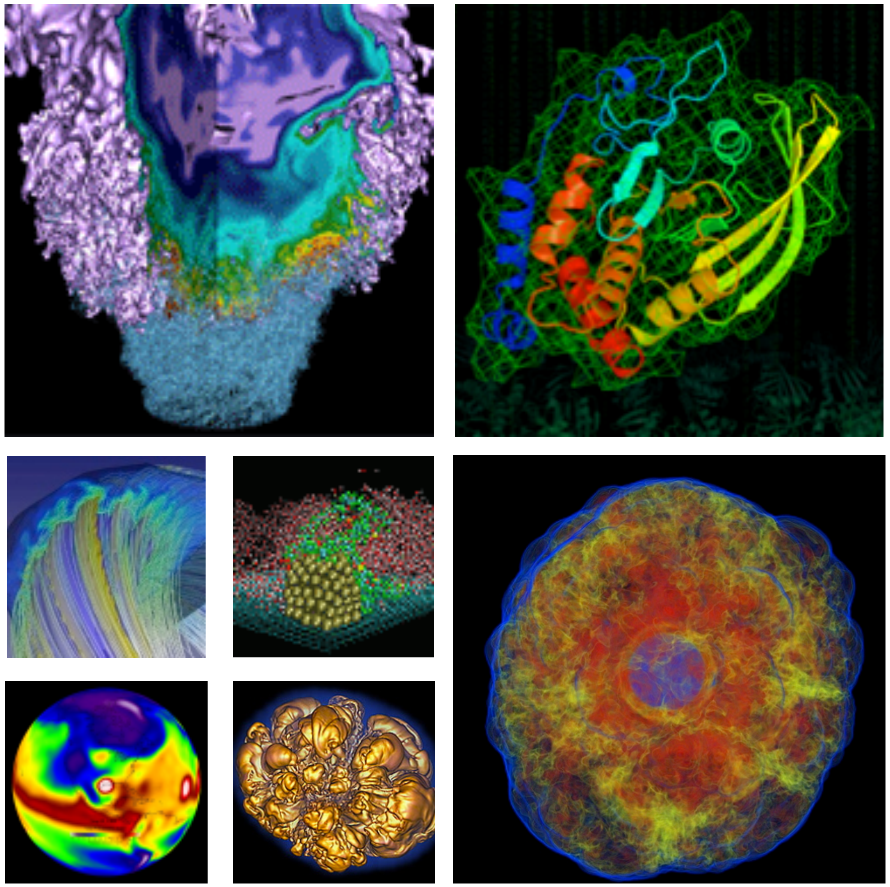
Continuous Integration with Gitlab



NERSC

Tony Wildish

Feb 6th 2017

U.S. DEPARTMENT OF ENERGY | Office of Science

BERKELEY LAB
Lawrence Berkeley National Laboratory

Today's session…

- http://bit.ly/2kAuhFo

Today's session…

- **Introduction to Gitlab**
- **Gitlab for Continuous Integration**
- **Hands-on session**
  - A 'hello world' tour of the basics
- **Aside - that incident, and what you can do about it**
  - Thank you Onur, Chris, Mario, Patrick, Michael, Joel, Alex, Andrew…
  - https://about.gitlab.com/2017/02/01/gitlab-dot-com-database-incident/

- **Pre-requisites:**
  - You will need basic knowledge of git, docker is useful too (e.g. see the git+docker training)
  - This presentation, and other Genepool training material: https://www.nersc.gov/users/computational-systems/genepool/genepool-training-and-tutorials/

Why should you care?

- **Safeguard your code against accidental loss**
  - Like with any git platform, distributed replicas
- **Automate checking that your code compiles**
  - …and works. Can benchmark it too
- **Automate deploying your code**
  - Including Docker containers -> useful for Shifter/cloud
- **Reproducibility!**
  - Know how that data or plot was produced
    - Useful one year from now when the referee starts asking awkward questions about your draft paper
- **Why gitlab, why not bitbucket, Travis, Jenkins…?**
  - Lots of active players in the CI world, gitlab seem to be ahead of the pack, have *very* flexible offering, easy to use
  - That said, if you prefer another option, give it a try!

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 today is about)
  - Distributed builds, can use many platforms
    - Laptop/desktop, Cori/Edison/Genepool, 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 multiple runners per repo with different compilers, runtimes, OS…
  - Can run anywhere: laptop, NERSC machines, cloud

Gitlab server

- **Two editions, three options**
  - 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
    - Cannot mirror external private repositories (update: see appendix)
    - Mirroring external public repositories has 0-1 hours latency
  - Full comparison at https://about.gitlab.com/products/
- **Which option works best for us?**
  - Not clear, nor do we need to choose only one
  - Come and discuss your needs at office hours

Gitlab runner

- **Can run on any platform**
  - Laptop, Cori/Edison/Genepool/Denovo, AWS/GCP/SPIN
  - Configure runners per project
    - Can share runners between projects, or be project-specific
    - *** Gitlab.com provides shared runners, all ready to use!**
  - Specify runners capabilities with tags
    - E.g. gcc/python/perl version, system capabilities (RAM, cores)
  - 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/use
  - Each runner can run a custom workflow
    - E.g. 'build' on Cori, 'build/test/deploy' on Genepool
    - Infinitely configurable, per project
    - Workflow conveniently specified in config file in the project repository

