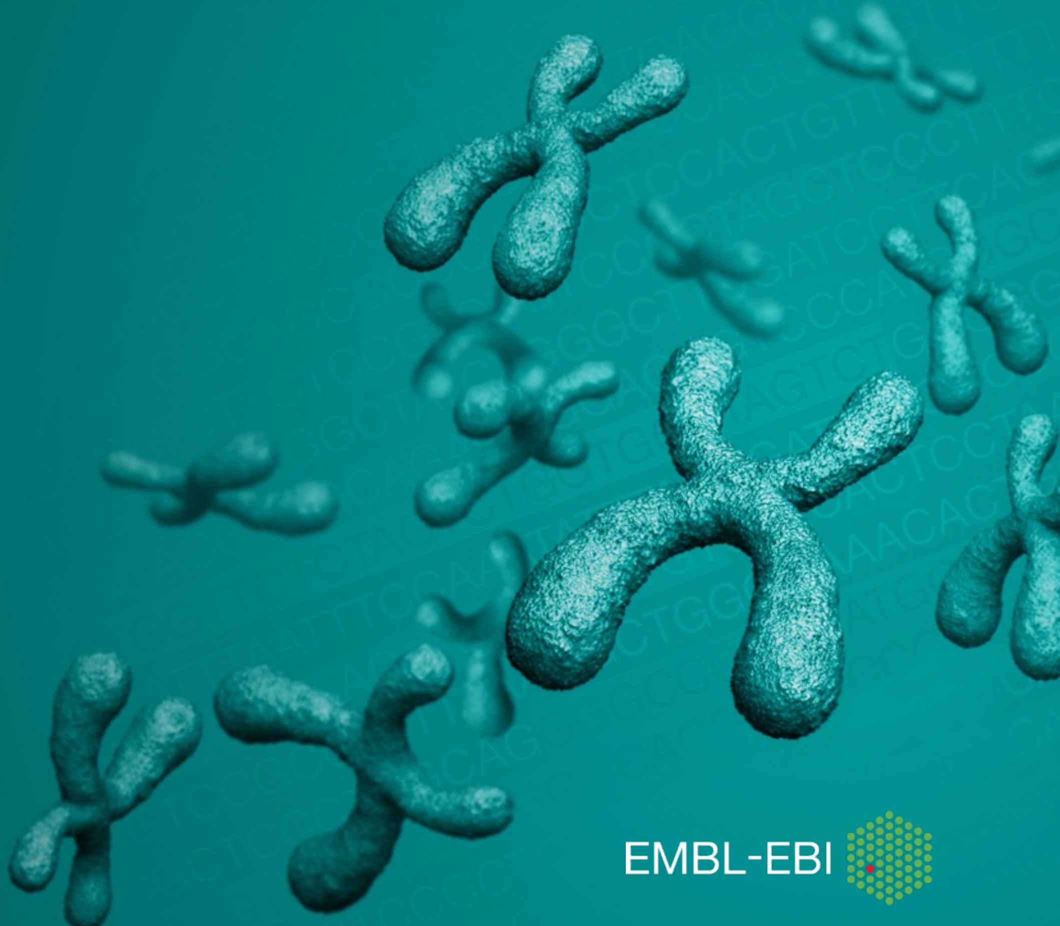
Continuous Integration with Gitlab

Tony Wildish

Cloud Bioinformatics Application Architect

EMBL-EBI



EMBL-EBI

Introduction to Gitlab

- Gitlab for Continuous Integration/Continuous Deployment
- Hands-on session
  - A ‘hello world’ tour of the basics
- Not covered:
  - Setting up your own runners
  - ‘AutoDevOps’: awesome for DevOps environments
    - https://docs.gitlab.com/ee/topics/autodevops/
- Pre-requisites:
  - Basic knowledge of git and an understanding of docker

Gitlab is…

- A git-based code hosting service
  - Like github.com, bitbucket.com, and many others
  - SCM, Wiki, issue-tracking, project/team-management…
- A continuous integration (CI) platform
  - Like Travis, Jenkins, and others
  - You commit/tag code, gitlab builds, tests, packages and deploys it
    - You tell it how! That's what this talk is about
  - Distributed builds, can use many platforms
    - Laptop/desktop, cloud (AWS, GCP)
    - Can even use multiple platforms in the same build

Gitlab components

- Gitlab server
  - The hosting service
  - Project management components
  - CI build system management (how ‘runners’ are used)
- Gitlab runners
  - User-space daemons that execute builds
  - Driven by the server on pushing to the repository
  - Highly configurable, can have many runners per repo, different compilers, runtimes, OS…
  - Can run anywhere: laptop, cloud, Embassy

Gitlab server

- Two editions
  - CE: Community Edition (free, self-hosted)
  - EE: Enterprise Edition (paid, self-hosted or cloud-hosted)
  - **gitlab.com** (EE, free)
    - Unlimited repositories, private or public
    - 10 GB disk space per project
    - Mirroring external public repositories has up to an hours latency
- We have the Enterprise Edition at **gitlab.ebi.ac.uk**
  - Requires 2FA, which is a pain for getting started, so we use **gitlab.com** today

