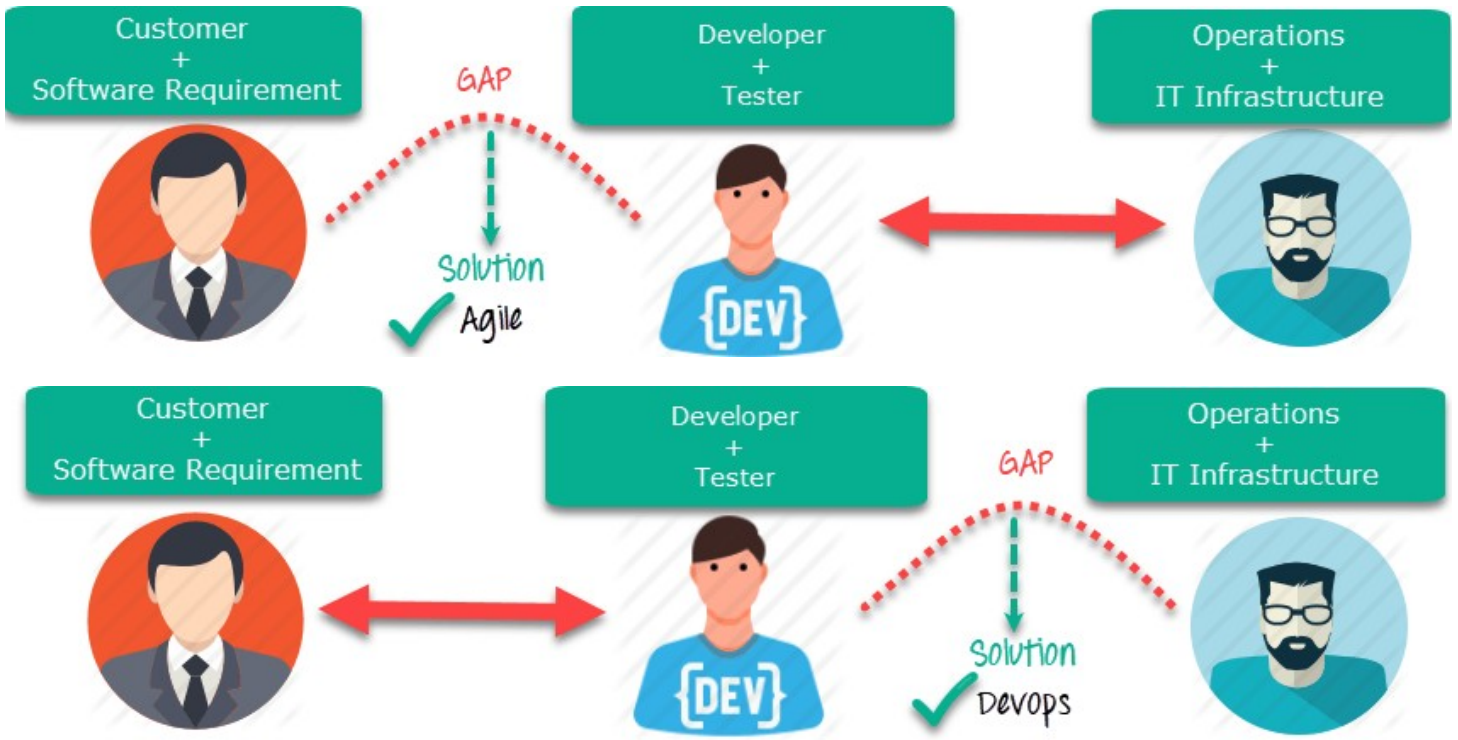
DevOps is a set of practices that combines software development (Dev) and IT operations (Ops). It aims to shorten the systems development life cycle and provide continuous delivery with high software quality. DevOps is complementary with Agile software development; several DevOps aspects came from the Agile methodology.

— Contributors, Wikipedia https://en.wikipedia.org/wiki/DevOps

⚠ There are many variants

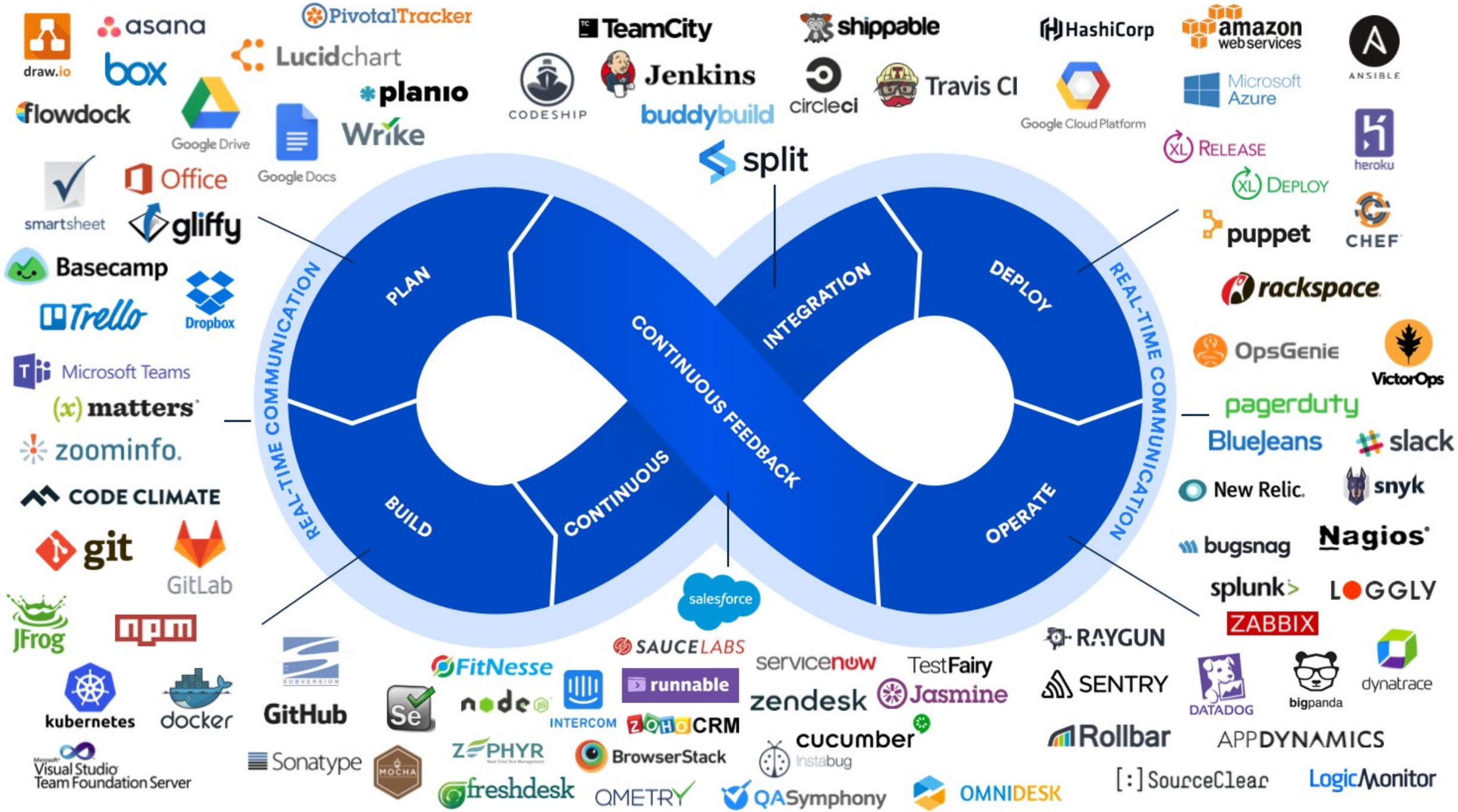
https://theagileadmin.com/what-is-devops/

Agile vs DevOps



- DevOps is a practice of bringing development and operations teams together whereas Agile is an iterative approach that focuses on collaboration, customer feedback and small rapid releases.
- DevOps focuses on constant testing and delivery while the Agile process focuses on constant changes.
- DevOps requires relatively a large team while Agile requires a small team.
- DevOps leverages both shifts left and right principles, on the other hand, Agile leverage shift-left principle.
- The target area of Agile is Software development whereas the Target area of DevOps is to give end-to-end business solutions and fast delivery.
- DevOps focuses more on operational and business readiness whereas Agile focuses on functional and non-function readiness.

