A gentle Introduction to docker and gitlab-CI

Erwan Jahier

Verimag Technical Seminar

4 october 2018
Outline

1 Docker
   - What?
   - How?
   - When?

2 Gitlab CI/CD
   - What?
   - How?
   - When?
Plan

1. Docker
   - What?
   - How?
   - When?

2. Gitlab CI/CD
   - What?
   - How?
   - When?
Docker: what?

- A platform to develop, deploy, and run applications with **containers**
- A container is a **lightweight** Virtual machine (VM), that shares the **kernel/OS** of the host machine
  - fast launching
  - fast executables
  - low memory consumption
  - smaller images
  - **but** are less versatile

```bash
docker run -it java
  javac -version
  cat /etc/os-release
  uname -a
docker run java javac -version
```
Docker Layers

source: http://neokobo.blogspot.com/2017/03/docker-container.html
Docker Layers

source: https://www.slideshare.net/mattfarina/a-dive-into-containers-and-docker
Docker versus VM

Containers vs. VMs

Containers are isolated, but share OS and, where appropriate, bins/libraries

source: https://www.fotozik.fr/docker
CLI versus GUI

- **docker** is CLI, while VMs are GUIs
- If you want to manage VMs via CLI
  - **docker-machine**
  - **Vagrant**: a CLI to uniformly manage Virtual machines such as:
    - Docker
    - Virtual box
    - vmware
    - and others
Docker Versatility

- Linux docker images can run under
  - Linux boxes
  - Windows 10 (via Virtual Box)
  - Mac (via Virtual Box or HyperKit)

- What about mac-os docker images?
  - ok under mac
  - possible also under pc/linux, but forbidden!

cf https://docs.docker.com/install/
docker: how?

• find an image

```bash
docker search gabu
```

• use an image

```bash
docker run -it gabuzomeu
docker run -v "$PWD":/current_dir -w /current_dir -v /tmp/.X11-unix:/tmp/.X11-unix -e DISPLAY=$DISPLAY -it gabuzomeu
```
Docker: how?

How to make your own image?

Dockerfile:

```
FROM ocaml/opam:ubuntu-16.04_ocaml-4.06.0
MAINTAINER Erwan Jahier erwan.jahier@univ-grenoble-alpes.fr

### install coq and coqide
RUN sudo apt-get install -qq -yy libexpat1-dev libgtk2.0-dev libgtksourceview2.0-dev m4 pkg-config
RUN opam update && opam install -y coqide && opam user-setup install

# set a nicer prompt
RUN echo "export PS1=[some nifty prompt] " >> ~/.bashrc

RUN sudo mkdir /usr/local/lib/jvm/java5 &
    wget http://download.oracle.com/otn/java/jdk/1.5.0_22/java5.tar.gz
    sudo tar xf java5.tar.gz -C /usr/local/lib/jvm/java5 &
    sudo update-alternatives --install "/usr/local/bin/java" "java" "/usr/local/lib/jvm/java5/jdk1.5.0/bin/java" 1 &
    sudo update-alternatives --install "/usr/local/bin/javac" "javac" "/usr/local/lib/jvm/java5/jdk1.5.0/bin/javac" 1 &
    sudo update-alternatives --install "/usr/local/bin/javaws" "javaws" "/usr/local/lib/jvm/java5/jdk1.5.0/bin/javaws" 1 &
    sudo chmod a+x /usr/local/bin/java &
    sudo chmod a+x /usr/local/bin/javac &
    sudo chown -R root:root /usr/local/lib/jvm/java5/jdk1.5.0

ENV PATH=$PATH:/usr/local/bin
```
Docker images: Build, Distribute, Use

- **build and use you own image**

  ```bash
  docker build -t my-image-with-little-onions - < Dockerfile
  docker run -it my-image-with-little-onions
  ```

- **Upload an image on the cloud**

  ```bash
  docker tag my-image-with-little-onions myrepo/my-image-with-little-onions
  docker push myrepo/my-image-with-little-onions
  ```