Gitlab runner

- Can run on any platform
  - Laptop, AWS/GCP, Embassy etc
  - Configure runners per project
    - Can share runners between projects, or be project-specific
    - **gitlab.com** provides shared runners, all ready to use!
    - **gitlab.ebi.ac.uk** has shared runners, but you are expected to provide your own for production deployments
- Specify runners capabilities with tags when you register them
  - E.g. gcc/python/perl version, system capabilities (RAM, cores)

Gitlab runner

- At build-time
  - Server chooses runners based on tags in config file – per step!
  - Server launches as many build processes as required
  - Can store products from each step back to server, for inspection later on or for use in subsequent steps

- Each runner can run a custom workflow
  - Infinitely configurable, per project
  - Workflow specified in YAML config file in the project repository

Gitlab runner

- Security
  - Gitlab runners have significant security implications
  - Will dutifully execute all instructions from the .gitlab-ci.yml file
  - Malicious users can inject dangerous commands
    - E.g. **rm –rf $HOME**
  - Control who has access to the .gitlab-ci.yml file
    - Use fork/pull model, not direct commit
  - Run runners as unprivileged users on dedicated infrastructure
    - Not as you in your home directory!

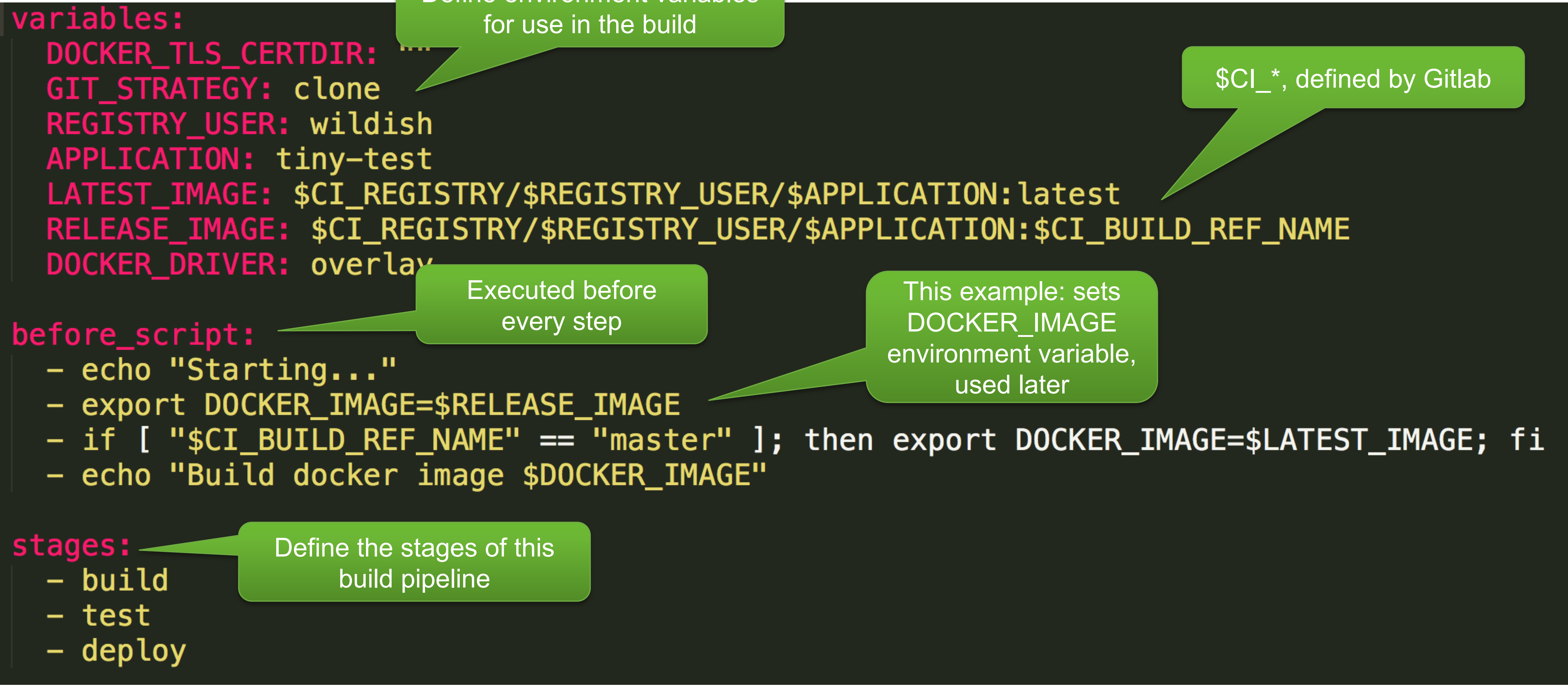
EMBL-EBI

Gitlab and Docker

- Many possible combinations…
  - Q: Can I do X with Docker and Gitlab? A: Yes, for all X!
- Run Gitlab Runner in a Docker container
- Pull/run Docker containers to *execute* your CI job
  - Use different docker containers per step
- Build Docker containers *inside* your CI job
  - Push them to Gitlab Container Registry or elsewhere
- Gitlab Container Registry
  - Integrated Docker registry, upload a container from your CI job
  - Can automatically tag with branch name/version etc