Source: https://www.guru99.com/agile-vs-devops.html



https://s32860.pcdn.co/wp-content/uploads/2019/11/devops-hero-1-87966cfbc9c5713ae047551c7b22985c.png

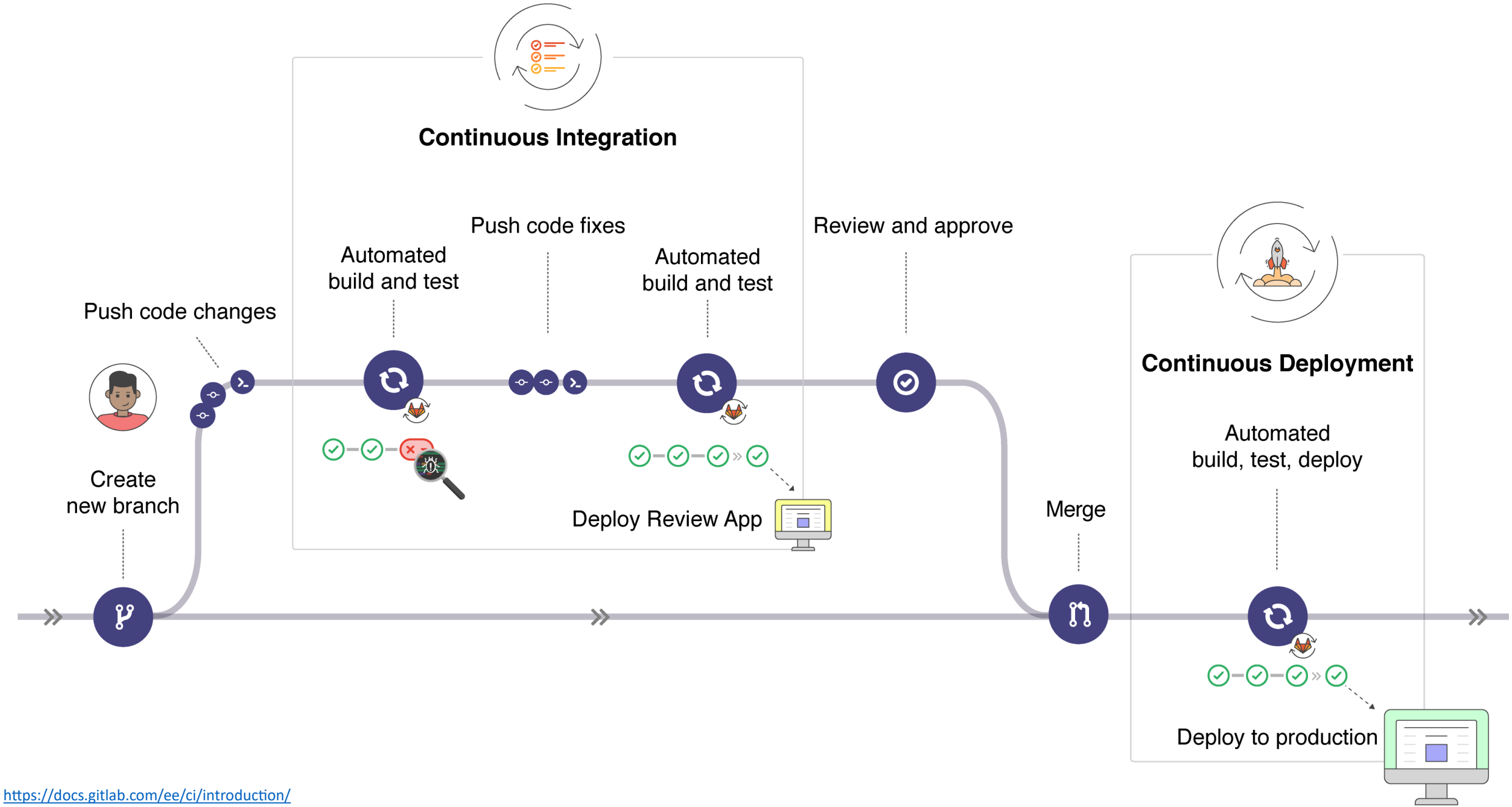
Definitions

Continuous Integration
for every change submitted to an application - even to development branches - it's built and tested automatically and continuously, ensuring the introduced changes pass all tests, guidelines, and code compliance standards you established for your app.

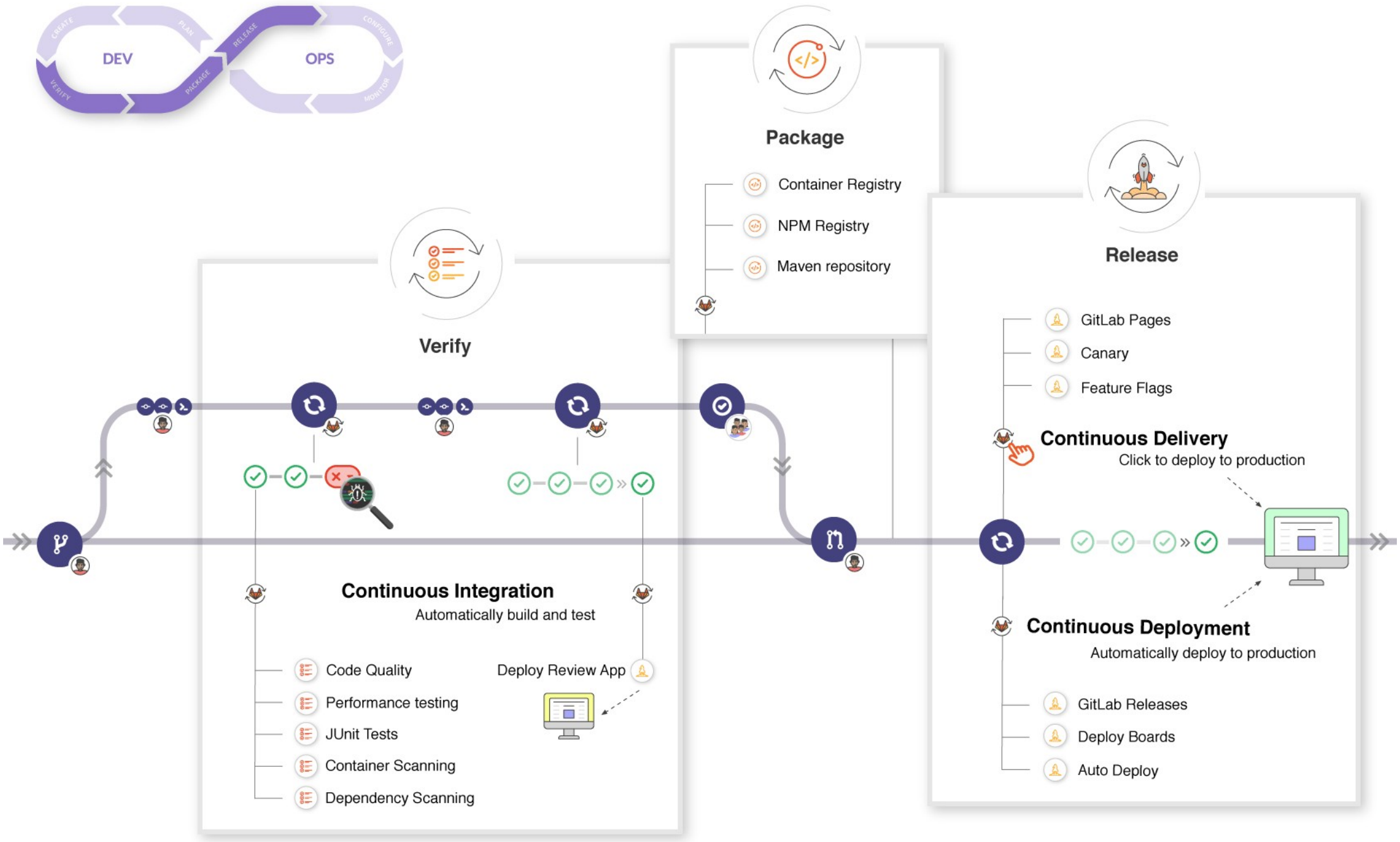
Continuous Delivery
not only built and tested at every code change pushed to the codebase, but, as an additional step, it's also deployed continuously, though the deployments are triggered manually

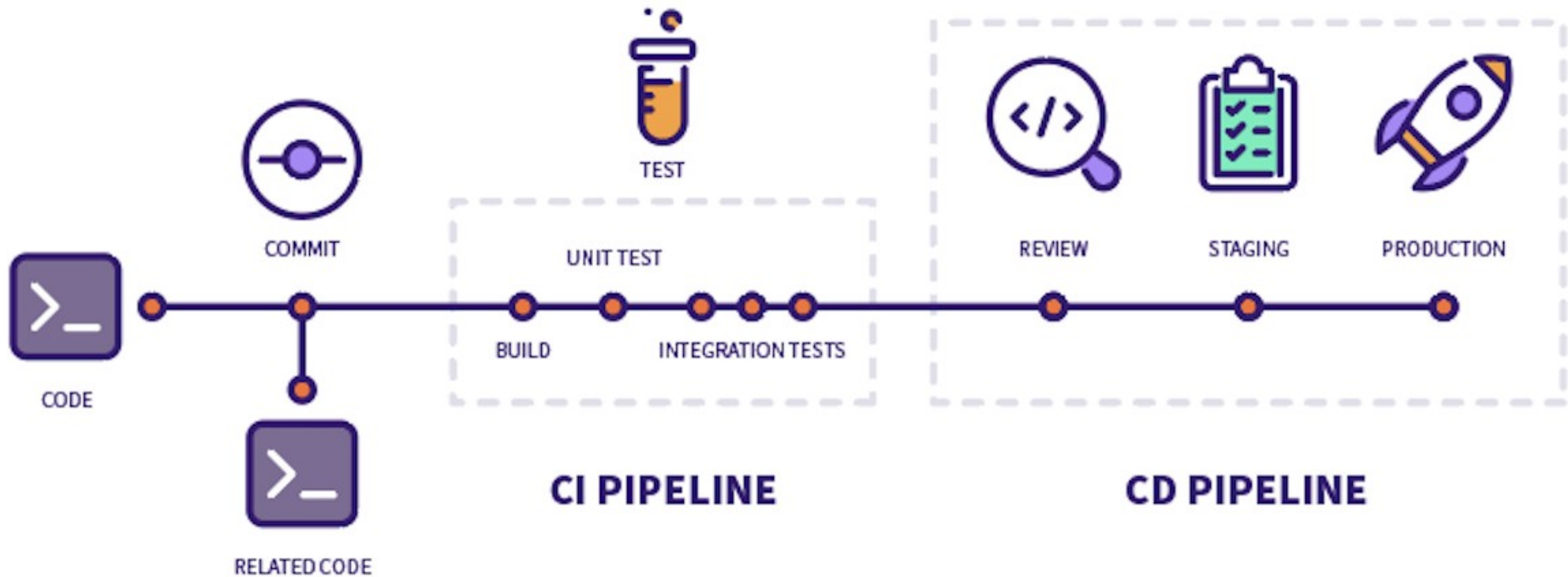
Continuous Deployment
instead of deploying your application manually, you set it to be deployed automatically. It does not require human intervention at all to have your application deployed

https://docs.gitlab.com/ee/ci/introduction/



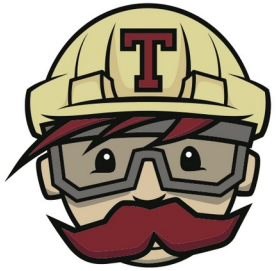
https://docs.gitlab.com/ee/ci/introduction/