Gitlab and Docker

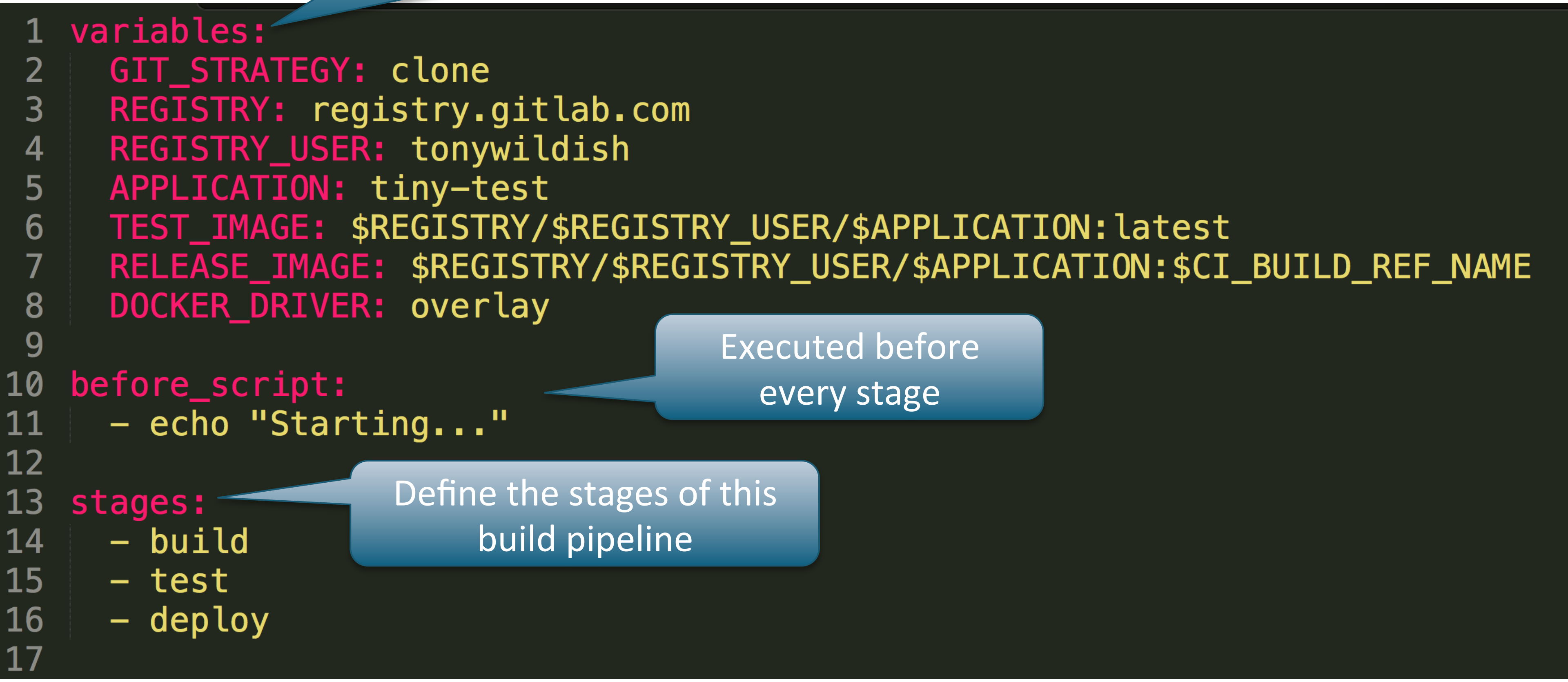
- **Many possible combinations…**
  - Q: Can I do X with Docker and Gitlab? A: Yes, for all X!
- **Run Gitlab Runner in a Docker container**
  - Avoids local installation
- **Pull/run Docker containers to *execute* your CI job**
  - Get exactly the build environment you want
  - *** Use different docker 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**
  - **Y**et **A**nother **M**arkup **L**anguage. Very human-friendly
  - **.gitlab-ci.yml**, in the top directory of your git repository
  - Describes **pipelines** which consist of **stages**
  - Each **stage** has a specific function: **build, test, deploy**…
  - Each **stage** can have its own **tags** (required environment)
  - Each **stage** can produce **artifacts**/re-use from other stages
  - Stages can run in parallel
  - Check/debug your YAML file at https://gitlab.com/ci/lint

- **Similar to makefiles in some ways**
  - Specify dependencies & actions, not explicitly coding workflows

Define environment variables for use in the build



```
variables:
  GIT_STRATEGY: clone
  REGISTRY: registry.gitlab.com
  REGISTRY_USER: tonywildish
  APPLICATION: tiny-test
  TEST_IMAGE: $REGISTRY/$REGISTRY_USER/$APPLICATION:latest
  RELEASE_IMAGE: $REGISTRY/$REGISTRY_USER/$APPLICATION:$CI_BUILD_REF_NAME
  DOCKER_DRIVER: overlay

before_script:
  - echo "Starting..."

stages:
  - build
  - test
  - deploy

```

Executed before every stage

Define the stages of this build pipeline

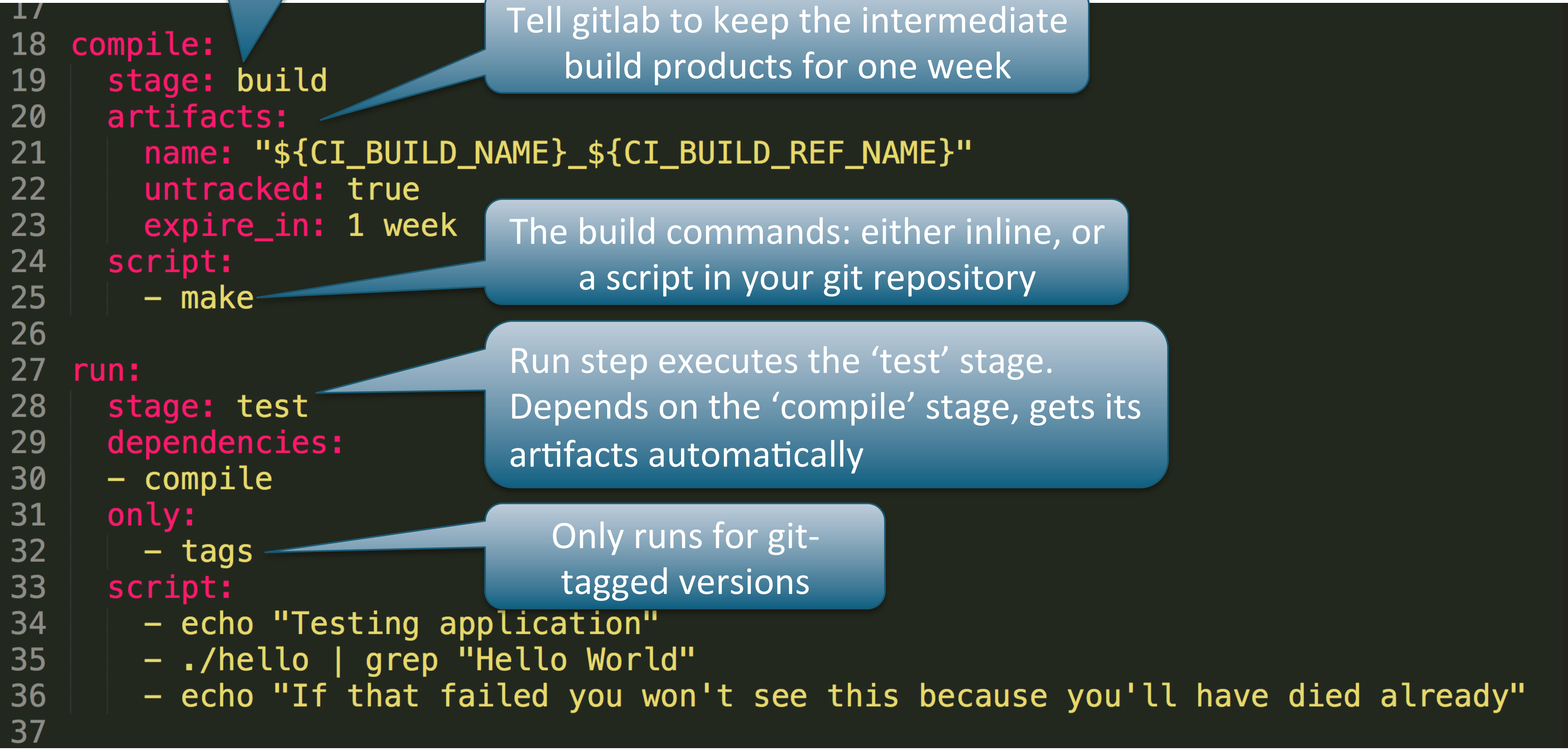
Compile step, executes the ‘build’ stage

