Continuous Delivery of embedded firmware using Docker and Jenkins

#TorinoTech Night, 2016-04-01

Gianpaolo Macario

https://gmacario.github.io/

The context

Firmware images of recent embedded devices are produced through a complex and time-consuming process

A few examples:

- In-Vehicle Infotainment
- Mobile Phone
- Home Gateway
- Wireless Router/Access Point
- etc

Example: GENIVI Demo Platform

```
$ sloccount gdp_ivi9_beta/workspace

Totals grouped by language (dominant language first):

ansic:      46260693 (69.83%)
cpp:         9910207 (14.96%)
asm:         2006557 (3.03%)
sh:          1575972 (2.38%)
perl:        1276566 (1.93%)
python:      1163866 (1.76%)
xml:         1054193 (1.59%)

...

Total Physical Source Lines of Code (SLOC) = 66,247,653
```

Example: Android 5.1.1

```
$ sloccount build_android_udooneo/workspace

Totals grouped by language (dominant language first):
ansic:      28909496 (48.66%)
cpp:        14820059 (24.95%)
xml:         6894345 (11.61%)
java:        5651249 (9.51%)
asm:         1105027 (1.86%)
python:      1061272 (1.79%)
sh:           434135 (0.73%)
perl:         197439 (0.33%)

...

Total Physical Source Lines of Code (SLOC) = 59,405,160
```

Build Host Configuration

Varies greatly between projects

Some examples:

- Android 1.5-2.2.x: Ubuntu 10.04, Oracle JDK 5
- Android 2.3.x-4.4.x: Ubuntu 12.04, Oracle JDK 6
- Android 5.x: Ubuntu 12.04, Oracle JDK 7
- Android 6.0: Ubuntu 14.04, Oracle JDK 7
- Android N: Ubuntu 14.04, OpenJDK 8
- Commercial IVI product: Ubuntu 10.04, CodeSourcery GCC 2014.05
- CommonAPI C: Fedora 23, systemd, libsystem-devel
- GENIVI Demo Platform: Ubuntu 14.04, gcc 4.8, make 4.22, python 2.7

Build host configuration (cont.)

- **Depends upon each embedded project**
  - Each developer may need to work on several projects with incompatible host requirements
  - The configuration of the host may affect the generated firmware image (host cross contamination)
  - Developers do not always follow instructions...
- **Should be maintained along the project lifetime**
  - We cannot assume that developers will not change the software configuration on their machines for years!

How can we ensure that each build is reproducible?