DEV
OPS
CREATE
PLAN
RELEASE
CONFIGURE
VERIFY
PACKAGE
MONITOR
Package
Container Registry
NPM Registry
Maven repository
Release
GitLab Pages
Canary
Feature Flags
Continuous Delivery
Click to deploy to production
Verify
Continuous Integration
Automatically build and test
Code Quality
Performance testing
JUnit Tests
Container Scanning
Dependency Scanning
Deploy Review App
Continuous Deployment
Automatically deploy to production
GitLab Releases
Deploy Boards
Auto Deploy


TEST
COMMIT
UNIT TEST
REVIEW
STAGING
PRODUCTION
BUILD
INTEGRATION TESTS
CODE
RELATED CODE
CI PIPELINE
CD PIPELINE

Many tools for CI/CD …

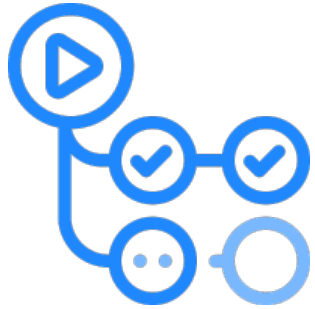
Circle CI
2011
https://circleci.com/
YAML



Travis CI
2011
http://travis-ci.com/
YAML

Jenkins
2011
https://jenkins.io/
Groovy

GitLab CI/CD
2013
https://gitlab.com/
YAML

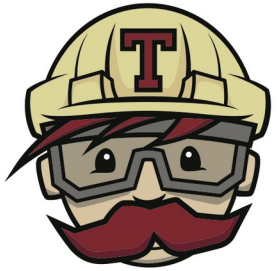


GitHub Actions
2018
https://github.com/features/actions
YAML

… and others

… all supporting Docker (2013)

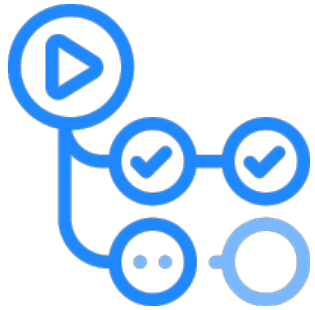
Circle CI
2011
https://circleci.com/
YAML
✅ Supports Docker



Travis CI
2011
http://travis-ci.com/
YAML
✅ Supports Docker

Jenkins
2011
https://jenkins.io/
Groovy
✅ Supports Docker

GitLab CI/CD
2013
https://gitlab.com/
YAML
✅ Supports Docker



GitHub Actions
2018
https://github.com/features/actions
YAML
✅ Supports Docker

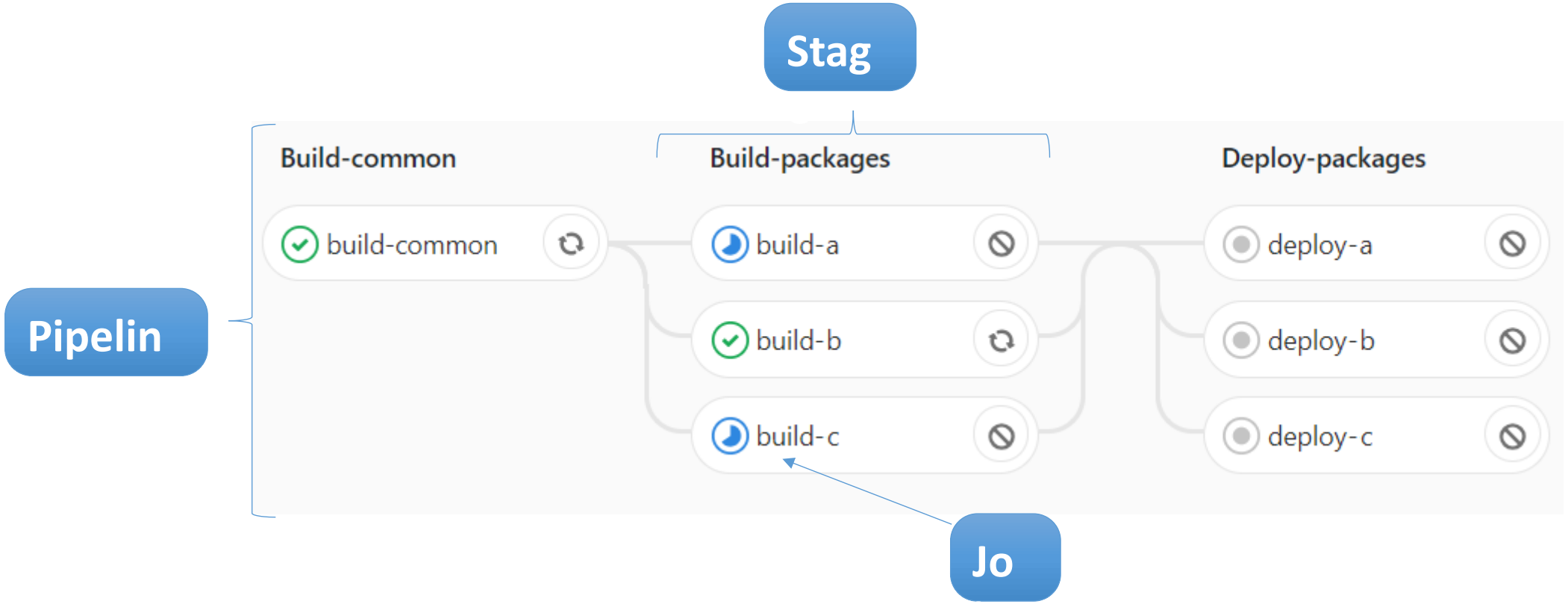
Technological foundations of software development

Intégrer et déployer son logiciel en continue

Part 2: Gitlab CI/CD

ICM – Computer Science Major – Course unit on Technological foundations of computer scienceopment
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

Pipelines



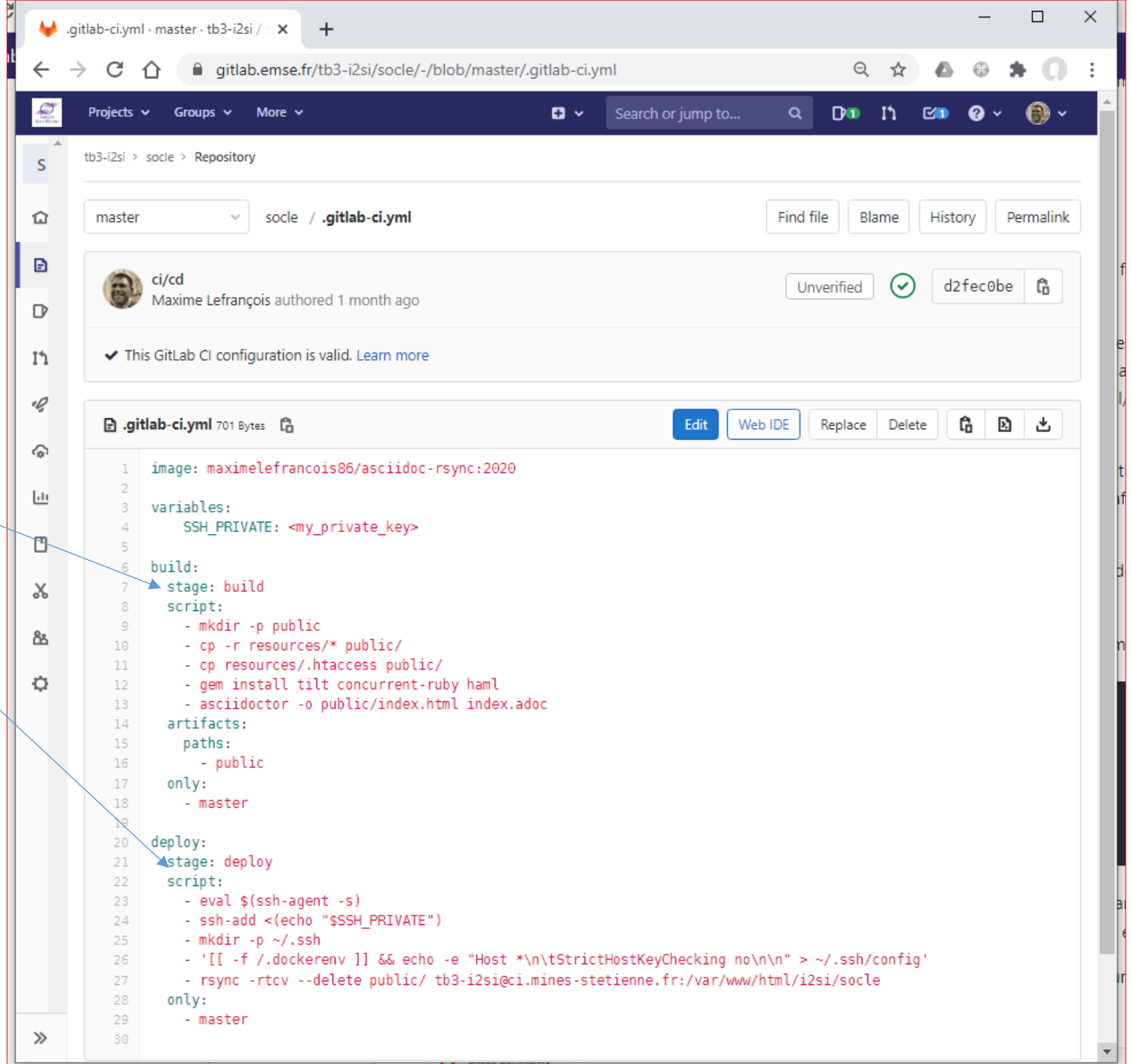
https://docs.gitlab.com/ee/ci/introduction/