Tell gitlab to keep the intermediate build products for one week

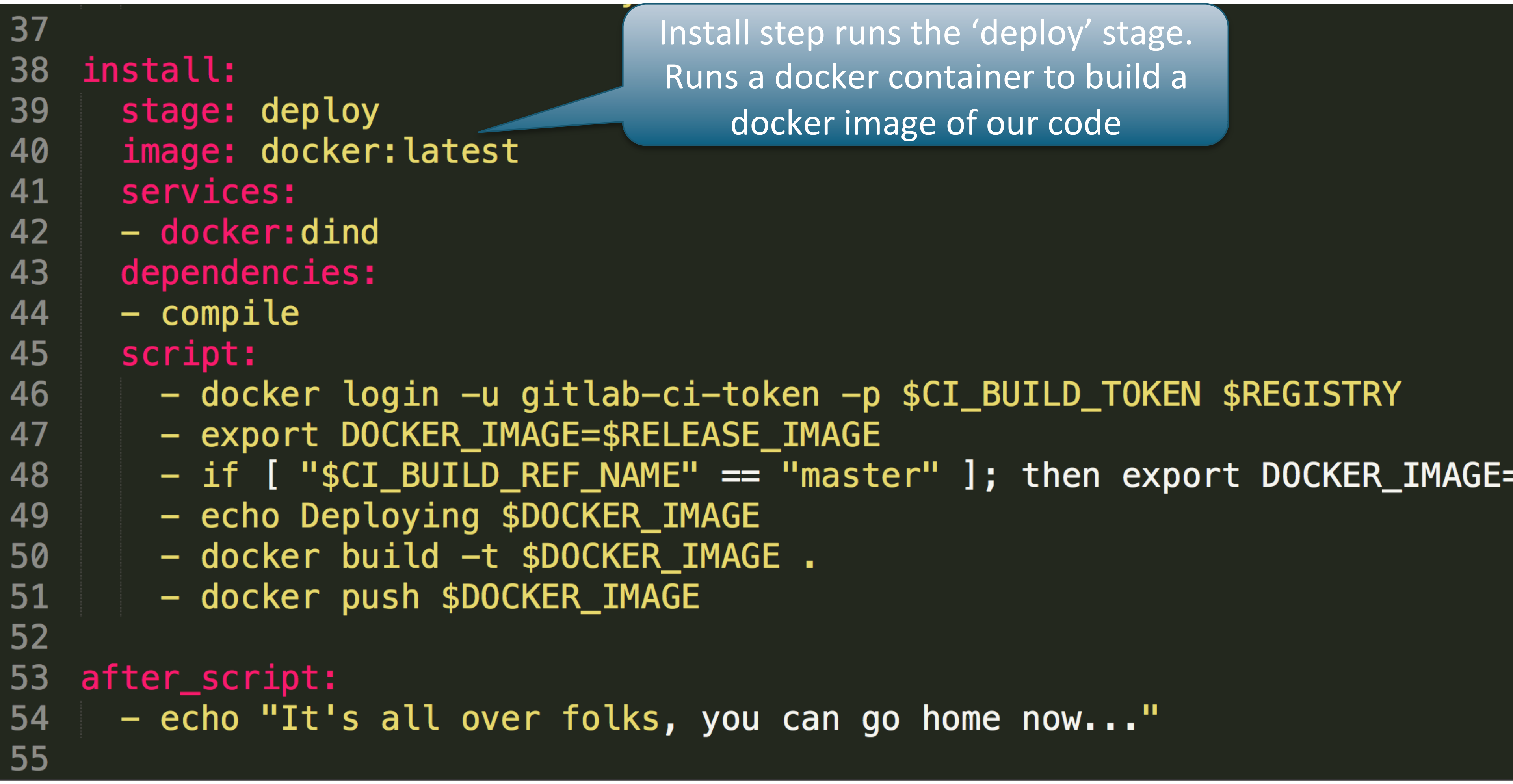
The build commands: either inline, or a script in your git repository

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

Only runs for git-tagged versions

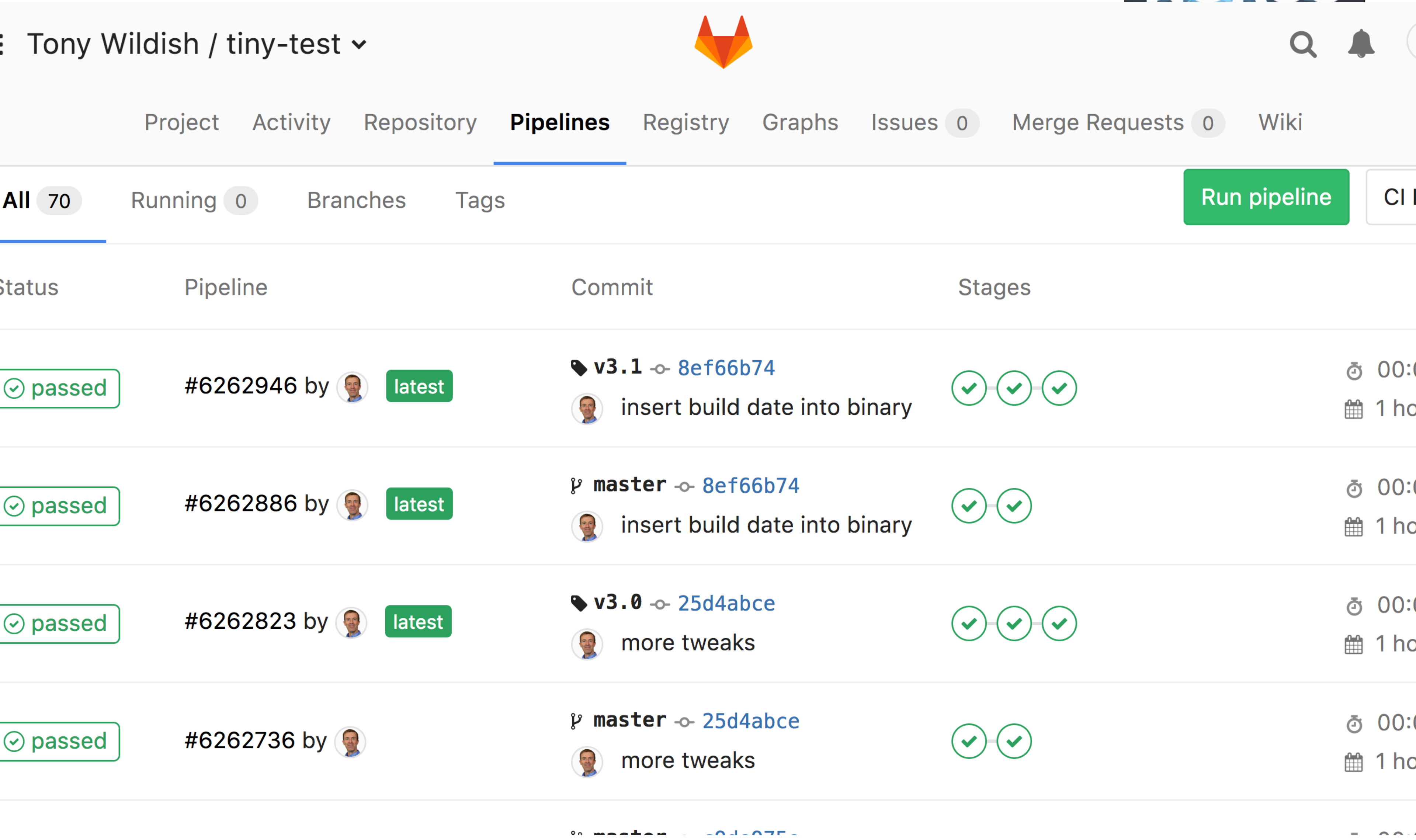


```
compile:
  stage: build
  artifacts:
    name: "${CI_BUILD_NAME}_${CI_BUILD_REF_NAME}"
    untracked: true
    expire_in: 1 week
  script:
    - make

run:
  stage: test
  dependencies:
  - compile
  only:
    - tags
  script:
    - echo "Testing application"
    - ./hello | grep "Hello World"
    - echo "If that failed you won't see this because you'll have died already"