The CI configuration file

- Standard YAML
  - **.gitlab-ci.yml**, in the top directory of your git repository
  - Describes **pipelines** which consist of **stages**, run by one or more **steps**
  - Each **stage** has a specific purpose: **build, test, deploy**…
  - Each **stage** can have its own **tags** (i.e. Its own required environment)
  - Each **stage** can produce **artifacts**/re-use from other stages
  - Stages run sequentially, steps can run in parallel
    - Each **step** in a **stage** must complete before the next **stage** can start
    - Each **step** in a **stage** must succeed or the whole pipeline will fail
- Similar to makefiles in some ways
  - Specify dependencies & actions, not explicitly coding workflows



```
variables:
  DOCKER_TLS_CERTDIR: ""
  GIT_STRATEGY: clone
  REGISTRY_USER: wildish
  APPLICATION: tiny-test
  LATEST_IMAGE: $CI_REGISTRY/$REGISTRY_USER/$APPLICATION:latest
  RELEASE_IMAGE: $CI_REGISTRY/$REGISTRY_USER/$APPLICATION:$CI_BUILD_REF_NAME
  DOCKER_DRIVER: overlay

before_script:
  - echo "Starting..."
  - export DOCKER_IMAGE=$RELEASE_IMAGE
  - if [ "$CI_BUILD_REF_NAME" == "master" ]; then export DOCKER_IMAGE=$LATEST_IMAGE; fi
  - echo "Build docker image $DOCKER_IMAGE"

stages:
  - build
  - test
  - deploy
```

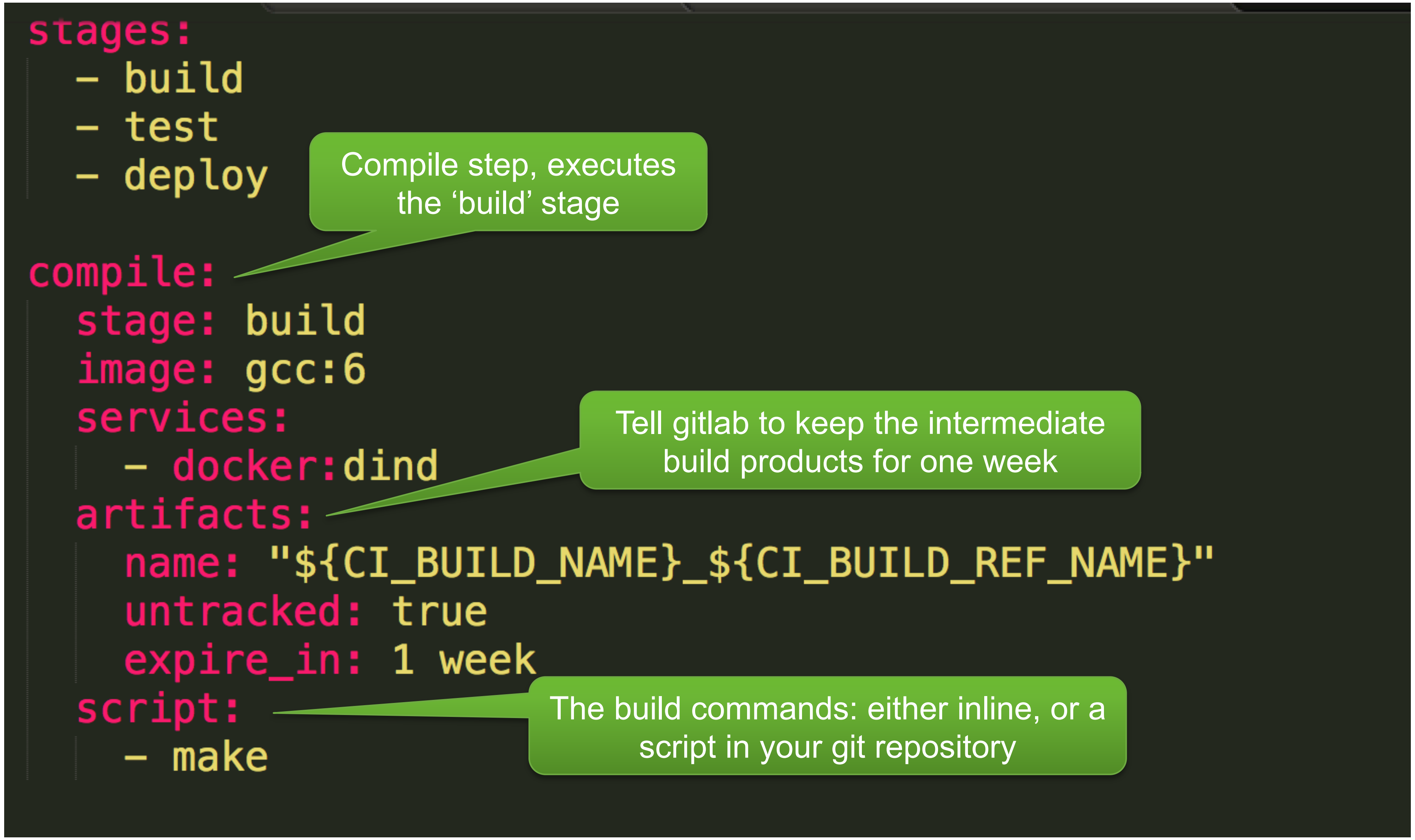
Define environment variables for use in the build

$CI_*, defined by Gitlab

Executed before every step

This example: sets DOCKER_IMAGE environment variable, used later

Define the stages of this build pipeline



```
stages:
  - build
  - test
  - deploy

compile:
  stage: build
  image: gcc:6
  services:
    - docker:dind
  artifacts:
    name: "${CI_BUILD_NAME}_${CI_BUILD_REF_NAME}"
    untracked: true
    expire_in: 1 week
  script:
    - make
```