Pipelines

Stage

By default:

- Build
- Test
- Deploy



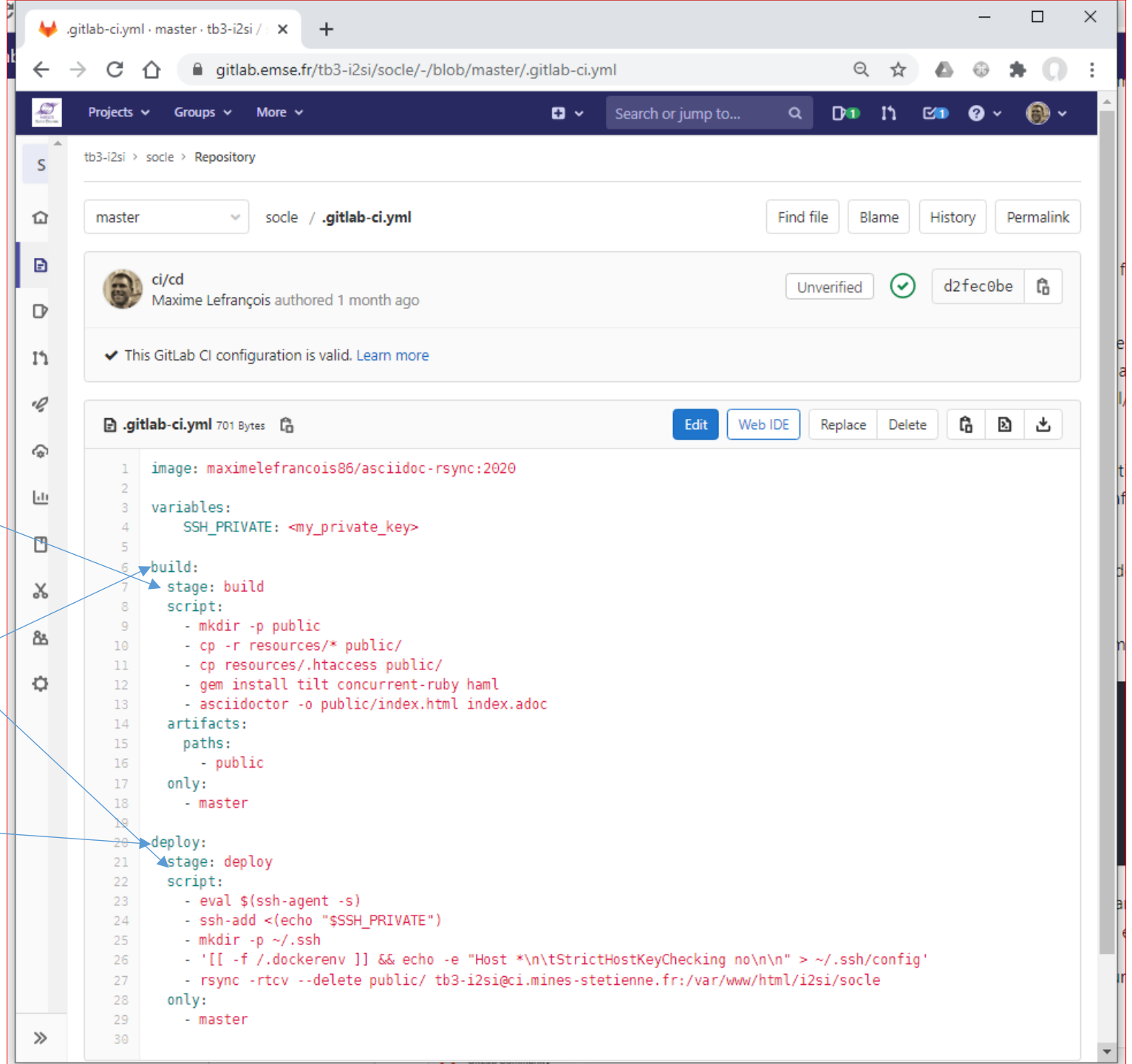
Pipelines

Stage

By default:
- Build
- Test
- Deploy

Job

YAML key names



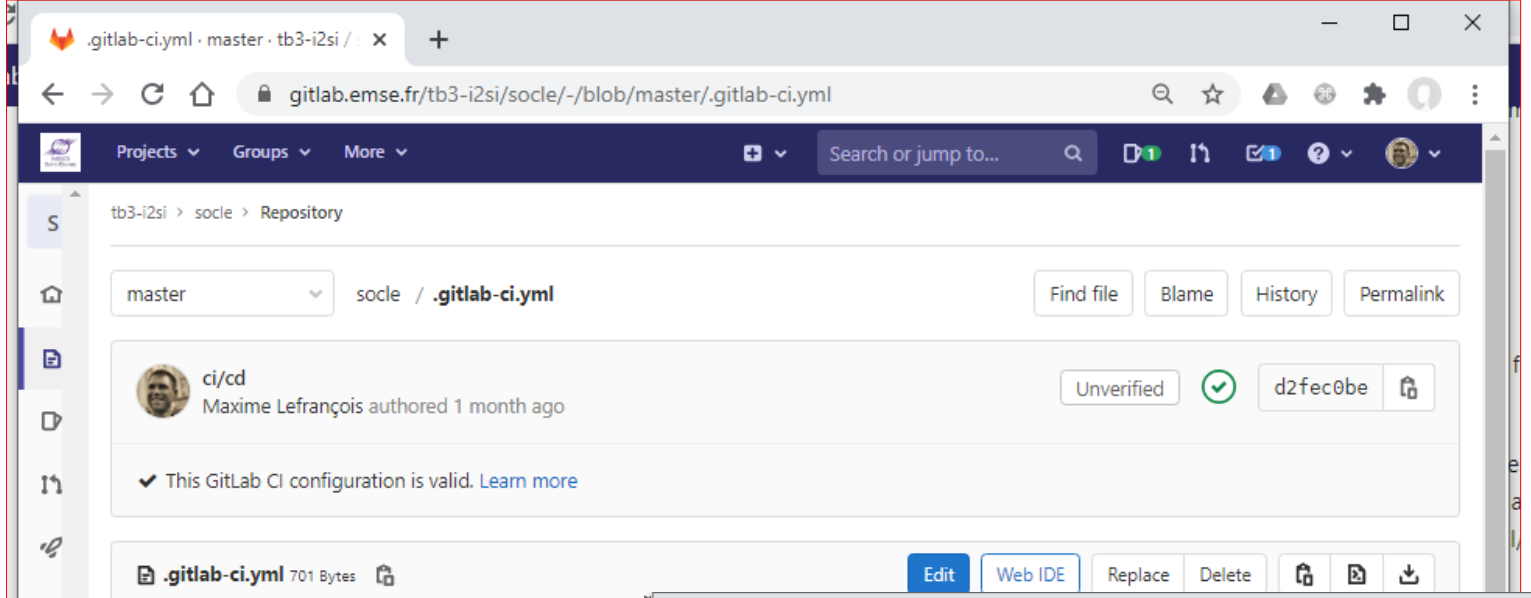
Env variables

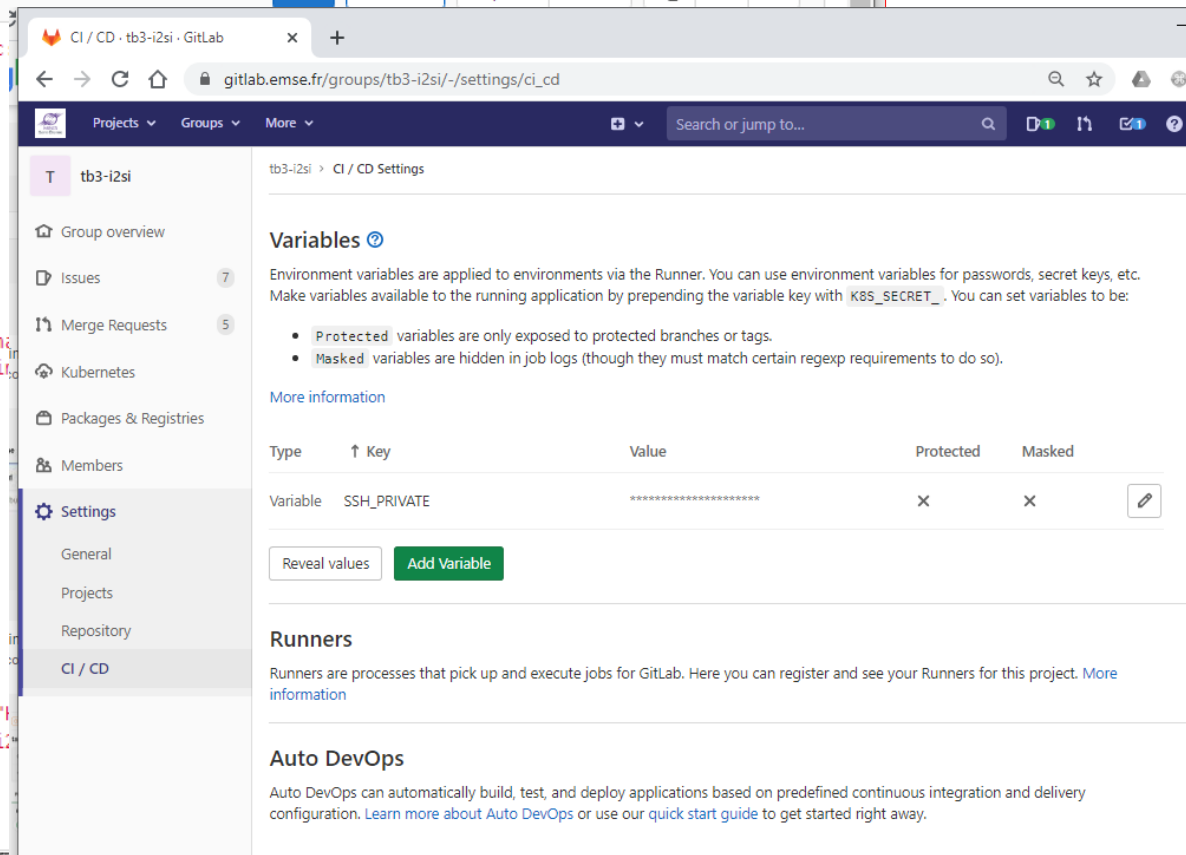
Variable

- At the project level
- Group level
- By default

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

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





Environments

Environments allow control of the continuous deployment of your software, all within GitLab.