```

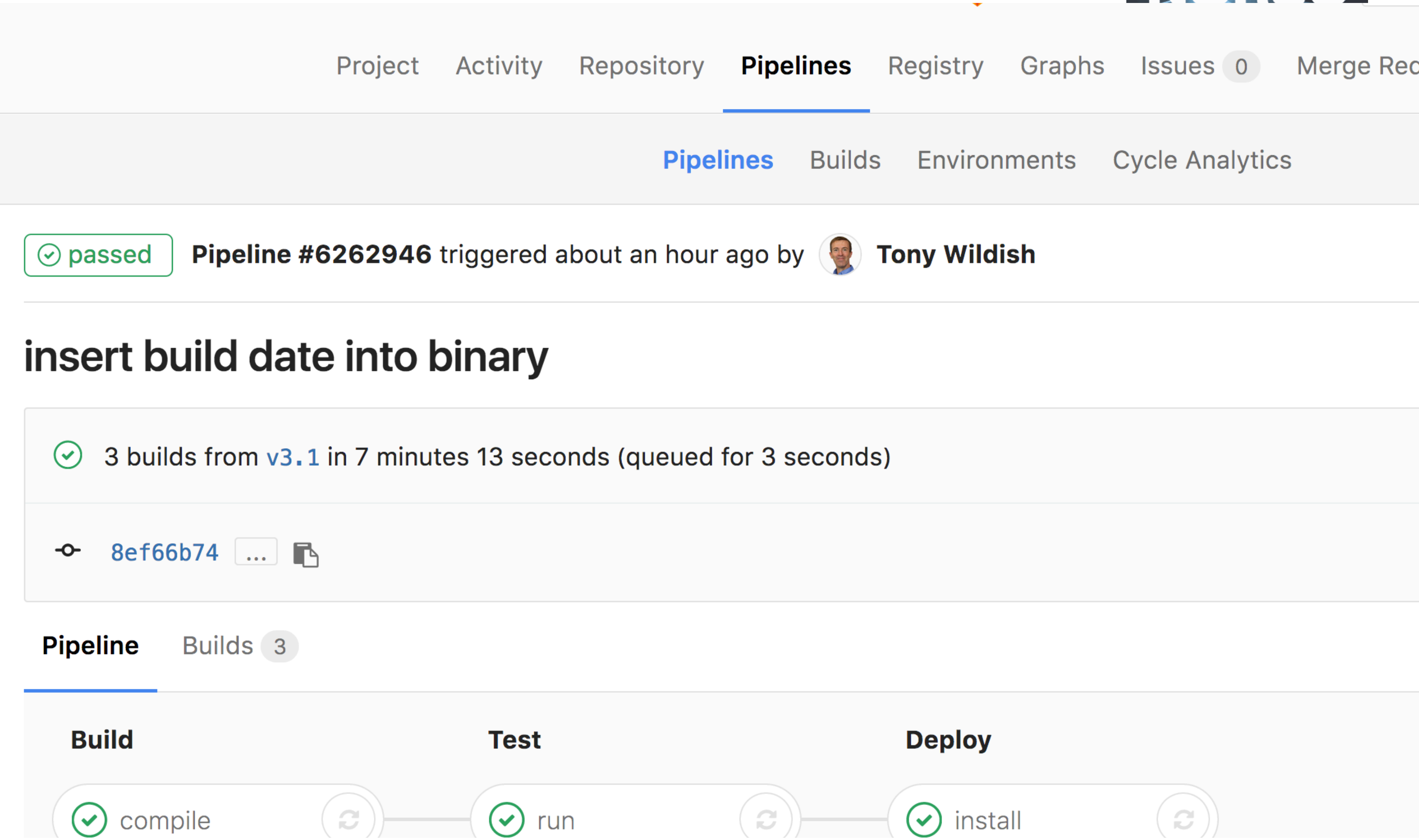


Install step runs the 'deploy' stage. Runs a docker container to build a docker image of our code

```
install:
  stage: deploy
  image: docker:latest
  services:
  - docker:dind
  dependencies:
  - compile
  script:
    - docker login -u gitlab-ci-token -p $CI_BUILD_TOKEN $REGISTRY
    - export DOCKER_IMAGE=$RELEASE_IMAGE
    - if [ "$CI_BUILD_REF_NAME" == "master" ]; then export DOCKER_IMAGE=
    - echo Deploying $DOCKER_IMAGE
    - docker build -t $DOCKER_IMAGE .
    - docker push $DOCKER_IMAGE

after_script:
  - echo "It's all over folks, you can go home now..."
```


Tony Wildish / tiny-test

Status	Pipeline	Commit	Stages	
✓ passed	#6262946 by latest	v3.1 8ef66b74 insert build date into binary	✓ ✓ ✓	00: 1 ho
✓ passed	#6262886 by latest	master 8ef66b74 insert build date into binary	✓ ✓	00: 1 ho
✓ passed	#6262823 by latest	v3.0 25d4abce more tweaks	✓ ✓ ✓	00: 1 ho
✓ passed	#6262736 by	master 25d4abce more tweaks	✓ ✓	00: 1 ho