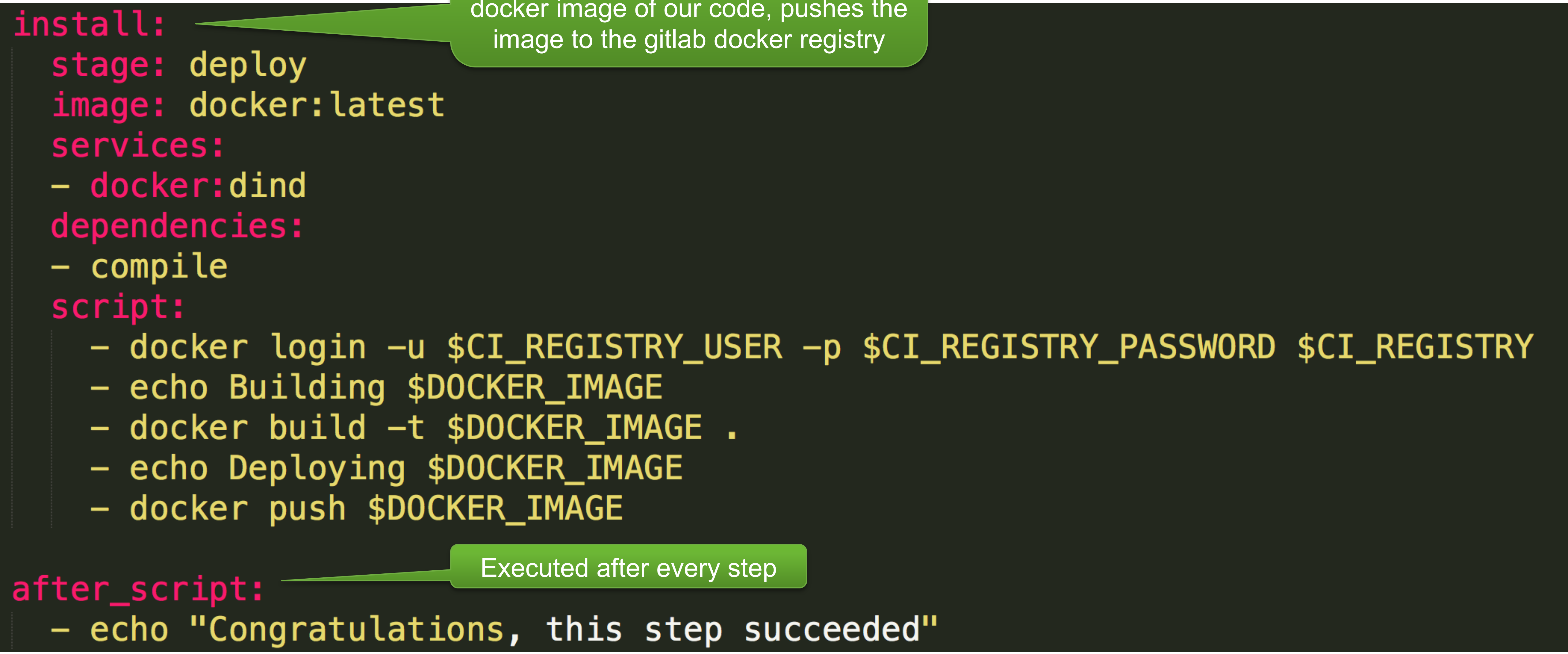
Compile step, executes the 'build' stage
Tell gitlab to keep the intermediate build products for one week
The build commands: either inline, or a script in your git repository

Run step executes the ‘test’ stage. Depends on the ‘compile’ step, gets its artifacts automatically