Introduction

There are many stages required in the software development process before the software is ready for public consumption.

For example:

1. Develop your code.
2. Test your code.
3. Deploy your code into a testing or staging environment before you release it to the public.

This helps find bugs in your software, and also in the deployment process as well.

GitLab CI/CD is capable of not only testing or building your projects, but also deploying them in your infrastructure, with the added benefit of giving you a way to track your deployments. In other words, you will always know what is currently being deployed or has been deployed on your servers.

It's important to know that:

- Environments are like tags for your CI jobs, describing where code gets deployed.
- Deployments are created when GitLab CI/CD is used to deploy versions of code to environments.

GitLab:

- Provides a full history of your deployments for each environment.
- Keeps track of your deployments, so you always know what is currently being deployed on your servers.

If you have a deployment service such as Kubernetes associated with your project, you can use it to assist with your deployments, and can even access a web terminal for your environment from within GitLab!

https://docs.gitlab.com/ee/ci/environments/index.html

Configuration of the pipeline execution

- Scheduled execution
- Configuration in the `.gitlab-ci.yml` file
- Configuration for a Job
  - tags: *to choose the "runner"*
  - stage: *in which stage the job is run*
  - variables: *define job variables on a job level.*
  - dependencies: *list of jobs whose artifacts will be used*
  - when: *When to run job. Also available: when:manual and when:delayed.*
  - allow_failure: *false (by default), true*
  - script: *the shell script(s) to execute.*

  - only: *limit when jobs are created. Also available: only:refs, only:kubernetes, only:variables, and only:changes.*
  - except: *limit when jobs are not created. Also available: except:refs, except:kubernetes, except:variables, and except:changes.*
  - image: *docker image to use name:tag*
  - services: *use Docker services images. Also available: services:name, services:alias, services:entrypoint, and services:command*.

https://docs.gitlab.com/ee/ci/yaml/

Configuration of the pipeline execution

- Scheduled execution
- Configuration in the `.gitlab-ci.yml` file
- Configuration for a Job
  - only: *limit when jobs are created. Also available: only:refs, only:kubernetes, only:variables, and only:changes.*
  - except: *limit when jobs are not created. Also available: except:refs, except:kubernetes, except:variables, and except:changes.*

`only` and `except` are two keywords that set a job policy to limit when jobs are created:

1. `only` defines the names of branches and tags the job runs for.
2. `except` defines the names of branches and tags the job does **not** run for.

There are a few rules that apply to the usage of job policy:

- `only` and `except` are inclusive. If both `only` and `except` are defined in a job specification, the ref is filtered by `only` and `except`.
- `only` and `except` allow the use of regular expressions (supported regexp syntax).
- `only` and `except` allow to specify a repository path to filter jobs for forks.

https://docs.gitlab.com/ee/ci/yaml/

Configuration of the pipeline execution

- Scheduled execution
- Configuration in the `.gitlab-ci.yml` file
- Configuration for a Job
  - only: *limit when jobs are created. Also available: only:refs, on*
  - except: *limit when jobs are not created. Also available: except*

`only` and `except` are two keywords that set a job policy to

1. `only` defines the names of branches and tags the job
2. `except` defines the names of branches and tags the jo

There are a few rules that apply to the usage of job policy:

- `only` and `except` are inclusive. If both `only` and `exc` `except`.
- `only` and `except` allow the use of regular expressions
- `only` and `except` allow to specify a repository path to

https://docs.gitlab.com/ee/ci/yaml/

In addition, `only` and `except` allow the use of special keywords:

Value	Description
`api`	For pipelines triggered by the pipelines API.
`branches`	When the Git reference for a pipeline is a branch.
`chat`	For pipelines created by using a GitLab ChatOps command.
`external`	When using CI services other than GitLab.
`external_pull_requests`	When an external pull request on GitHub is created or updated (See Pipelines for external pull requests).
`merge_requests`	For pipelines created when a merge request is created or updated. Enables merge request pipelines, merged results pipelines, and merge trains.
`pipelines`	For multi-project pipelines created by using the API with `CI_JOB_TOKEN`, or the `trigger` keyword.
`pushes`	For pipelines triggered by a `git push` event, including for branches and tags.
`schedules`	For scheduled pipelines.
`tags`	When the Git reference for a pipeline is a tag.
`triggers`	For pipelines created by using a trigger token.
`web`	For pipelines created by using **Run pipeline** button in the GitLab UI, from the project's **CI/CD** > **Pipelines** section.

Configuration of the pipeline execution

- Scheduled execution
- Configuration in the `.gitlab-ci.yml` file
- Configuration for a Job
  - image: *docker image to use name:tag*
  - services: *use Docker services images. Also available: services:name, services:alias, services:entrypoint, and services:command*.

https://docs.gitlab.com/ee/ci/yaml/

Configuration de l'exécution du pipeline

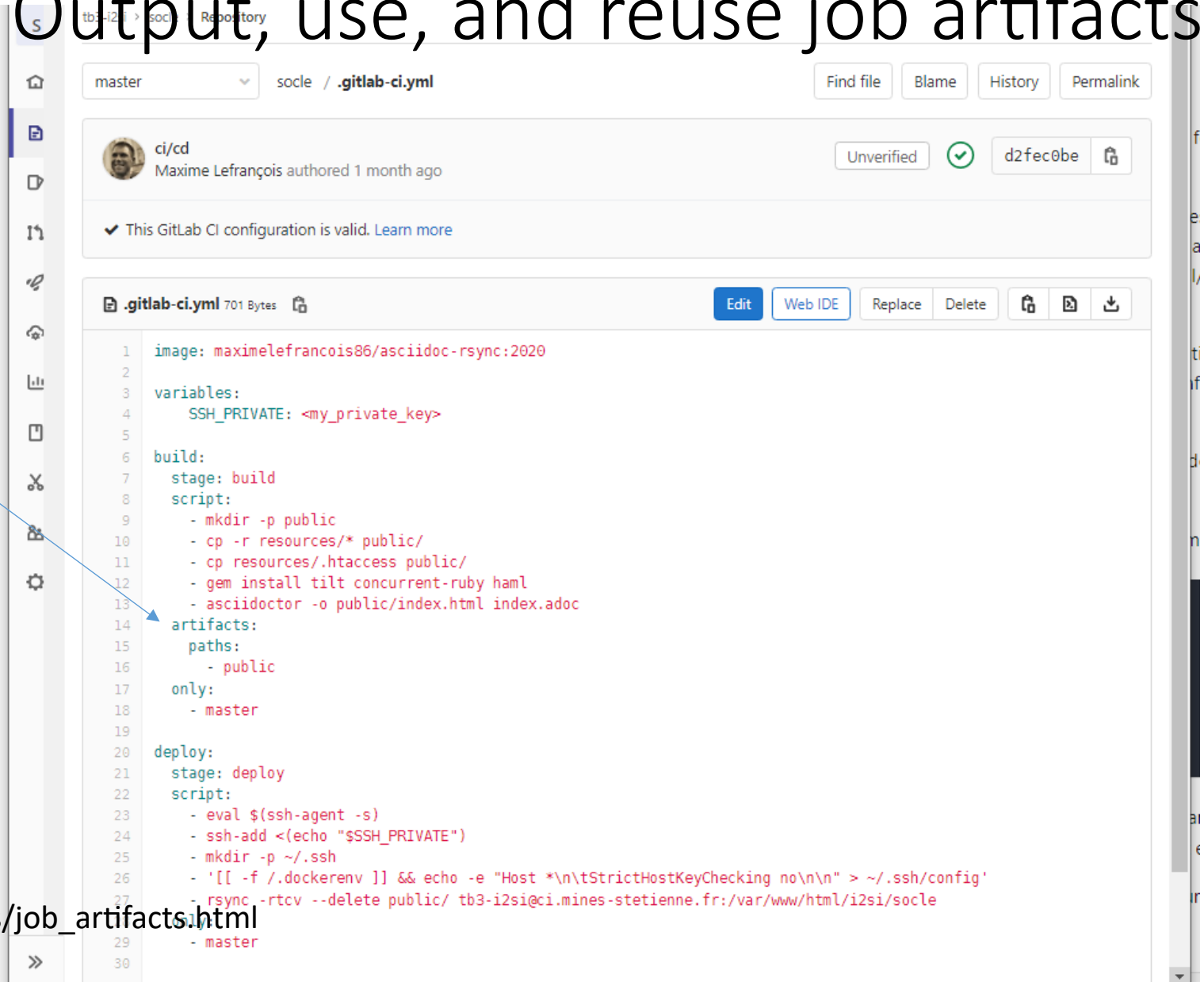
- Scheduled execution
- Configuration in the `.gitlab-ci.yml` file
- Configuration for a Job
  - before_script
  - after_script
  - cache: *List of files that should be cached between subsequent runs. Also available: cache:paths, cache:key, cache:untracked, cache:when, and cache:policy.*
  - variables
  - image: defines the default image
  - services : defines the default services

https://docs.gitlab.com/ee/ci/yaml/

Job artifacts: Output, use, and reuse job artifacts.