One solution: Virtual Machines

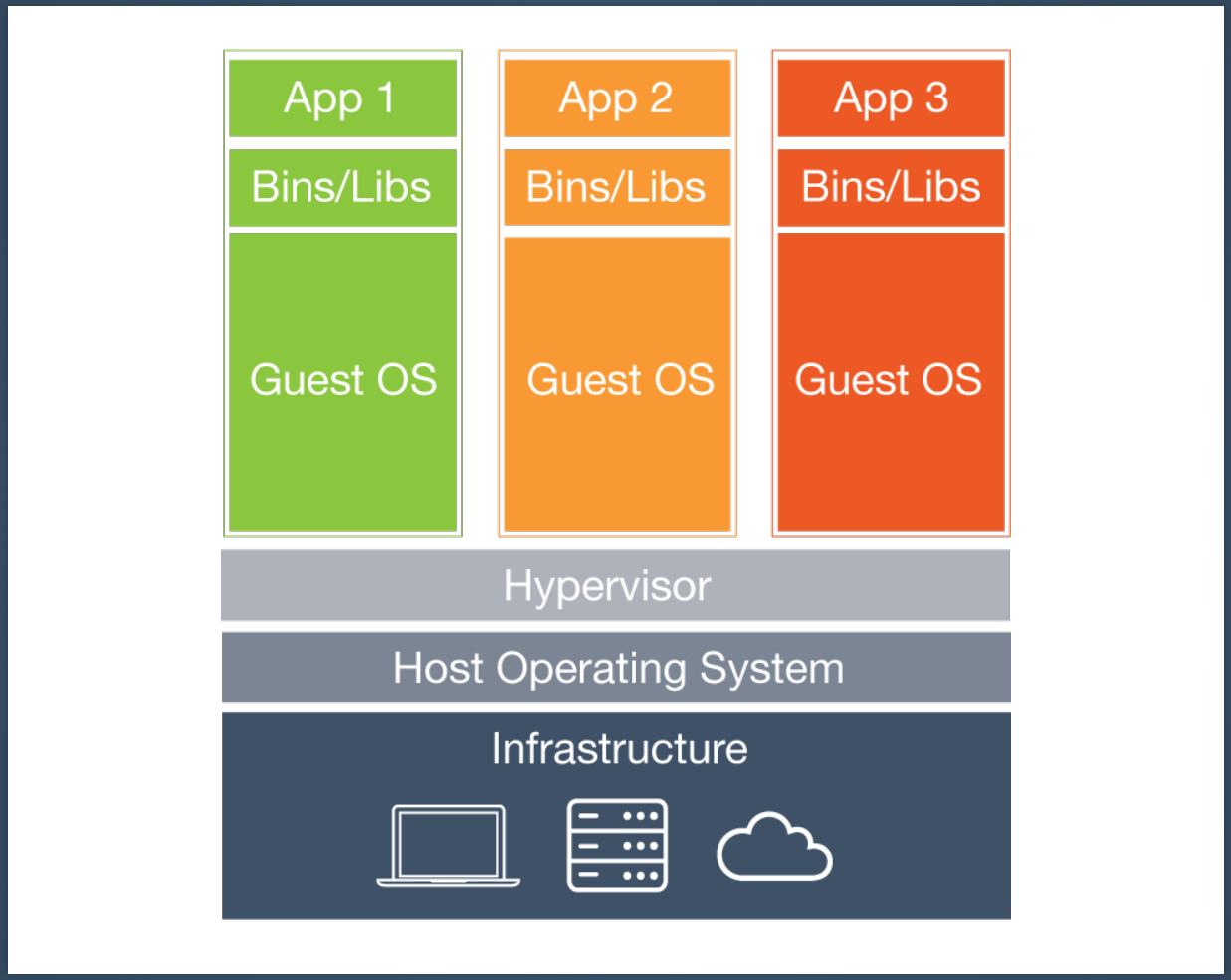