Only runs for git-tagged versions

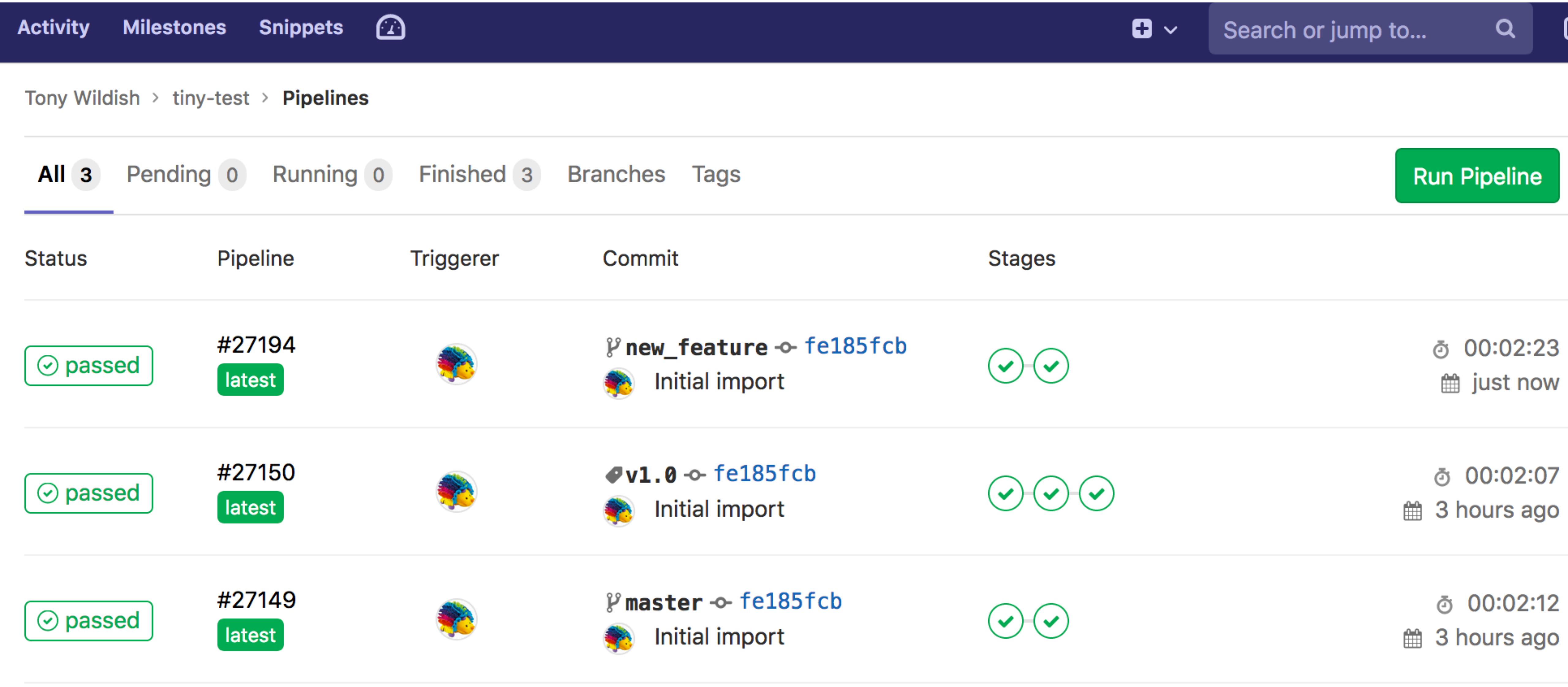
```
run:
  stage: test
  dependencies:
  - compile
  only:
    - tags
  script:
    - echo "Testing application. First, list the files here, to show we have the git repo + the artifacts fro
    - ls -l
    - echo 'Now try running it'
    - ./hello
    - echo "If that failed you won't see this because you'll have died already"
```

Install step runs the ‘deploy’ stage. Runs a docker container to build a docker image of our code, pushes the image to the gitlab docker registry



```
install:
  stage: deploy
  image: docker:latest
  services:
  - docker:dind
  dependencies:
  - compile
  script:
    - docker login -u $CI_REGISTRY_USER -p $CI_REGISTRY_PASSWORD $CI_REGISTRY
    - echo Building $DOCKER_IMAGE
    - docker build -t $DOCKER_IMAGE .
    - echo Deploying $DOCKER_IMAGE
    - docker push $DOCKER_IMAGE

after_script:
  - echo "Congratulations, this step succeeded"
```

Executed after every step



Activity Milestones Snippets

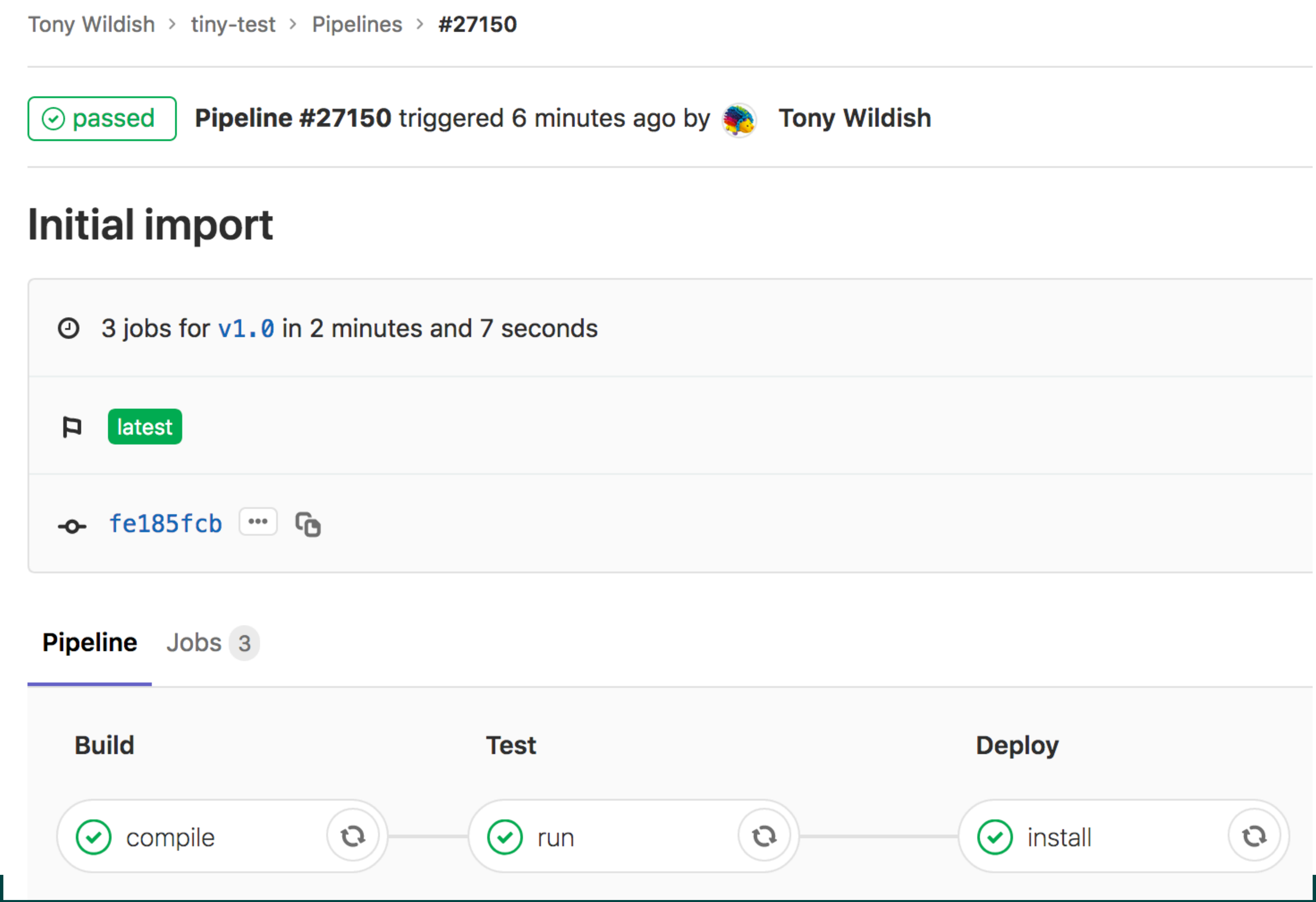
Search or jump to...