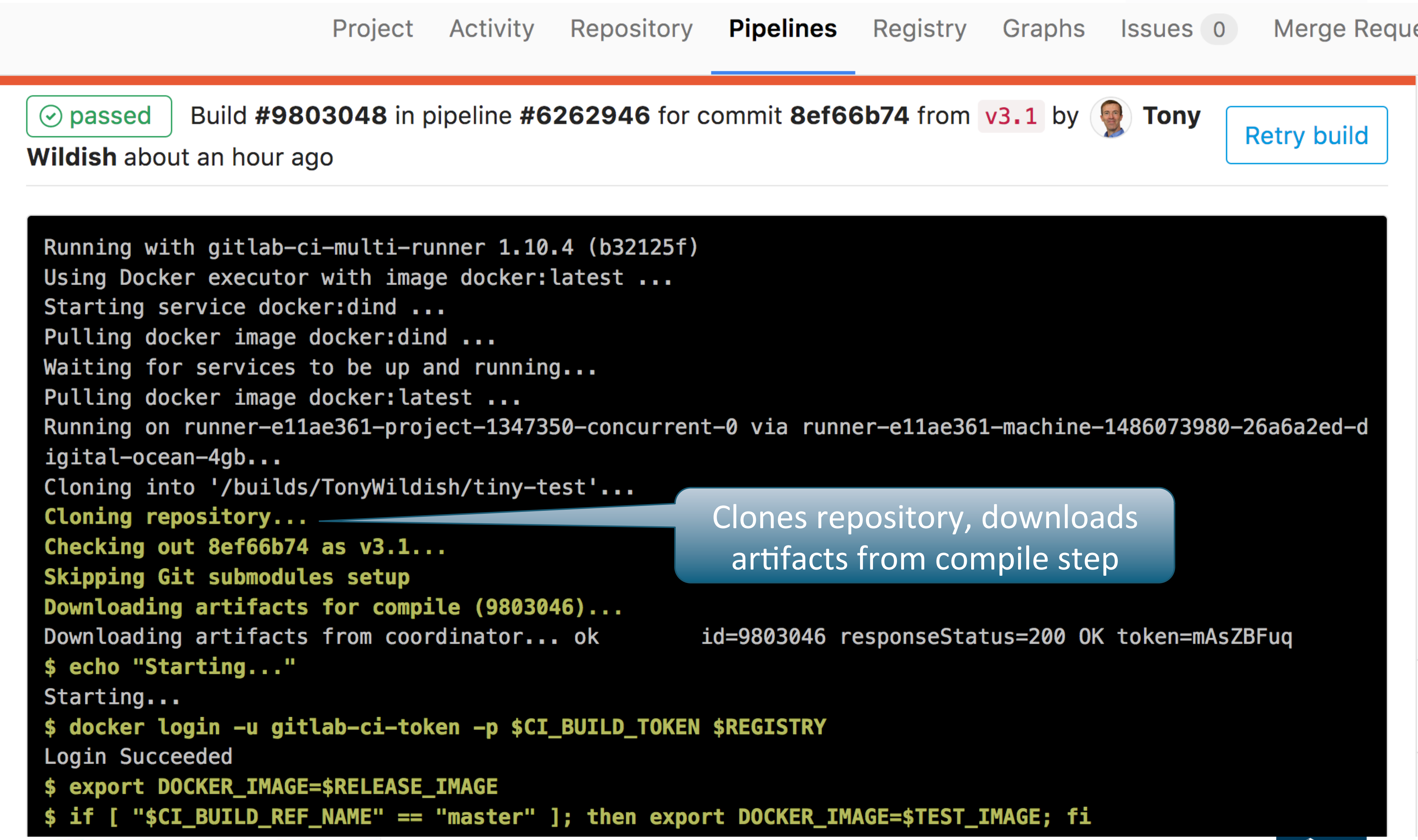
passed **Pipeline #6262946** triggered about an hour ago by **Tony Wildish**

insert build date into binary

3 builds from v3.1 in 7 minutes 13 seconds (queued for 3 seconds)

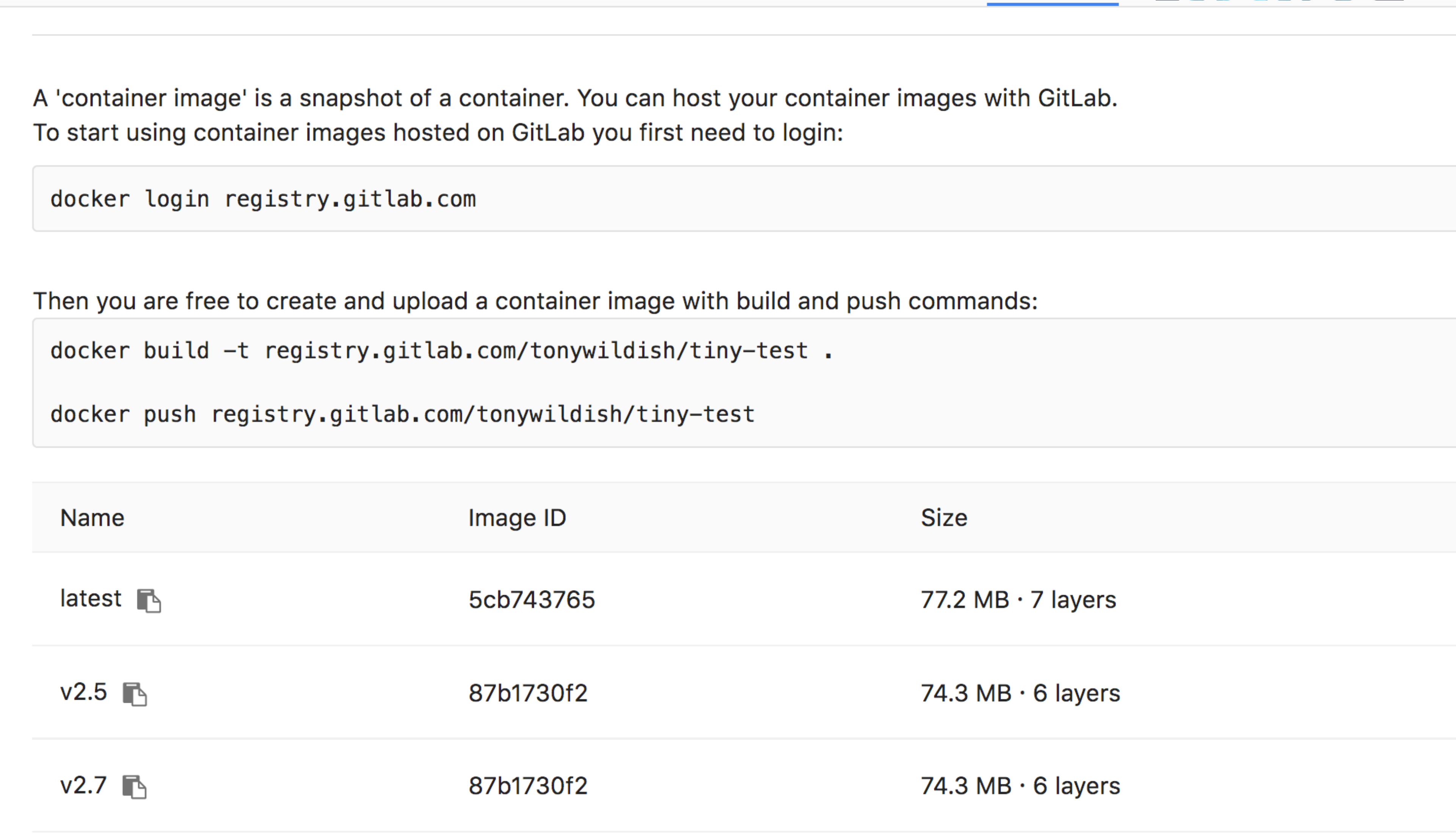
8ef66b74 ...