Image credit: https://www.docker.com

A better solution: Linux Containers

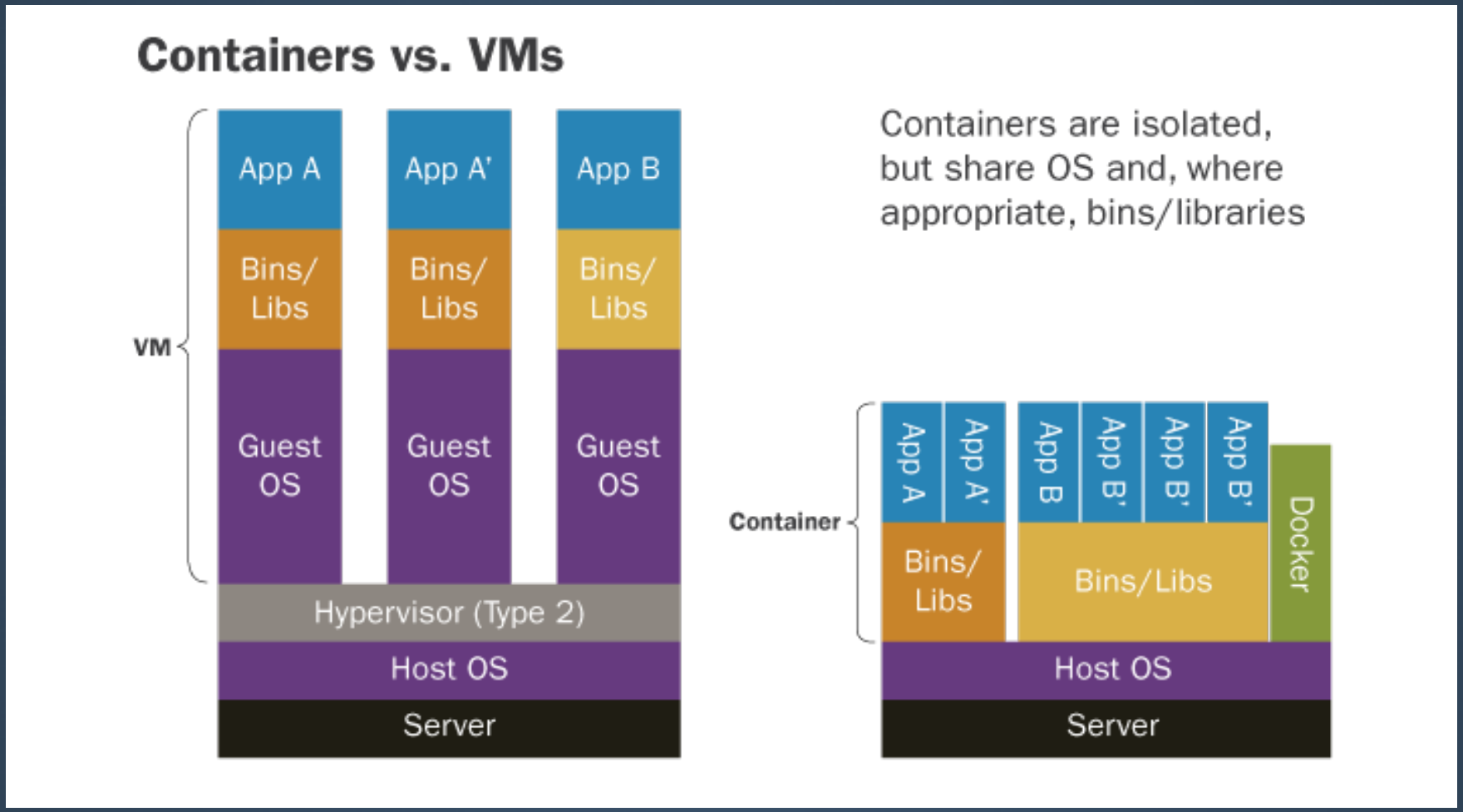
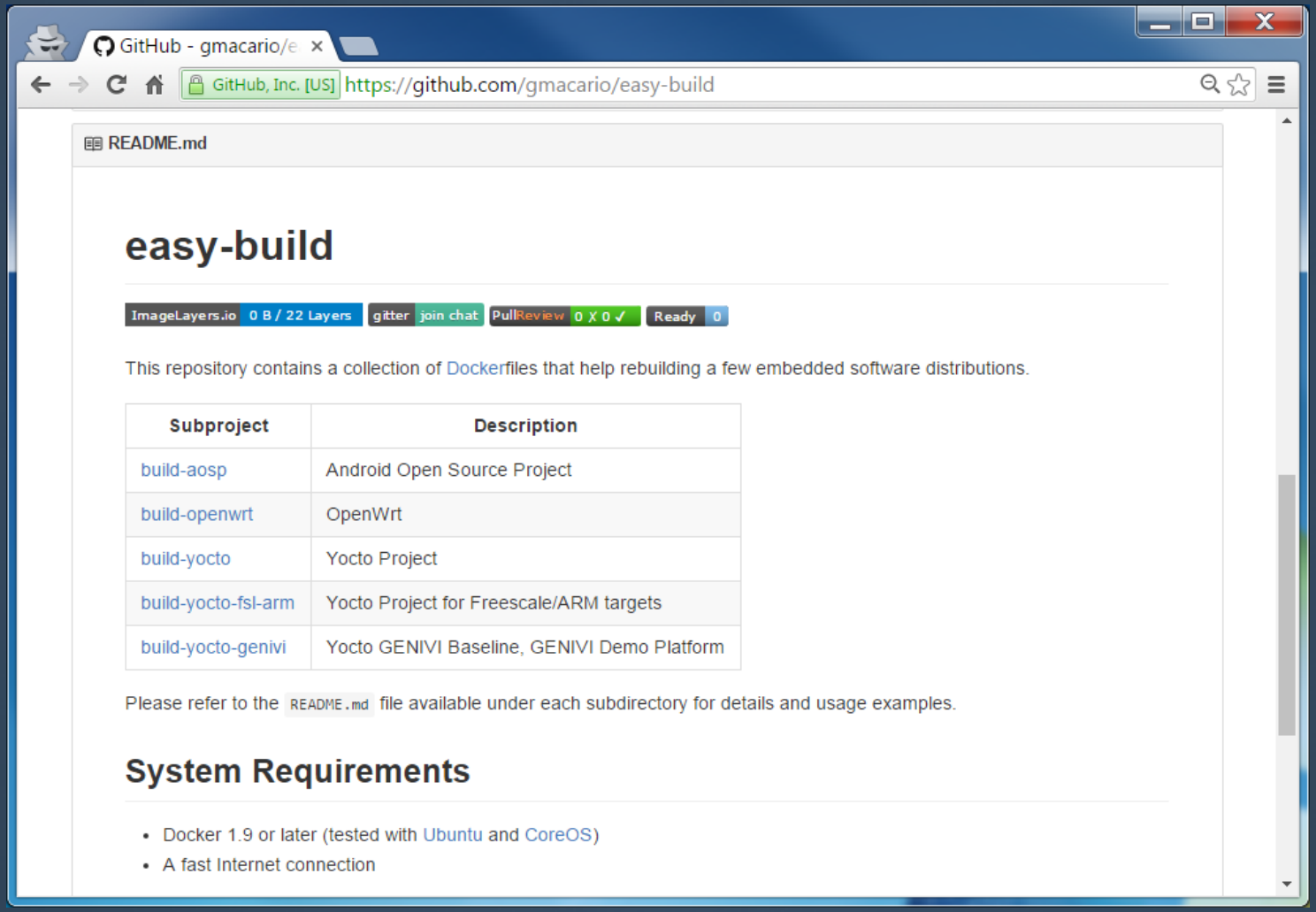