Tony Wildish › tiny-test › Pipelines

All 3 Pending 0 Running 0 Finished 3 Branches Tags

Run Pipeline

Status	Pipeline	Triggerer	Commit	Stages	
✓ passed	#27194 latest		new_feature fe185fcb Initial import	✓ ✓	00:02:23 just now
✓ passed	#27150 latest		v1.0 fe185fcb Initial import	✓ ✓ ✓	00:02:07 3 hours ago
✓ passed	#27149 latest		master fe185fcb Initial import	✓ ✓	00:02:12 3 hours ago


Tony Wildish › tiny-test › Pipelines › #27150

passed **Pipeline #27150** triggered 6 minutes ago by **Tony Wildish**

Initial import

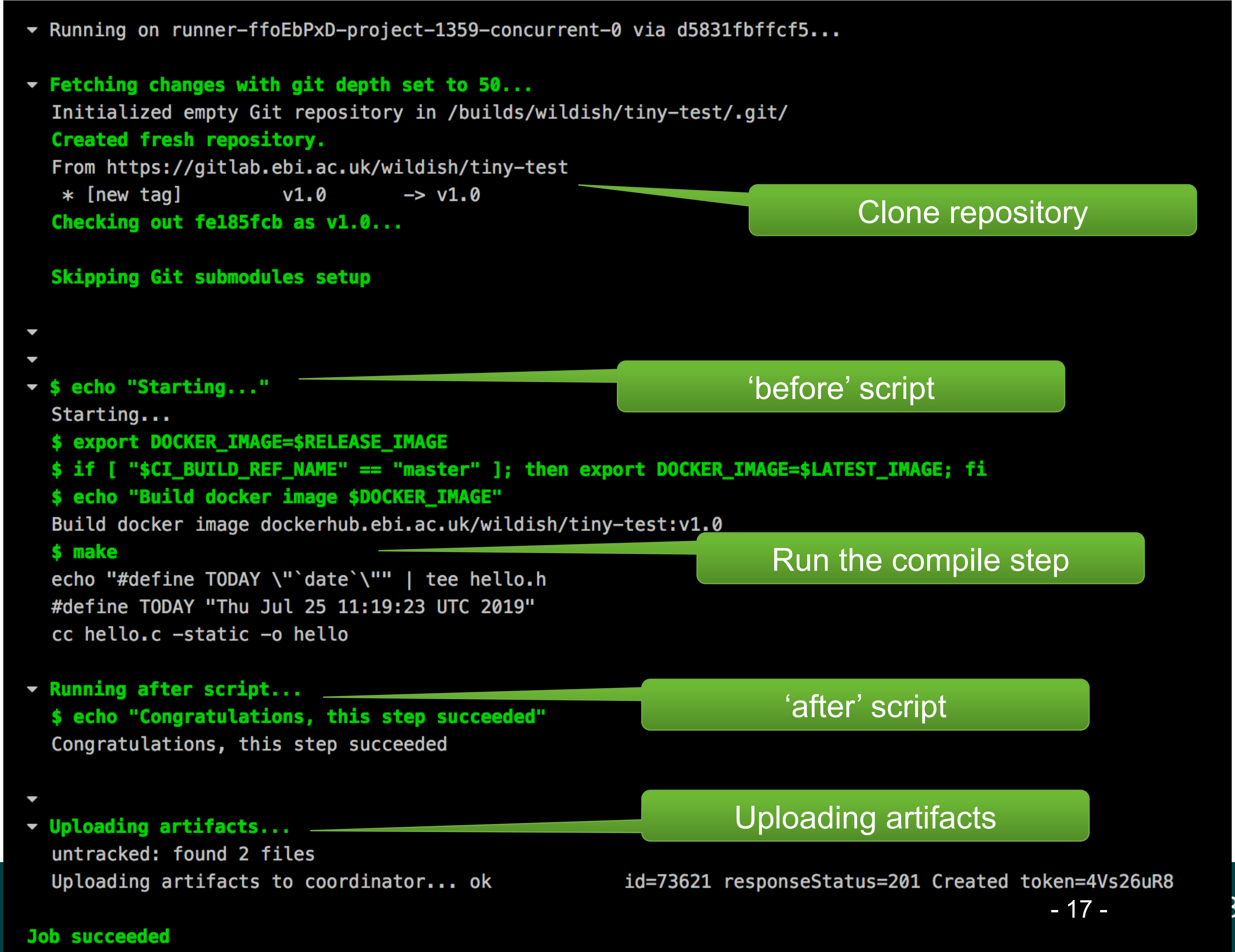
3 jobs for v1.0 in 2 minutes and 7 seconds