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



Build **#9803048** in pipeline **#6262946** for commit **8ef66b74** from v3.1 by **Tony Wildish** about an hour ago

```
Running with gitlab-ci-multi-runner 1.10.4 (b32125f)
Using Docker executor with image docker:latest ...
Starting service docker:dind ...
Pulling docker image docker:dind ...
Waiting for services to be up and running...
Pulling docker image docker:latest ...
Running on runner-e11ae361-project-1347350-concurrent-0 via runner-e11ae361-machine-1486073980-26a6a2ed-d
igital-ocean-4gb...
Cloning into '/builds/TonyWildish/tiny-test'...
Cloning repository...
Checking out 8ef66b74 as v3.1...
Skipping Git submodules setup
Downloading artifacts for compile (9803046)...
Downloading artifacts from coordinator... ok        id=9803046 responseStatus=200 OK token=mAsZBFuq
$ echo "Starting..."
Starting...
$ docker login -u gitlab-ci-token -p $CI_BUILD_TOKEN $REGISTRY
Login Succeeded
$ export DOCKER_IMAGE=$RELEASE_IMAGE
$ if [ "$CI_BUILD_REF_NAME" == "master" ]; then export DOCKER_IMAGE=$TEST_IMAGE; fi
```



A 'container image' is a snapshot of a container. You can host your container images with GitLab.
To start using container images hosted on GitLab you first need to login:

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

Then you are free to create and upload a container image with build and push commands:

```
docker build -t registry.gitlab.com/tonywildish/tiny-test .

docker push registry.gitlab.com/tonywildish/tiny-test
```

Name	Image ID	Size
latest	5cb743765	77.2 MB · 7 layers
v2.5	87b1730f2	74.3 MB · 6 layers
v2.7	87b1730f2	74.3 MB · 6 layers

Hands-on, exercise 1, part 1

- **Go to Gitlab.com, create an account**
- **Upload your SSH public key (*not* your private key!)**
  - Avatar top-right -> pull-down menu -> Settings -> SSH-keys
- **Create a new project**
  - ‘Hamburger’ icon top-left -> Projects -> New Project (top-right)
  - Follow the steps to set it up from scratch
- **Enable the Container Registry for this project**
  - Gear icon top-right -> Edit Project -> scroll down
- **Go to http://bit.ly/2kAuhFo, download tiny-test.tar**
  - Untar it, move all the files into your project (including ‘.git*’)
  - Edit .gitlab-ci.yml, change REGISTRY_USER and APPLICATION to your username and your project name, **all in lowercase**
- **Add/commit/push this code to your project**
  - git add . ; git commit –m ‘blah...’ ; git push
- **Go to your project ‘Pipelines’ page**
  - Watch the progress of your build!

Hands-on, exercise 1, part 2

- **Go to your project ‘Registry’ page**
  - You should see a Docker image listed, with version ‘latest’
- **Log in to the gitlab docker registry**
  - From a terminal window, type:
    - **docker login registry.gitlab.com**
  - Give your Gitlab username/password when prompted
- **Run your docker image!**
  - **docker run registry.gitlab.com/$USER/$PROJECT**
    - **$USER** is your gitlab username
    - **$PROJECT** is the name of your project
    - You should see the ‘**Hello World**’ message on your terminal!

Hands-on, exercise 2

- **Now add a git tag:**
  - **git tag v1.0**
  - **git push –tags**
    - **That's two '-'s there, dash-dash-tags**
- **Watch the Pipelines page**
  - You should see a three-step build, with the 'test' stage
- **Check the Repository page**
  - You should see a **v1.0** docker image there too
  - Check you can run it with:
    - **docker run registry.gitlab.com/$USER/$PROJECT**:v1.0

Hands-on – offline, for bonus points…

- **Ex.3. Change the pipeline to do the following:**
  - For tagged code, do the **test** stage after the **deploy**, not before
    - Hint:
      - Where do you specify the order of **stages**?
      - Where do you specify the **dependencies**?

- **Ex.4. Then add another test to run the Docker image, not the live executable**
  - Hint:
    - Pick a unique name for the test, specify it runs the **test** stage
    - See how the Docker image is built, copy/modify to run it instead

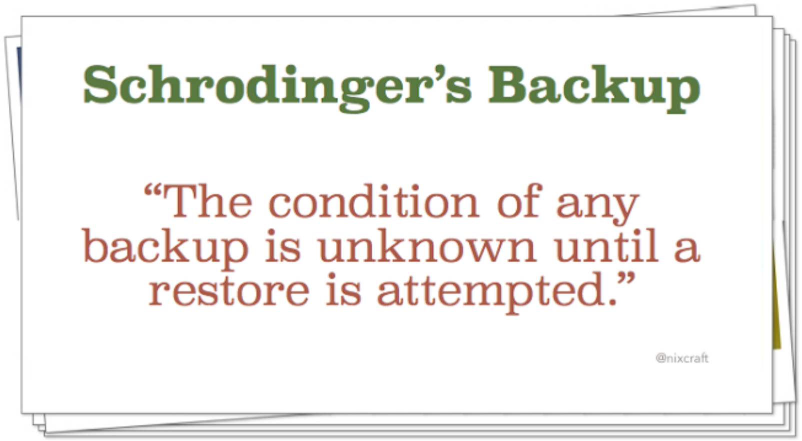
- **Ex.5. Install a gitlab-runner locally on your machine**
  - Make it project-specific, not shared
  - See '**Creating and Registering a Runner**' in the docs (https://docs.gitlab.com/ee/ci/runners/README.html)

Further steps…

- **Install/run runners on Cori/Genepool?**
  - Can't build docker images there, docker not supported
  - Will have access to the full NERSC build environment
  - Gotcha w.r.t. installation, come talk to us first
- **Install runners on SPIN (NERSC internal cloud)**
  - Under development, watch this space…
  - Should be able to build docker images from builds on Cori
- **Install runners on your laptop/desktop?**
  - Good way to get experience/practice until we have runners supported on SPIN

That incident…

- **On Feb 1st, Gitlab accidentally 'rm –rf'ed in the wrong directory**
  - They lost 6 hours of data
  - 5 backup methods all failed
  - Laugh only if you've never screwed up yourself ☺



Schrodinger's Backup

"The condition of any backup is unknown until a restore is attempted."

@nixcraft

- **What was lost?**
  - Issues, merge requests, anything done through the web
  - Any code commits from repositories which were then removed from disk during that time-window
    - If you still have your repo on disk, 'git push' and nothing is lost!