Image credit: https://insights.sei.cmu.edu/devops/2015/01/devops-and-docker.html

easy-build


GitHub - gmacario/e
GitHub, Inc. [US] https://github.com/gmacario/easy-build
README.md
easy-build
ImageLayers.io 0 B / 22 Layers
gitter join chat
PullReview 0 X 0 ✓
Ready 0
This repository contains a collection of Dockerfiles that help rebuilding a few embedded software distributions.
Subproject
Description
build-aosp
Android Open Source Project
build-openwrt
OpenWrt
build-yocto
Yocto Project
build-yocto-fsl-arm
Yocto Project for Freescale/ARM targets
build-yocto-genivi
Yocto GENIVI Baseline, GENIVI Demo Platform
Please refer to the README.md file available under each subdirectory for details and usage examples.
System Requirements
Docker 1.9 or later (tested with Ubuntu and CoreOS)
A fast Internet connection

The build is just one step

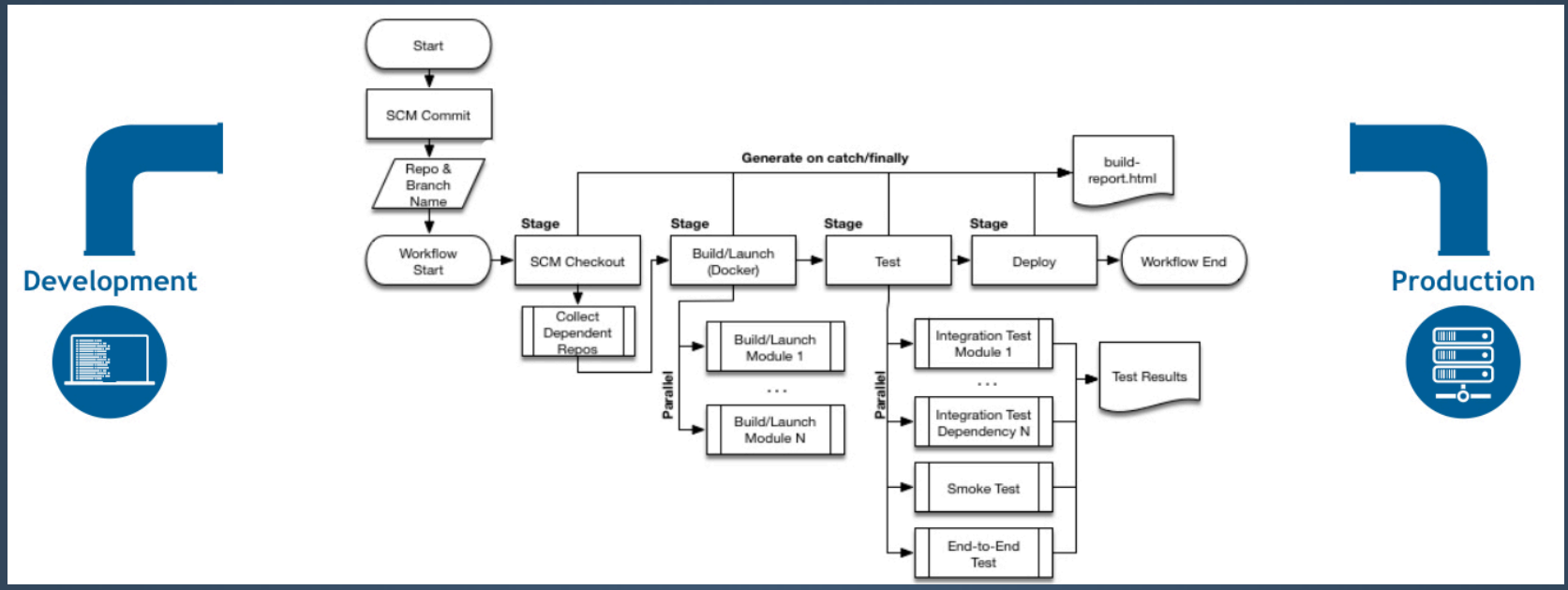
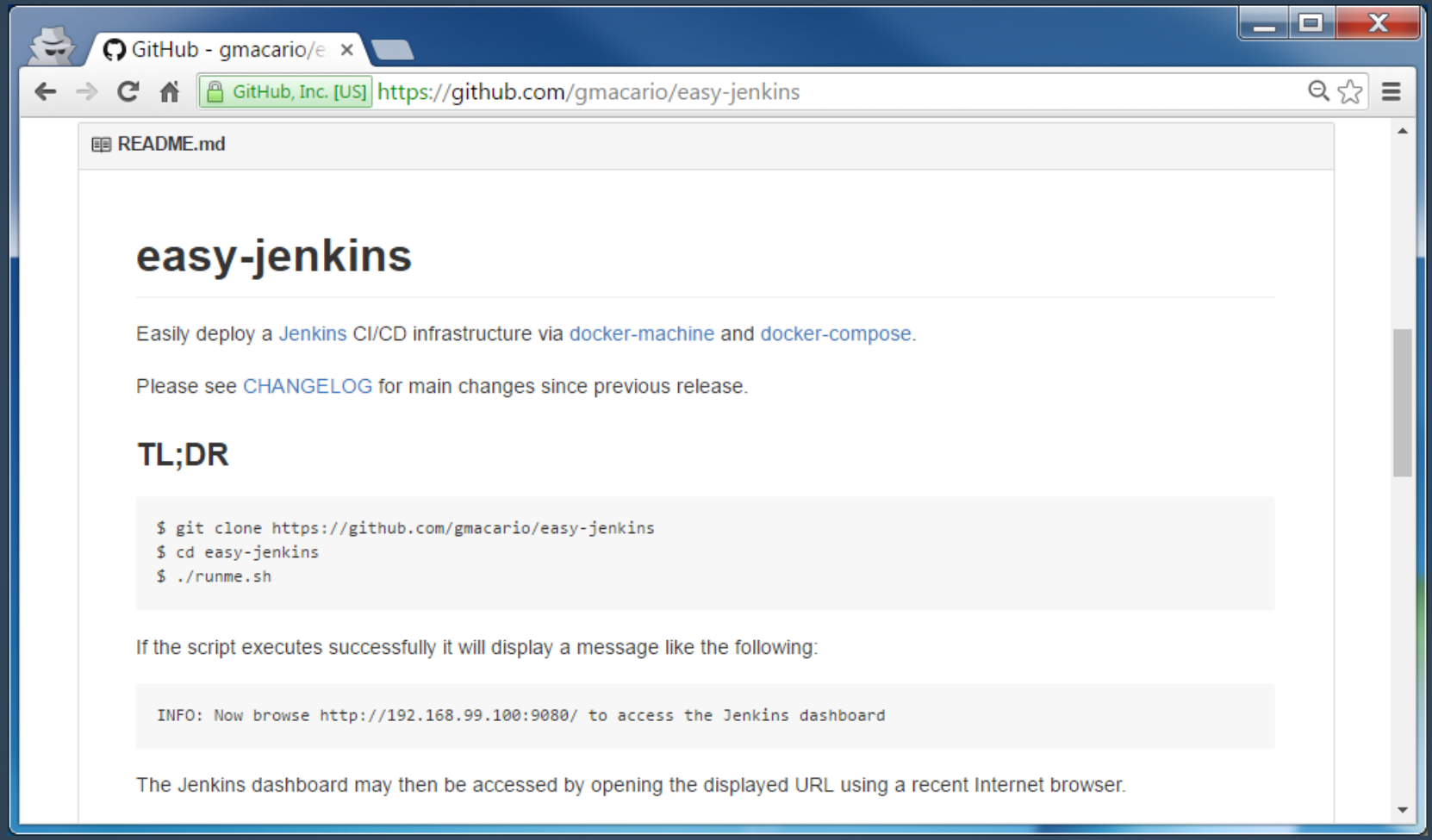