latest

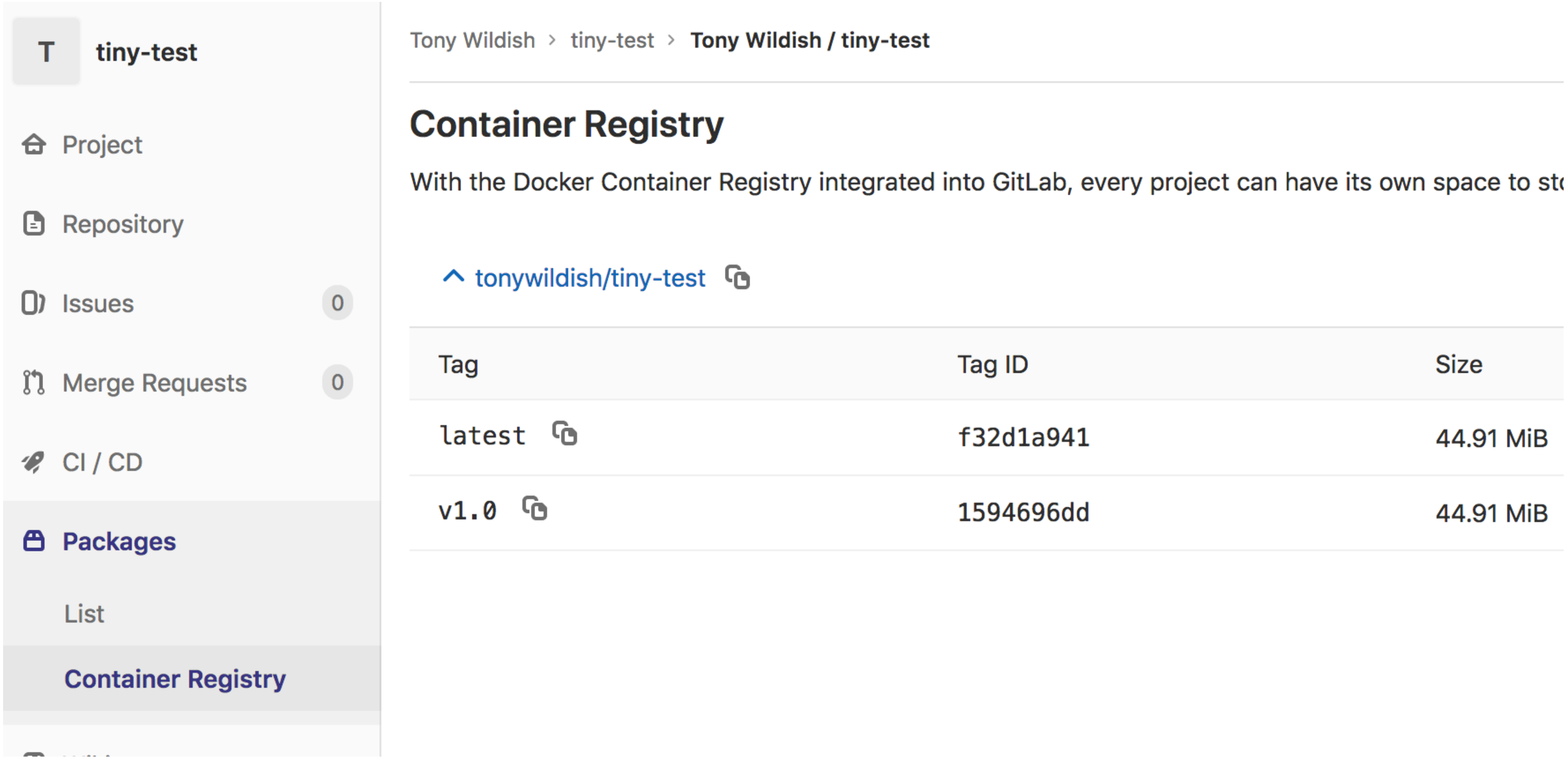
fe185fcb

Pipeline Jobs 3

Build	**Test**	**Deploy**
✓ compile	✓ run	✓ install



```
▾ Running on runner-ffoEbPxD-project-1359-concurrent-0 via d5831fbffcf5...

▾ Fetching changes with git depth set to 50...
  Initialized empty Git repository in /builds/wildish/tiny-test/.git/
  Created fresh repository.
  From https://gitlab.ebi.ac.uk/wildish/tiny-test
   * [new tag]         v1.0       -> v1.0
  Checking out fe185fcb as v1.0...

  Skipping Git submodules setup


▾
▾
▾ $ echo "Starting..."
  Starting...
  $ export DOCKER_IMAGE=$RELEASE_IMAGE
  $ if [ "$CI_BUILD_REF_NAME" == "master" ]; then export DOCKER_IMAGE=$LATEST_IMAGE; fi
  $ echo "Build docker image $DOCKER_IMAGE"
  Build docker image dockerhub.ebi.ac.uk/wildish/tiny-test:v1.0
  $ make
  echo "#define TODAY \"`date`\"" | tee hello.h
  #define TODAY "Thu Jul 25 11:19:23 UTC 2019"
  cc hello.c -static -o hello

▾ Running after script...
  $ echo "Congratulations, this step succeeded"
  Congratulations, this step succeeded

▾
▾ Uploading artifacts...
  untracked: found 2 files
  Uploading artifacts to coordinator... ok          id=73621 responseStatus=201 Created token=4Vs26uR8

Job succeeded
```



Tony Wildish › tiny-test › Tony Wildish / tiny-test

Container Registry

With the Docker Container Registry integrated into GitLab, every project can have its own space to st

^ tonywildish/tiny-test

Tag	Tag ID	Size
latest	f32d1a941	44.91 MiB
v1.0	1594696dd	44.91 MiB

EMBL-EBI

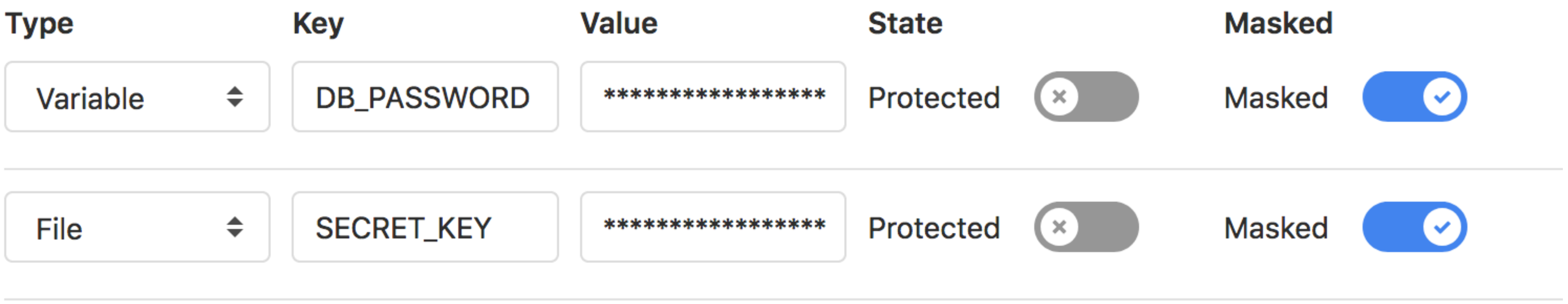
Secrets

- Q: How do you pass a database password to a CI/CD pipeline?
  - 1) Hard-code it in the repository where anyone can see it?
  - 2) Use a **gitlab variable** to pass it to the runner without exposing it?

Secrets

- Q: How do you pass a database password to a CI/CD pipeline?
  - 1) Hard-code it in the repository where anyone can see it?
  - 2) Use a **gitlab variable** to pass it to the runner without exposing it?

- Pass an environment variable, or a file with preset contents