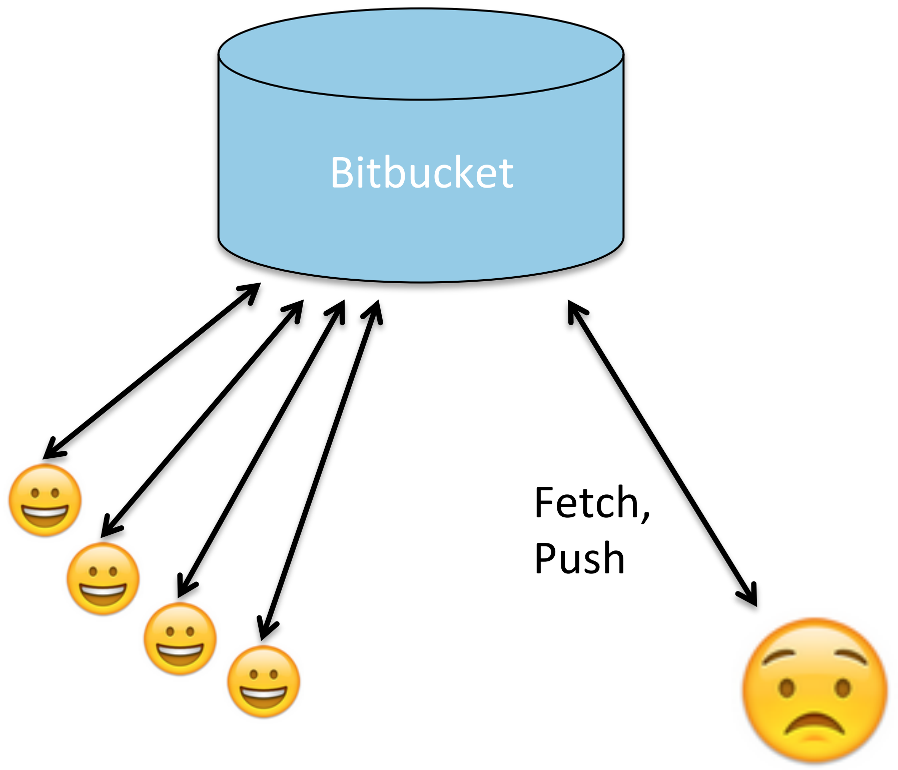
What *could* you do to be even safer?

- **Dual-remote git repositories**
  - Store your code in 2 or more of gitlab, github, bitbucket...
- **How?**
  - Create a repository, R1, on one service, populate as usual
  - Create a second repository, R2, somewhere else, leave it empty
  - Clone R1 to your local disk
  - Set R2 as a second remote push destination
  - Then hack, commit, push, push R2; update both remotes!
- **Gotchas?**
  - R1 and R2 know nothing about each other
    - If they're both modified independently, you can get into trouble
  - However, fine if R2 is *only* used for specific purposes, like CI
  - ...and it's a very good way to get started with gitlab!

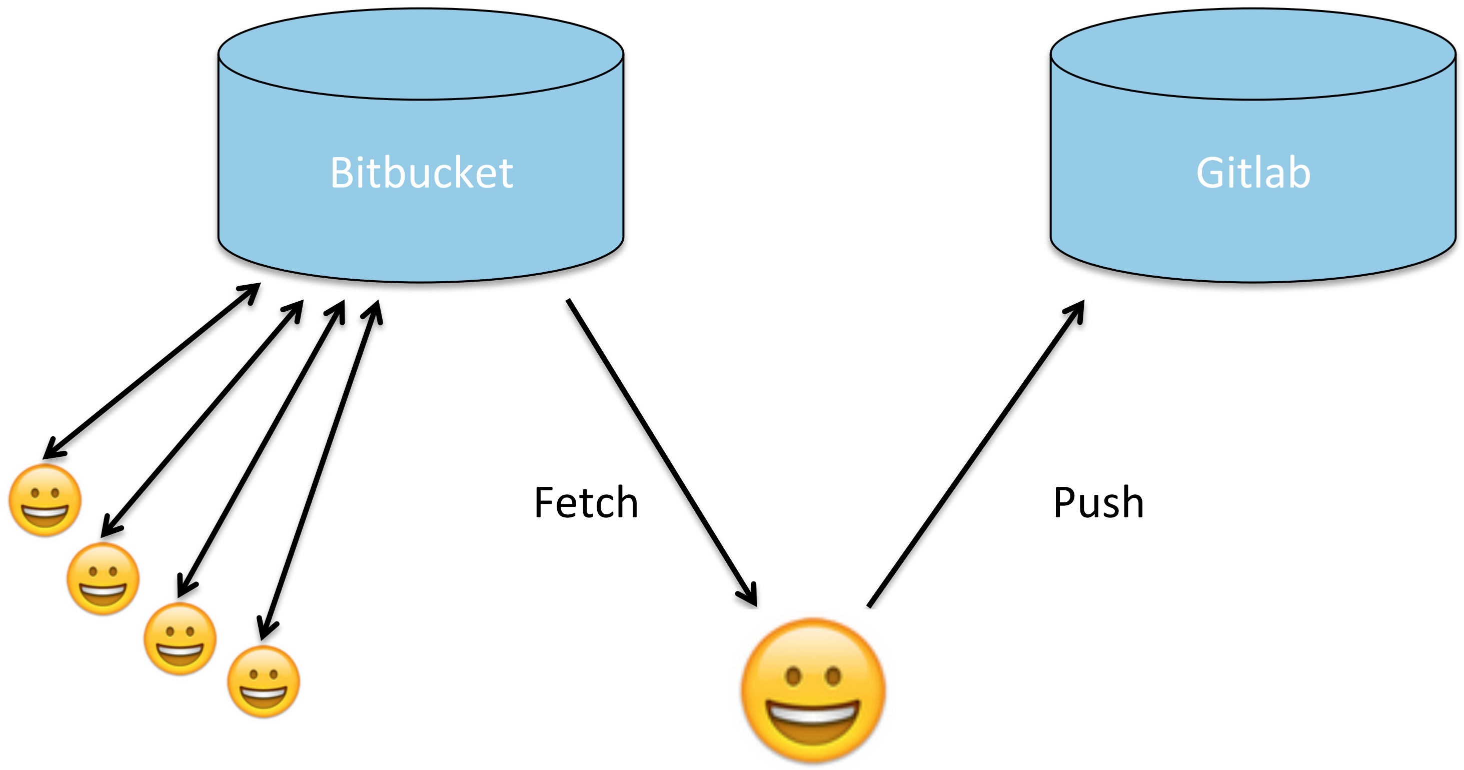
Using dual git-remotes for CI

- **Problem: you want to use Gitlab CI, but…**
  - You have code in a private repository in Bitbucket
  - Gitlab.com can't easily mirror external private repositories
    - See appendix to this presentation
  - You don't want to move your repository to Gitlab – (yet!)
- **Solution: use dual git remotes**
  - Create an empty Gitlab repository
  - Clone your Bitbucket repository somewhere
  - Configure your clone to push to Gitlab
    - But to pull only from Bitbucket!
  - Continue working *exactly* as before, even on shared projects
    - Can pull changes committed to Bitbucket by other people
    - Then push them, to send them to Gitlab
- **This is advanced git, amaze your friends ☺**