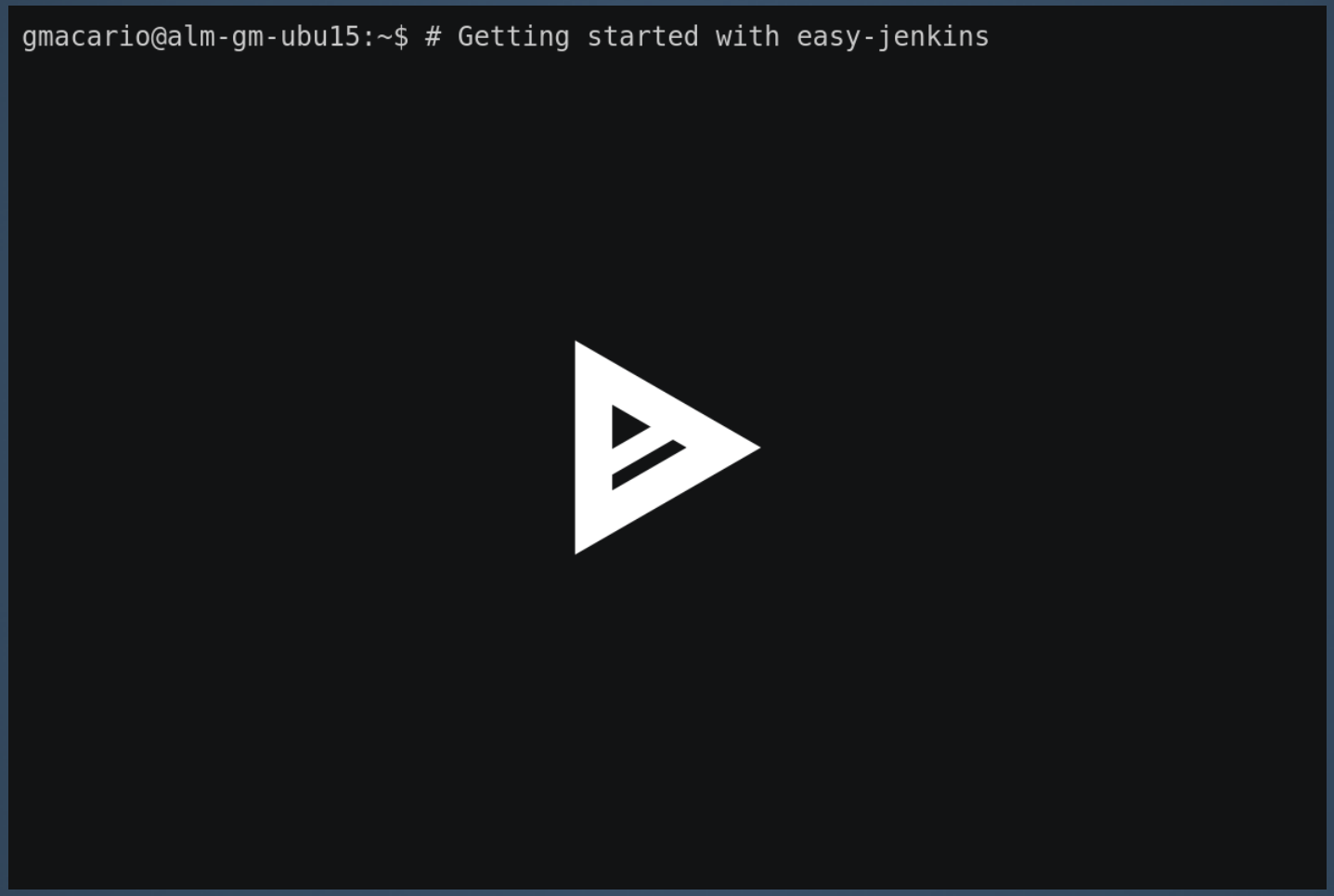
Image credit: https://jenkins.io/doc/pipeline/