- **Use your image anywhere**

  ```bash
  docker run -it myrepo/my-image-with-little-onions
  ```
Docker Trusted Registry (a.k.a. DTR)

If you do not trust the cloud, you can set-up your own registry

- a registry is a collection of repository
- a repository is a collection of images
- the docker CLI uses Docker’s public registry by default

```bash
docker pull myregistry.local:5000/myrepo/my-image-with-little-onions
```
And much more

config container image network node plugin secret service stack swarm system trust volume attach build commit cp create diff events exec export history images import info inspect kill load login logout logs pause port ps pull push rename restart rm rmi run save search start stats stop tag top unpause update version wait
Security

```
sudo docker run -it some_repo/some_image_from_an_unkown_source
docker run -v "$HOME:/some_dir -w /some_dir
docker run -v "$PWD:/some_dir -w /some_dir
```
Some Use Cases

- Install/try new software without polluting its `.bashrc`
- Reproduce an experiment (reproducible research)
- Provide a working/uniform tool environment to students
- Distribute software
- Test (on various architectures)
- CI/CD
Some Alternatives

- Linux Containerization
  - runc
  - rkt
  - LXC, LXD
  - Linux-VServer
  - OpenVZ

- nanobox (a wrapper around docker)

- Vagrant

- nix (declarative package manager)
  - aims at installing packages in isolation
  - uses less disk space
Plan

1. Docker
   - What?
   - How?
   - When?

2. Gitlab CI/CD
   - What?
   - How?
   - When?
Continuous Integration (CI)

1 Source de l’image: https://dotnetvibes.com/2018/04/10/continuous-delivery-is-not-continuous-deployment/
Continuous Integration (CI)

*CI is the practice of merging all developer working copies to a shared mainline several times a day*

- Requires
  - a version manager
  - automated builds
  - automated testing scripts

- Comes from the Extreme Programming community
CI/CD = Continuous Integration/Deployment

RELEASE PIPELINE

- CODE
- BUILD
- INTEGRATE
- TEST
- RELEASE
- DEPLOY

Continuous Integration
Continuous Delivery
Continuous Deployment
Gitlab CI/CD: how?

Gitlab CI scripts are written in YAML

- Yet Another Markup Language (xml)
- Yaml Ain’t a Markup Language
- Yaml Goals:
  1. serialize/deserialize structured data
  2. Independently of any programming language
  3. Standard
  4. human-eye (and human-finger) friendly (more than xml or even json)
YAML

Allow to write in a natural way:

• Association Tables (:)
• Lists (-)
• Scalars
  ▶ Integers
  ▶ Floats
  ▶ String
  ▶ Boolean
  ▶ Date
  ▶ etc.

Indentation => expressions scope
YAML: examples

- **ex1**: a table associating a string to a list of floats

  ```yaml
  x1:
  - 1.0
  - 2.0
  - 3.0
  x2:
  - 5.0
  - 6.0
  ```

- **ex2**: a table of tables; **spaces define scope!**

  ```yaml
  table1:
  1:
  - 2012-08-06
  - 2013-02-05
  2:
  - 1973-19-06
  table2:
  1:
  - 2018-28-02
  ```
**YAML: Anchors and References**

- Anchors (`&ident`) and references (`*ident`) can avoid code duplication

```yaml
foo: &anchor
  K1: "One"
  K2: "Two"
bar: *anchor
  K3: "Three"
```

- `<<: *ident` let one extend an association table

```yaml
foo: &anchor
  K1: "One"
  K2: "Two"
bar:
  <<: *anchor
    K2: "I Changed"
    K3: "Three"
```

During the merge, the last one wins; K2 is therefore associated to “I Changed”
YAML serialization / deserialization

An example in java

- **Sérialisation**

  ```java
  Yaml.dump(object, new File("dump.yml"));
  ```

- **Désérialisation**

  ```java
  exemple.yaml:
  ```
  ```yaml
  - A char string
  - 2
  - { three: 3.0, four: true }
  ```

  ```java
  System.out.println(Yaml.load(new File("exemple.yaml")));
  ```

  ```java
  [A char string, 2, {tree=3.0, four=true}]`
  ```
The CI work is controlled by a YAML file named `.gitlab-ci.yml` located at the project root.