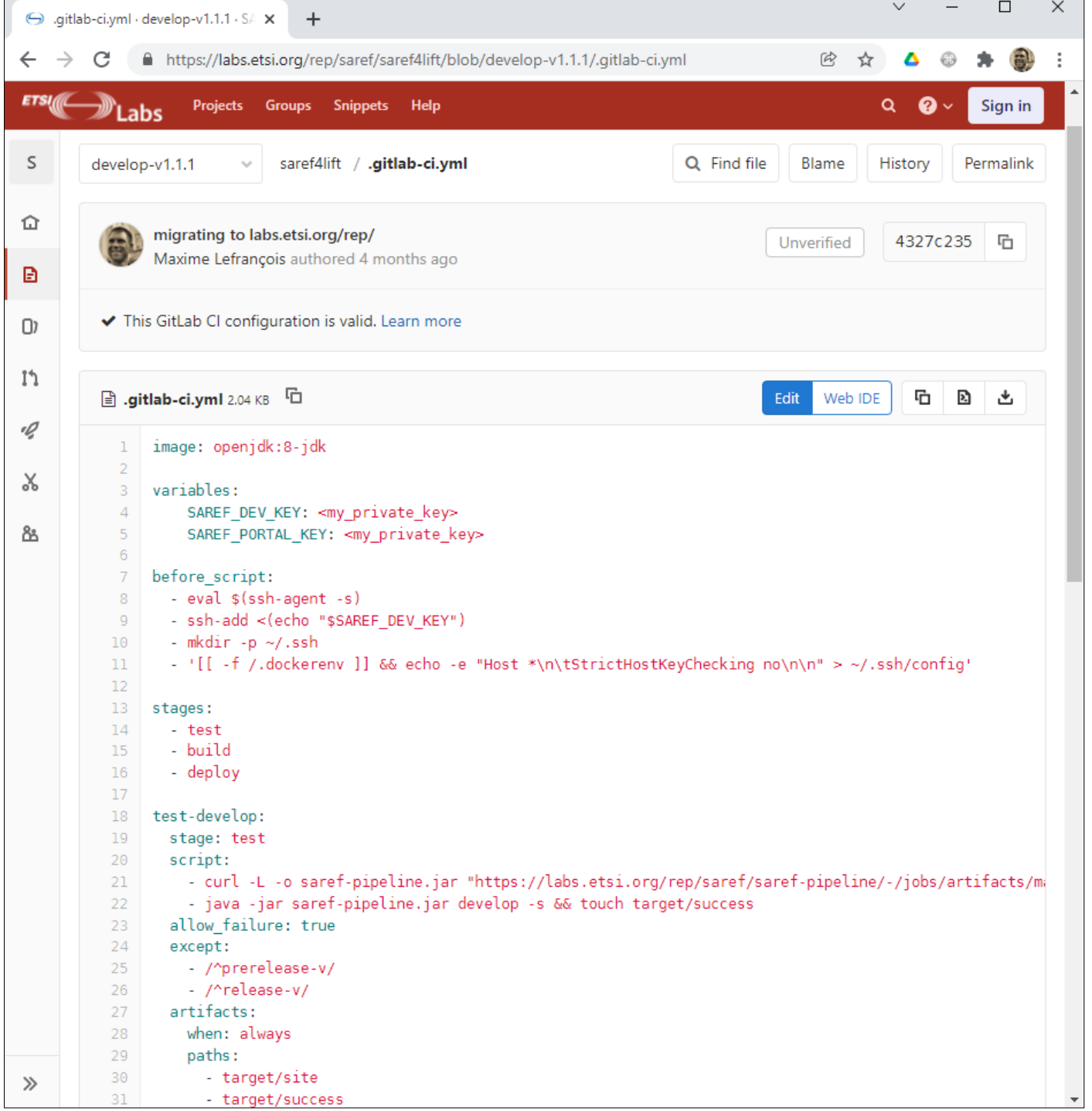
Artifact

Archive available on success.
[Configuration](#) (sub-list):
artifacts:paths,
artifacts:exclude,
artifacts:expose_as,
artifacts:name,
artifacts:untracked,
artifacts:when,
artifacts:expire_in, and
[artifacts:reports](#).



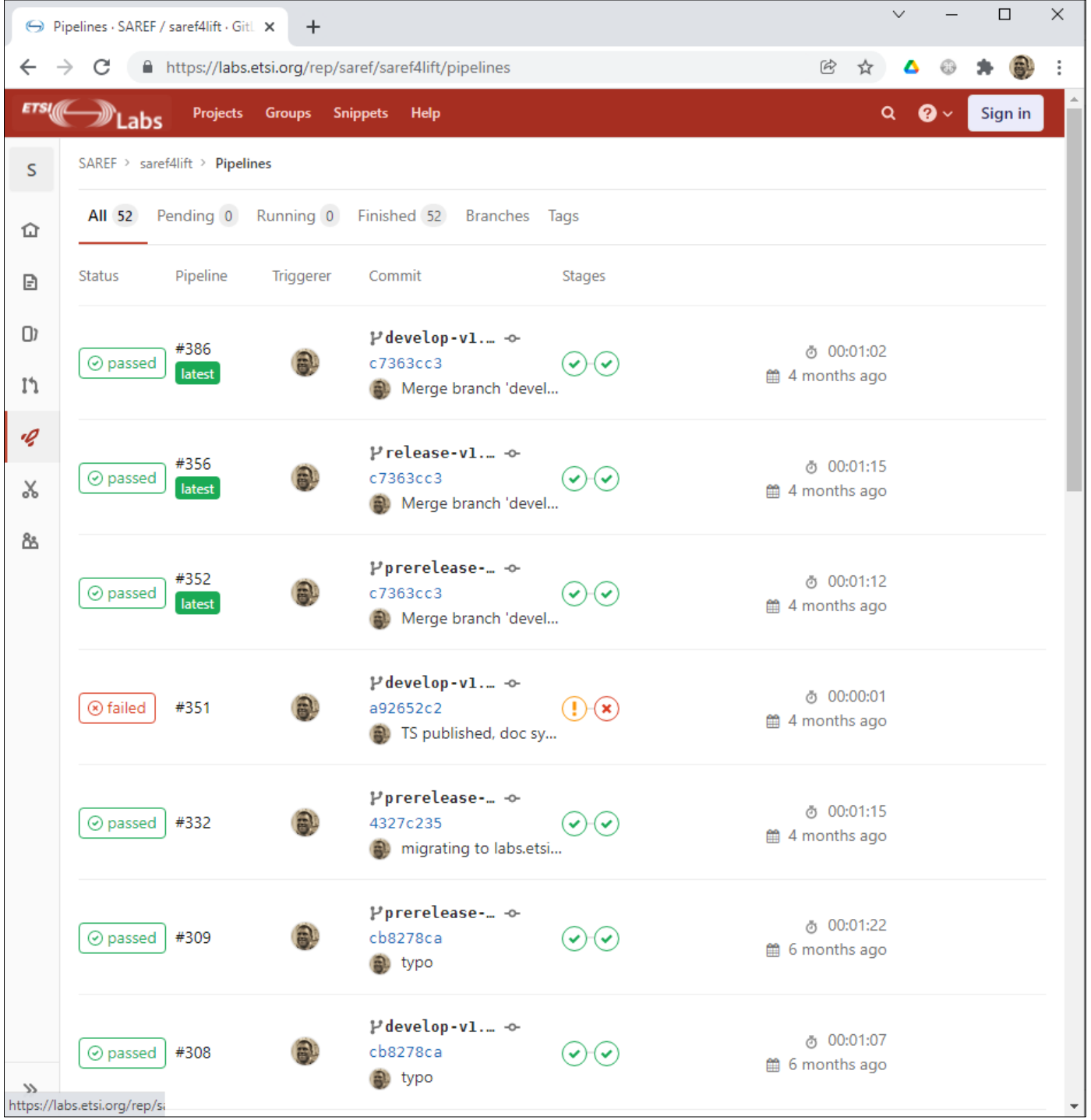
https://docs.gitlab.com/ee/ci/pipelines/job_artifacts.html