- **Settings** -> **CI/CD** -> **Variables** -> **Expand**
- => Exercise 7



Type	Key	Value	State	Masked
Variable	DB_PASSWORD	****************	Protected ✗	Masked ✓
File	SECRET_KEY	****************	Protected ✗	Masked ✓

EMBL-EBI

Other gitlab features

- API, programmable interface to Gitlab
  - https://docs.gitlab.com/ee/api/
- Build hooks
  - Trigger actions on external services other than gitlab
    - Similar capabilities on github, bitbucket
  - Trigger actions in gitlab from external service
    - E.g. nightly build, regardless of commits
- Mirroring repositories
  - Master repository in bitbucket/github?
  - Can mirror to gitlab, automatically, transparently

AutoDevOps

- AutoDevOps is a fairly new feature from Gitlab
  - Detects the language, application style and structure of your project
  - Automatically defines a CI/CD pipeline for it
  - Can automatically build/test/deploy, right through to production
  - Highly configurable
- Not covered today
  - See https://docs.gitlab.com/ee/topics/autodevops/ for more
  - The video there is worth it

Best practices, gotchas…

- Be careful with environment variables
  - Gitlab sets some secret environment variables (API keys etc) for you to use in your builds
  - If you echo them to your logfiles, they will be visible on the web
- Check your YAML configuration file for errors
  - Your-project-page -> CI/CD -> Pipelines -> "CI Lint" (top-right): can edit live and validate
- Set your artifacts to expire
  - Stuff you want to keep should be properly deployed, e.g. in a Docker image
- Keep your build environments clean, simple
  - Unix configure, make, make-test, make-install is a de-facto standard
  - Tag your own runners to specify requirements, avoid complex runtime scripts
- Control access to your repositories
  - Don't give out *any* tokens of any sort, until you've thought through the consequences
  - Don't give others admin/developer-access to the project, use the fork/pull model instead

Exercises

- Go to http://bit.ly/resops-2019
- Click on **‘Gitlab Practical’**
- Follow the exercises