- `.gitlab-ci.yml`
  - define « jobs »
  - jobs can be executed in sequence (pipeline) or in parallel
  - jobs are executed by « runners »

- « Runners » can execute jobs via
  - a local shell
  - ssh
  - a virtual machine
  - docker

Runners requires some set-up/configuration work
- A shared docker runner is provided by `gitlab/gricad`
- `veri-gitlabrun @ Verimag` (10 Xeon @2.00GHz; 8M RAM)
Gitlab CI jobs in parallel

Here is a `.gitlab-ci.yml` defining 2 jobs in parallel on a docker image running the latest ubuntu:

```
image: ubuntu:latest
job1:
  - echo "you can run any command available on the latest ubuntu" > cmd.sh
  - . cmd.sh

job2:
  - ls
  - pwd
  - apt-get update
  - apt-get install
  - make
```

- each of this job is executed by a (possibly different) "Runner"
- the git clone has been done implicitly
Gitlab CI jobs in sequence (Pipeline)

In order to run jobs in sequence, one need to define “stages”

```
stages:
  - test
  - build
  - deploy

job1:
  stage: build
  script:
    - mvn build

job2:
  stage: test
  script:
    - mvn test

job3:
  stage: deploy
  script:
    - mvn deploy
```

nb: by default a job is scheduled during the test stage
how to avoid Code duplication

1. One can use YAML anchors and references
2. One can extend its own image
3. The keywords `before_script` allows to factorize a sequence of commands

```yaml
image: ubuntu:latest
before_script:
  - apt-get update
  - apt-get install maven openjdk-8-jre
tp1 tests:
  script:
    - cd tp1/
    - mvn test
tp2 tests:
  script:
    - cd tp2/
    - mvn test
```
- By default, CI pipelines are triggered by `git push`.
- CI jobs can be run on some specific branches only.
- External trigger (POST), useful for multi-project CI.
- Scheduled trigger.
- `nb: jobs can be executed locally before doing a « git push »`
  (using `docker` and `gitlab-runner`.)
**TL;DR**

- At every « git push », a script (.gitlab-ci.yml) is executed
- which launch **jobs in sequence and/or in parallel** (pipelines)
  - Build
  - Test
  - Deploy (pdf, binaries, web pages)

- Jobs are executed by « **Runners** » that requires some set-up work
- ∃ a **shared** runner @ **Gricad** + veri-gitlabrun @ **Verimag**
When should we write CI scripts?

- Well, of course, it depends on the kind of projects
  - For software development: always! (at least build and basic tests)
  - What about other version-controlled files such as article?
Even on papers it makes sense!

```
image: aergus/latex

paper:
  script:
    - cd papers; make; make bib; make; make
  artifacts:
    paths:
      - papers/main.pdf

slides:
  script:
    - cd slides; make; make
  artifacts:
    paths:
      - slides/main.pdf
```
Give a public access to your CI artifacts

```yaml
pages:
  stage: deploy
  script:
    - cp papers/main.pdf public/paper.pdf
    - cp slides/main.pdf public/slides.pdf
    - cd public; tree -H > index.html

artifacts:
  paths:
    - public
```

The resulting public link is available via Settings > Pages

nb: does not work for projects in sub-groups
Reproducible Research

- A neglected topic in computer science community
- script the experimentation and the generation of Figures that are included in the article
- cf the Gitlab group verimag/reproducible-research
  - anyone can see the source and the output of the CI jobs
  - anyone can should be able to reproduce the experimentation locally (if it is based on docker runners)
- org-mode is a great tool (used in conjunction of gitlab/docker) to:
  - write articles
  - write slides
  - write web site
  - do literate programming
Conclusion

• We now have a much nicer software forge
• Running basic tests automatically is easy
• Your test are run on a docker runner
• docker is worth having a look at (independently of gitlab-CI)