Example



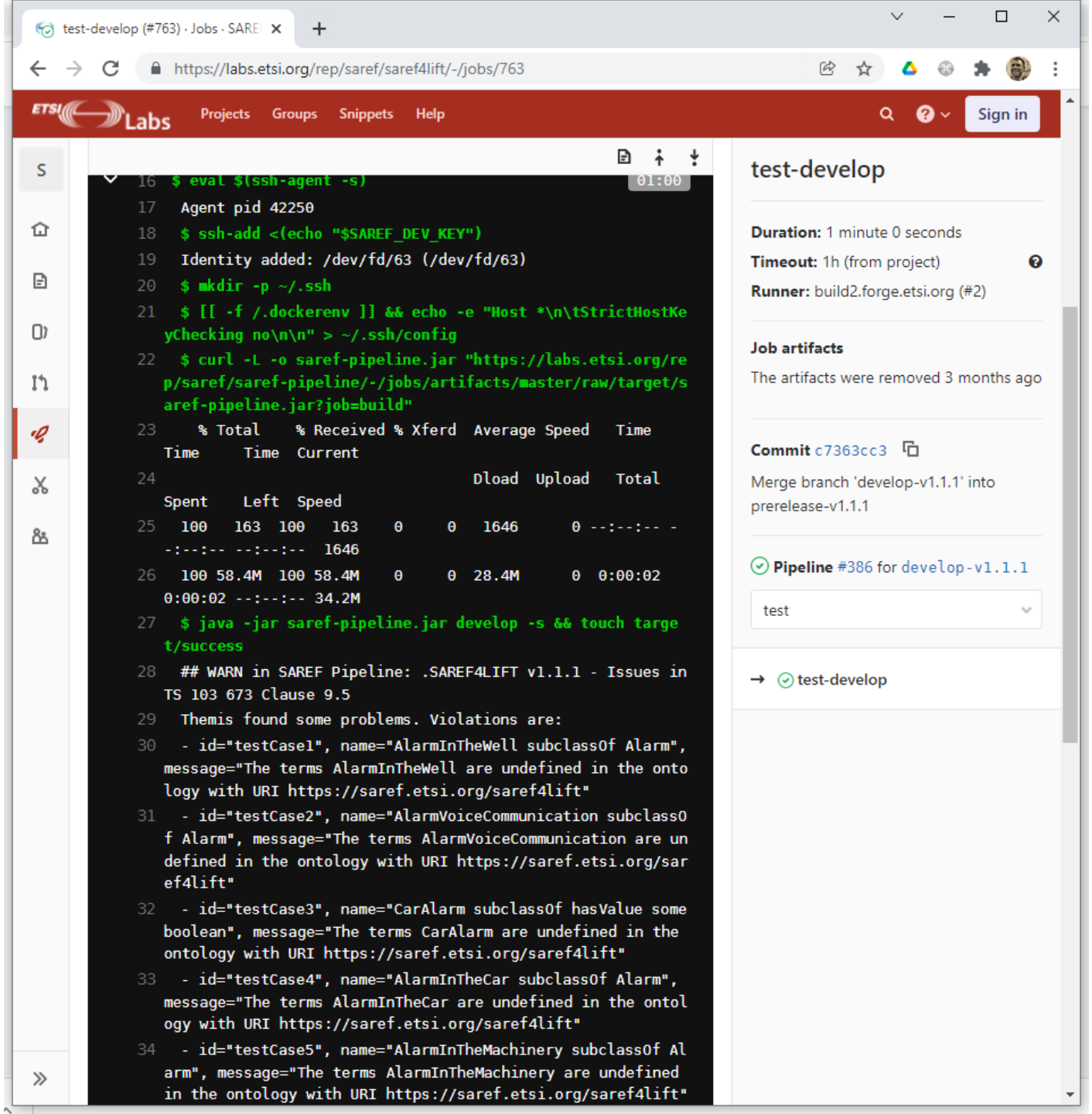
https://saref.etsi.org/sources/saref4lift/

Example



https://saref.etsi.org/sources/saref4lift/

Example



https://saref.etsi.org/sources/saref4lift/

Technological foundations of software development

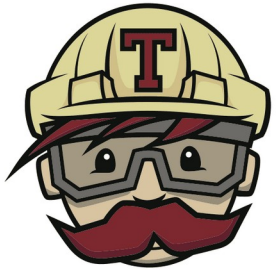
Integrate and deploy your software continuously

Part 3: GitHub Actions

ICM – Computer Science Major – Course unit on Technological foundations of computer scienceopment
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

GitHub Actions: GitHub's response

Circle CI
2011
https://circleci.com/
YAML
Supports Docker



Travis CI
2011
http://travis-ci.com/
YAML
Supports Docker

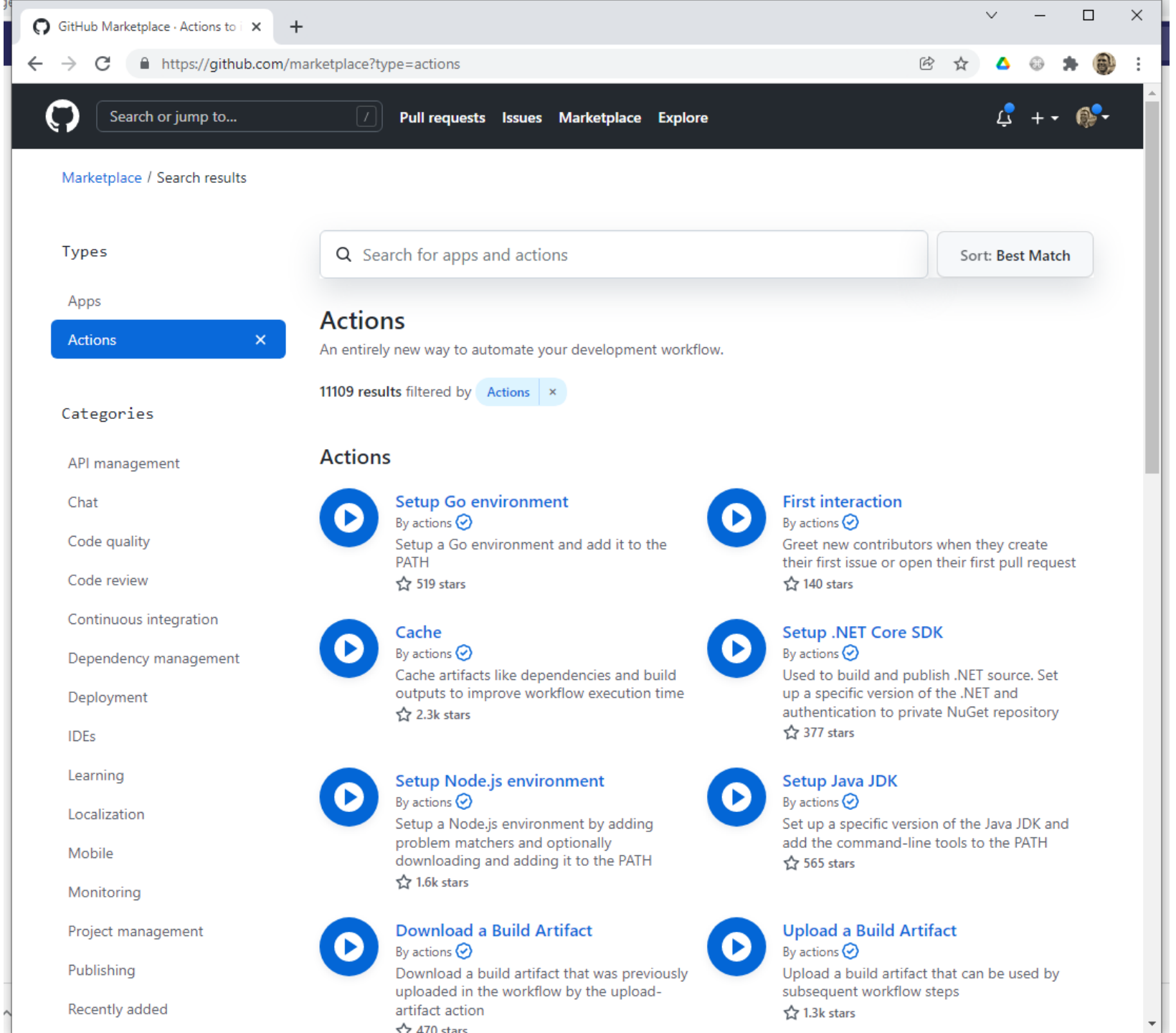
Jenkins
2011
https://jenkins.io/
Groovy
Supports Docker

GitLab CI/CD
2013
https://gitlab.com/
YAML
Supports Docker

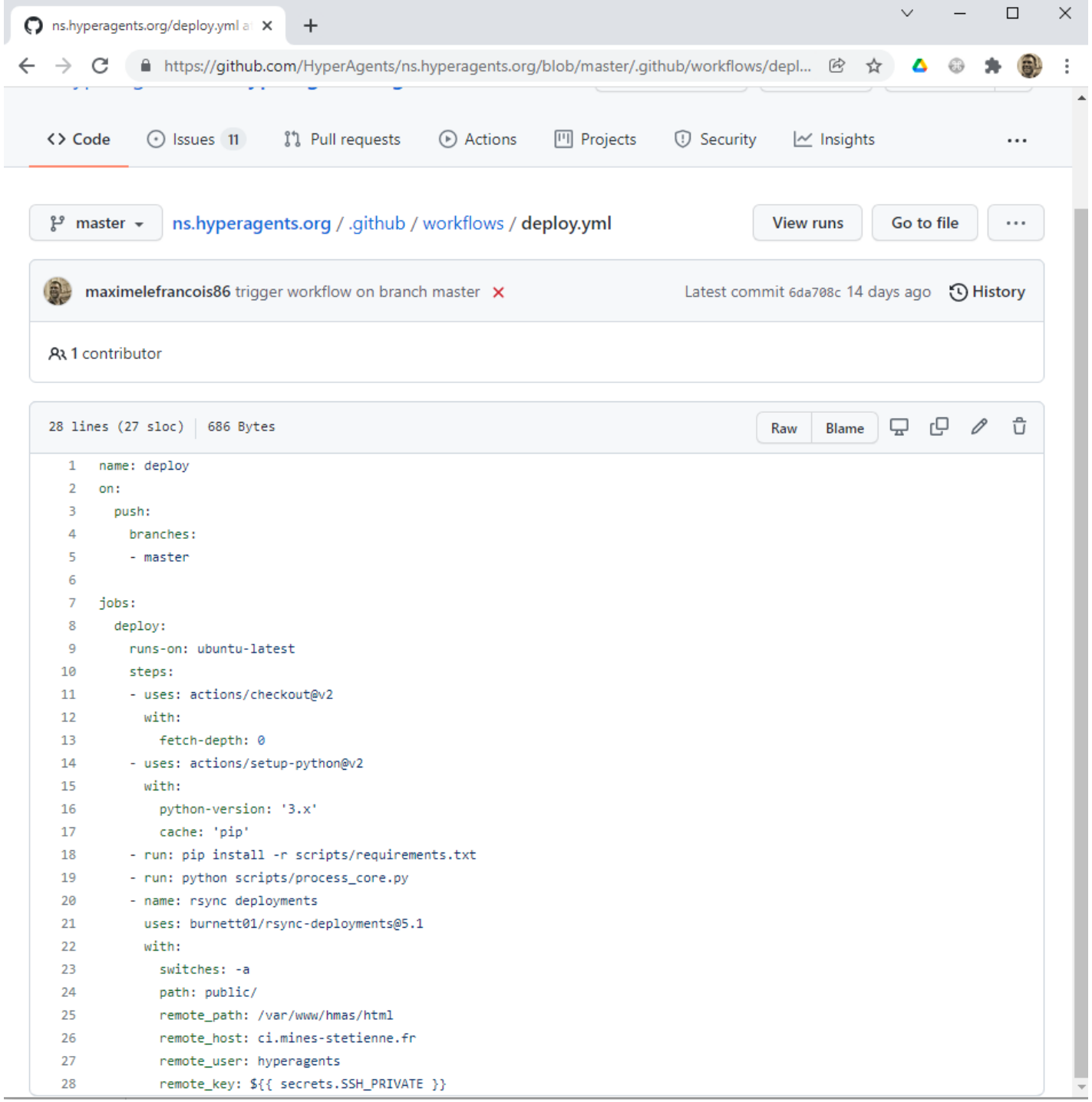
GitHub Actions
2018
https://github.com/features/actions
YAML
Supports Docker