Introducing Jenkins

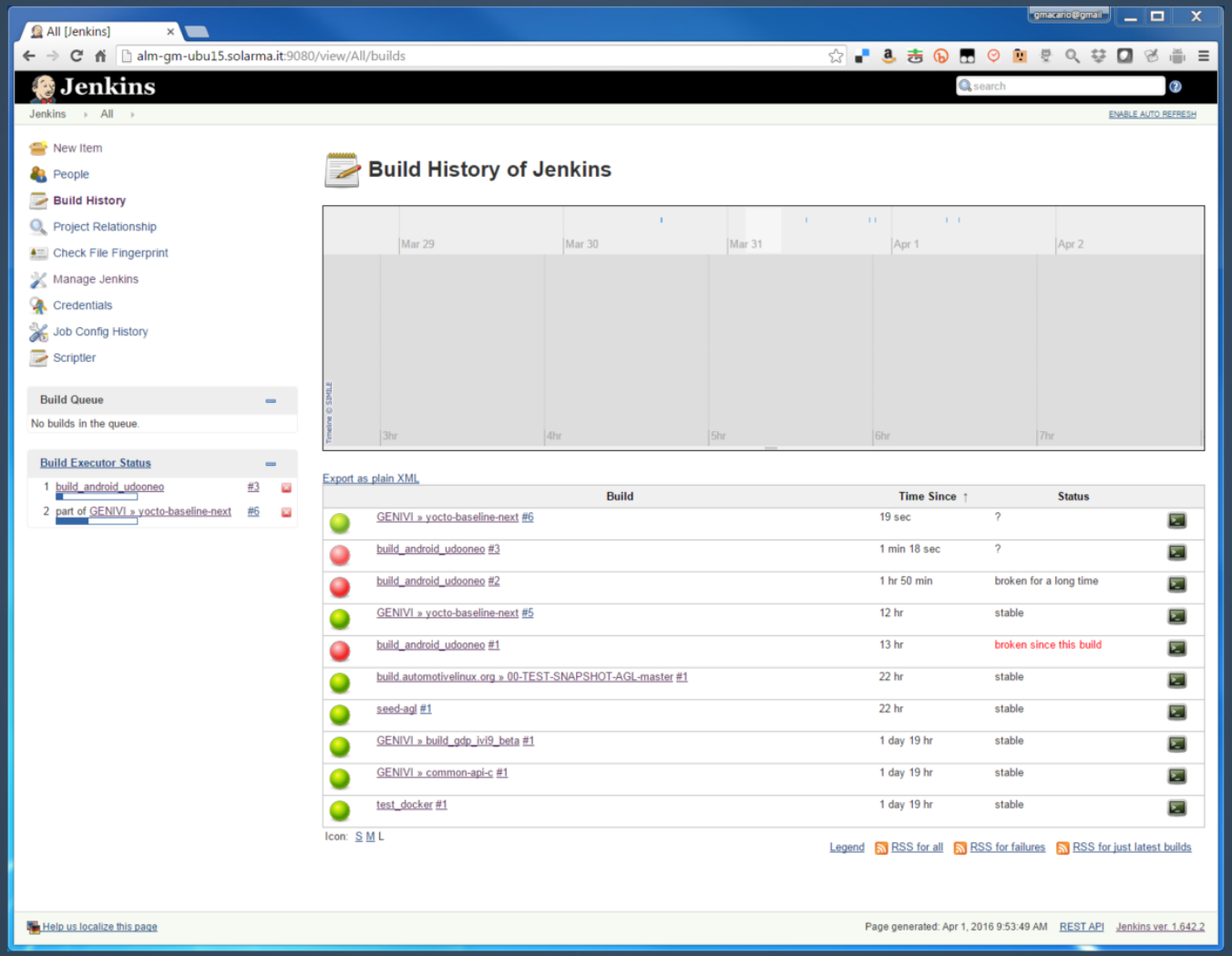
easy-jenkins



Running easy-jenkins



easy-jenkins in action



Summary

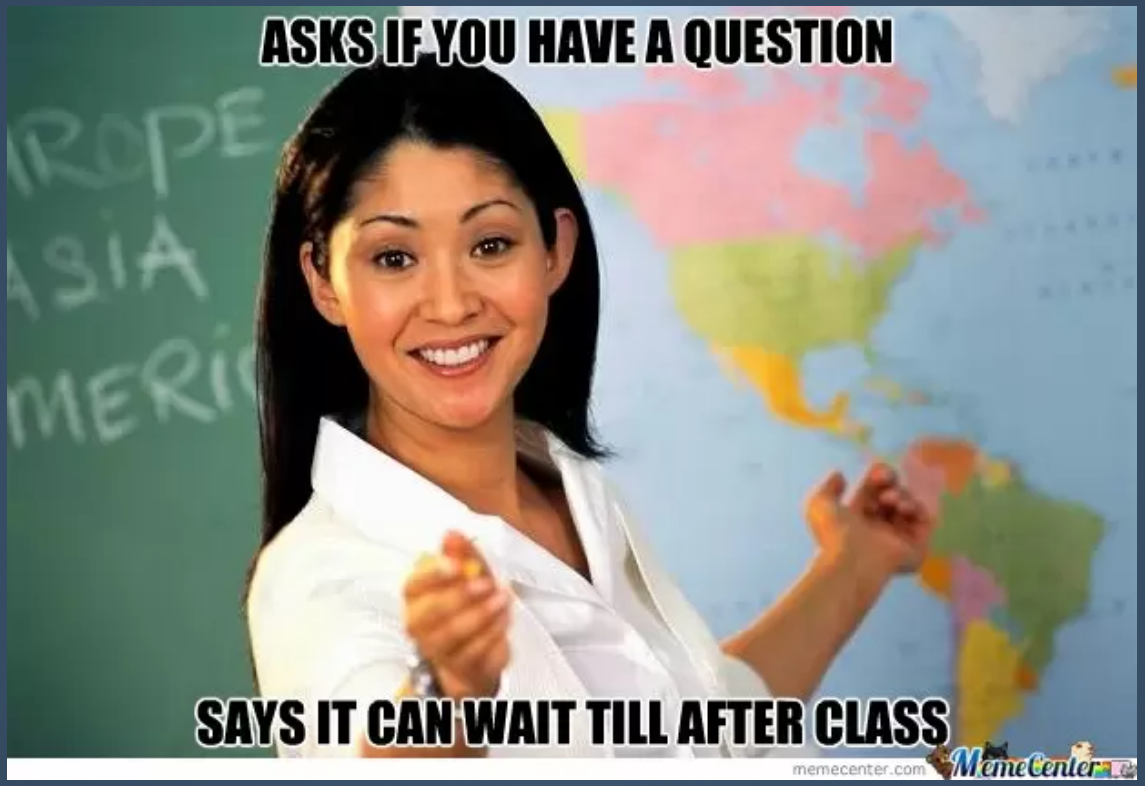
- Working on an embedded project requires
  - Setting up a customized build environment
  - Maintaining it during the whole project lifetime

- Create a reproducible build environment using Docker
  - Develop your own image, or start from easy-build

- Jenkins can automate the complete delivery pipeline

- Use easy-jenkins together with easy-build
  - Setup your CD infrastructure in minutes, not days
  - Upgrade your infrastructure config with a `git commit`

Thanks!



Gianpaolo Macario
https://gmacario.github.io/