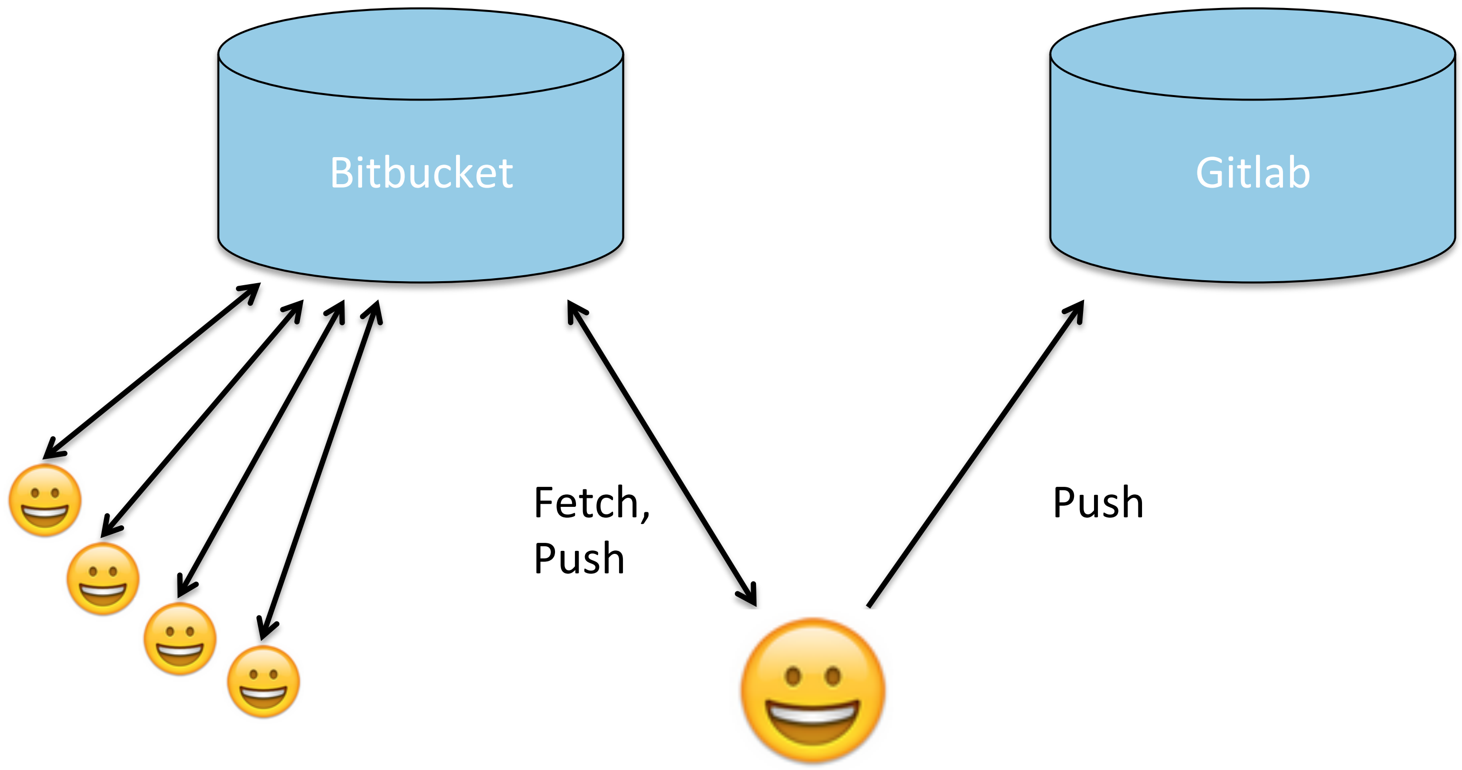
Using dual git-remotes for CI



Using dual git-remotes for CI



Using dual git-remotes for CI



Hands-on, exercise 6

- **Go to https://bitbucket.org/TWildish/gitlab-ci-demo**
- **Follow the instructions in the README.md**
  - Fork the repository, so you have your own copy in bitbucket,
  - Clone it to your local disk
  - Create an empty repository in gitlab
  - Set the **push** destination of your clone to point to gitlab
  - **Git push**, and watch the code build!

  - In another directory, clone the bitbucket repository again, as normal
  - Modify it in some way (add a file) and commit those changes
  - Go back to your 'bitbucket+gitlab' clone
  - **Pull** the changes, and **push** them to gitlab!

- **Not the only way to do it**
  - Can have multiple **push** destinations in the same clone
  - Which you do is a matter of personal choice, no clear advantage
  - More info on multiple remotes: 'Pro Git', https://git-scm.com/book/en/v2, free on the web. Or ask us

Best practices, gotchas…

- **Be careful with environment variables**
  - Gitlab sets some secret environment variables (API keys etc)
  - If you echo them to your logfiles, they will be visible on the web
  - The only way to delete old logfiles from gitlab.com is to delete the build!
- **Check your YAML configuration file for errors**
  - Use ‘CI Lint’, at https://gitlab.com/ci/lint, 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 runners to specify requirements, avoid complex runtime scripts
    - E.g. runner with tag ‘genepool’, use that tag in YAML config file ☺
    - Scripts with “if $NERSC_HOST==‘genepool’” ☹

National Energy Research Scientific Computing Center

Mirroring private bitbucket repositories

- **It *is* possible to mirror private bitbucket or github repositories, but there are risks**
  - You give your bitbucket username & password in the URL of the repository you want to mirror
  - This is visible to anyone with the rights to manage your project
  - Anyone who gets access can modify or delete your private repositories
- **Here's the recipe:**
  - Create a new account on bitbucket, call it 'YourNameRO'
  - Grant it Read Only access to your private bitbucket repositories
  - Give the username & password of *that* account to gitlab, instead of your real account
  - **Only** ever use the YourNameRO account for **read-only access**
    - Never create repositories or forks, it's just a gateway account
  - Now if your gitlab account is compromised you leak far less access
    - Someone can read your private bitbucket code, but not change it
    - Change your YourNameRO account password and you're safe again!

Mirroring private bitbucket repositories 2

- **Bitbucket (and other services) require a unique email address for account registration**
- **How do you register for a new account without an alias for your lbl.gov email address?**
  - Lbl.gov is managed by Google, it's Gmail under the hood
  - Any Gmail address can have arbitrary 'extensions' to the username as aliases for the primary account
    - Just add '+' followed by more text
  - E.g., these are all equivalent to your primary address
    - user@lbl.gov
    - user+bitbucket_ro@lbl.gov
    - user+other_service@lbl.gov
  - You don't need to register these email aliases anywhere, you can just use them. Go ahead, try it!