GitHub Flow with GitHub Actions

Find actions

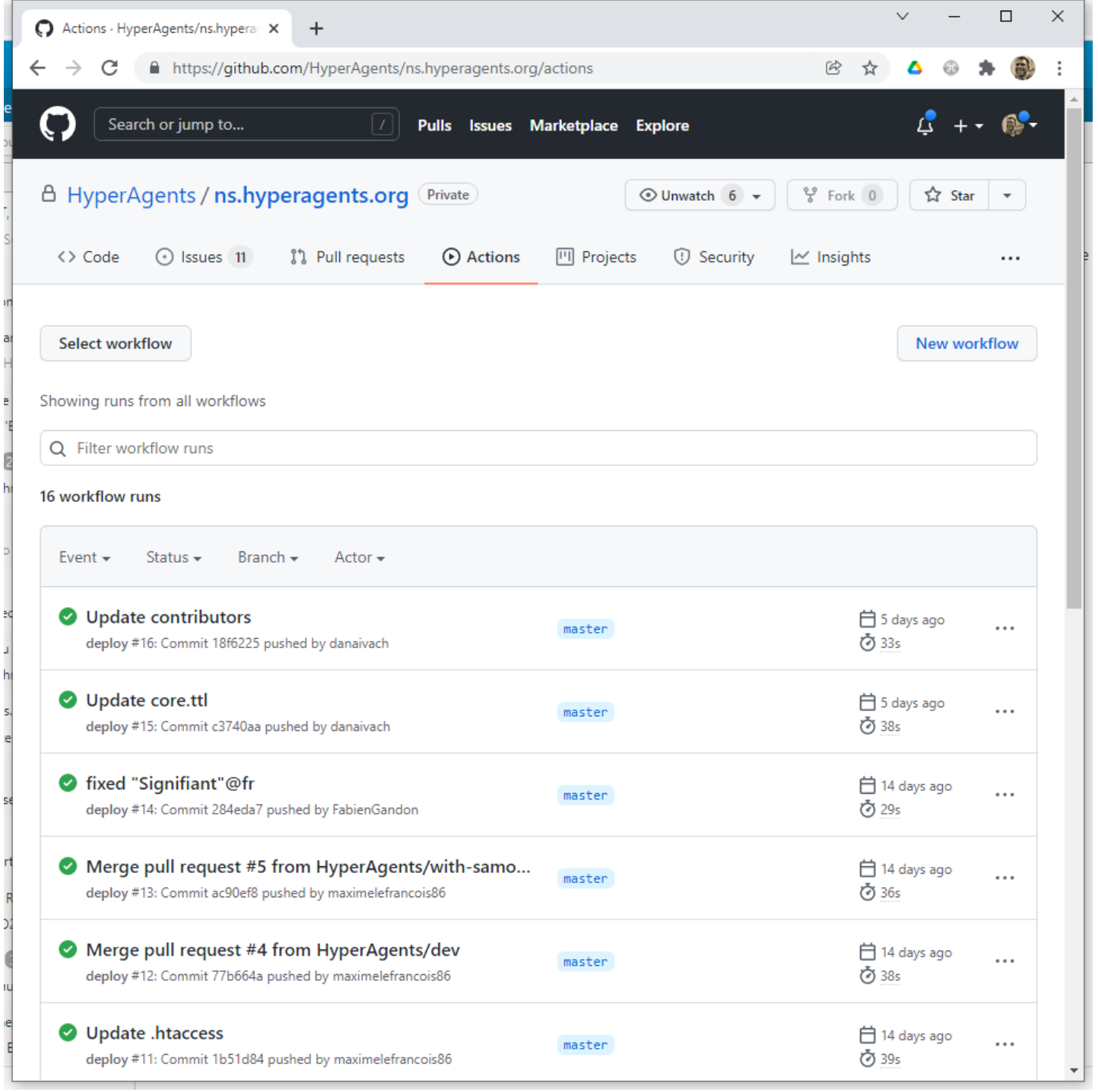


Example



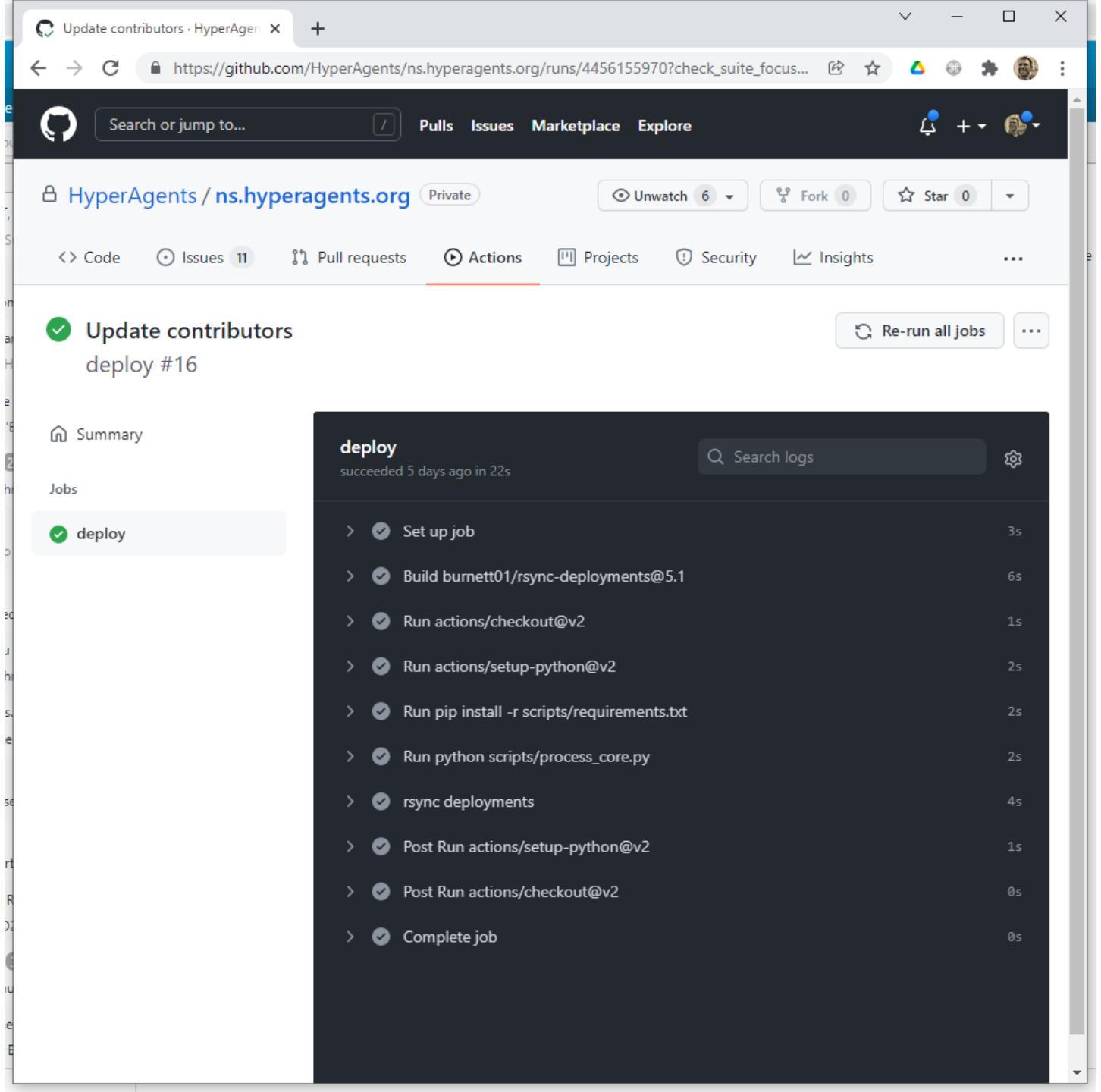
https://github.com/HyperAgents/ns.hyperagents.org/

Example



https://github.com/HyperAgents/ns.hyperagents.org/

Example



https://github.com/HyperAgents/ns.hyperagents.org/

Technological foundations of software development

Integrate and deploy your software continuously

ICM – Computer Science Major – Course unit on Technological foundations of computer scienceopment
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

… Your turn

Complete the TODO section:

https://ci.mines-stetienne.fr/cps2/course/tfsd/course-8.html#_todos