Yocto Project Summit – Lyon
Day 2 : Friday 1 November 2019

Marco Cavallini, Leon Anavi, Jaewon Lee, Bernardo A. Rodrigues,

Manjukumar Harthikote Matha, Chandana Kalluri, Tim Orling, David Reyna

Presenter Slides:

https://wiki.yoctoproject.org/wiki/YP_Summit_Lyon_2019

Agenda – Yocto Project Summit - Day 2

9:00- 9:45	Strengthen your Yocto deployments with Autobuilder2 CI tool
9:50- 10:35	Working with NVIDIA Tegra BSP and Supporting Latest CUDA Versions
10:40-10:50	Morning Break
10:55-11:40	Sstate-cache Magic!
11:45-12:30	Bringing IOTA Distributed Ledger Technology (DLT) into Yocto/OpenEmbedded
12:35-1:20	Lunch
1:25- 2:55	Class: Devtool hands-on Seminar
3:00- 3:10	Afternoon Break
3:15- 4:45	Class: User Space 2.0 Seminar

https://wiki.yoctoproject.org/wiki/YP_Summit_Lyon_2019

1. Strengthen your Yocto deployments with Autobuilder2 CI tool

Marco Cavallini, KOAN
https://koansoftware.com

Who am I

- Founded **KOAN** on 1996
- Working with software for industrial automation until 1999
- Linux embedded devloper since 2000
- Openembedded board member since 2009
- Yocto Project participant since 2010
- Yocto Project Advocate since 2012

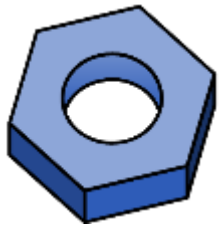
- **Software development and consulting**
- **BSP creation**
- **Device driver kernel development**
- **Open embedded and Yocto Project trainmg**

Agenda

- What is a Continuous Integration (CI) system
- Autobuilder2 History
- Buildbot, the foundations
- Buildbot mechanics
- Buildbot installation
- Autobuilder2 installation
- Autobuilder2 configuration
- Autobuilder2 usage (as-is)
- Autobuilder2 customization
- Autobuilder2 usage for CI on a single machine

What is Autobuilder

- Autobuilder is a project based on Buildbot

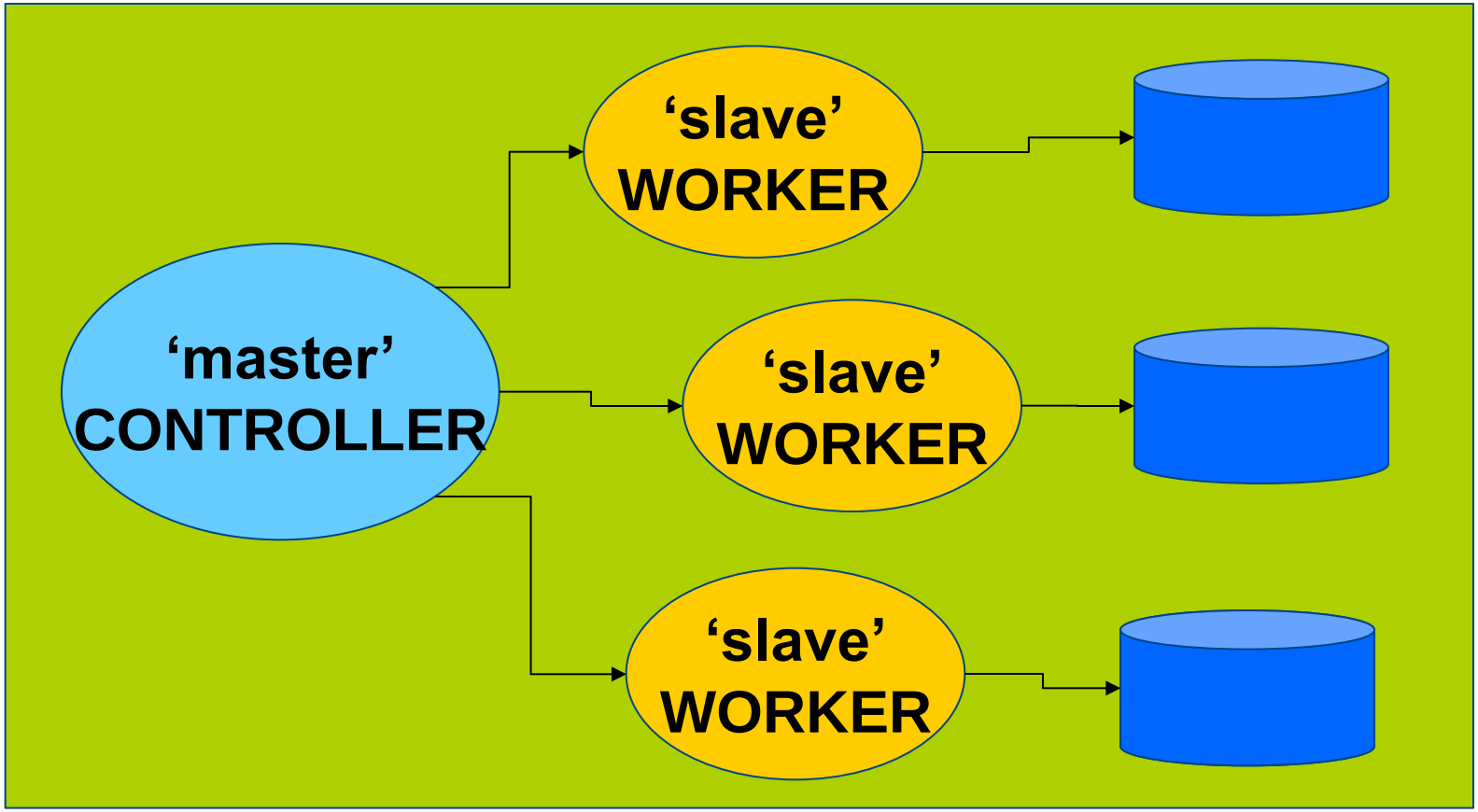


- Buildbot is a Python open-source application used to build, test, and release a wide variety of software.

- Autobuilder and Buildbot are licensed under GPLv2

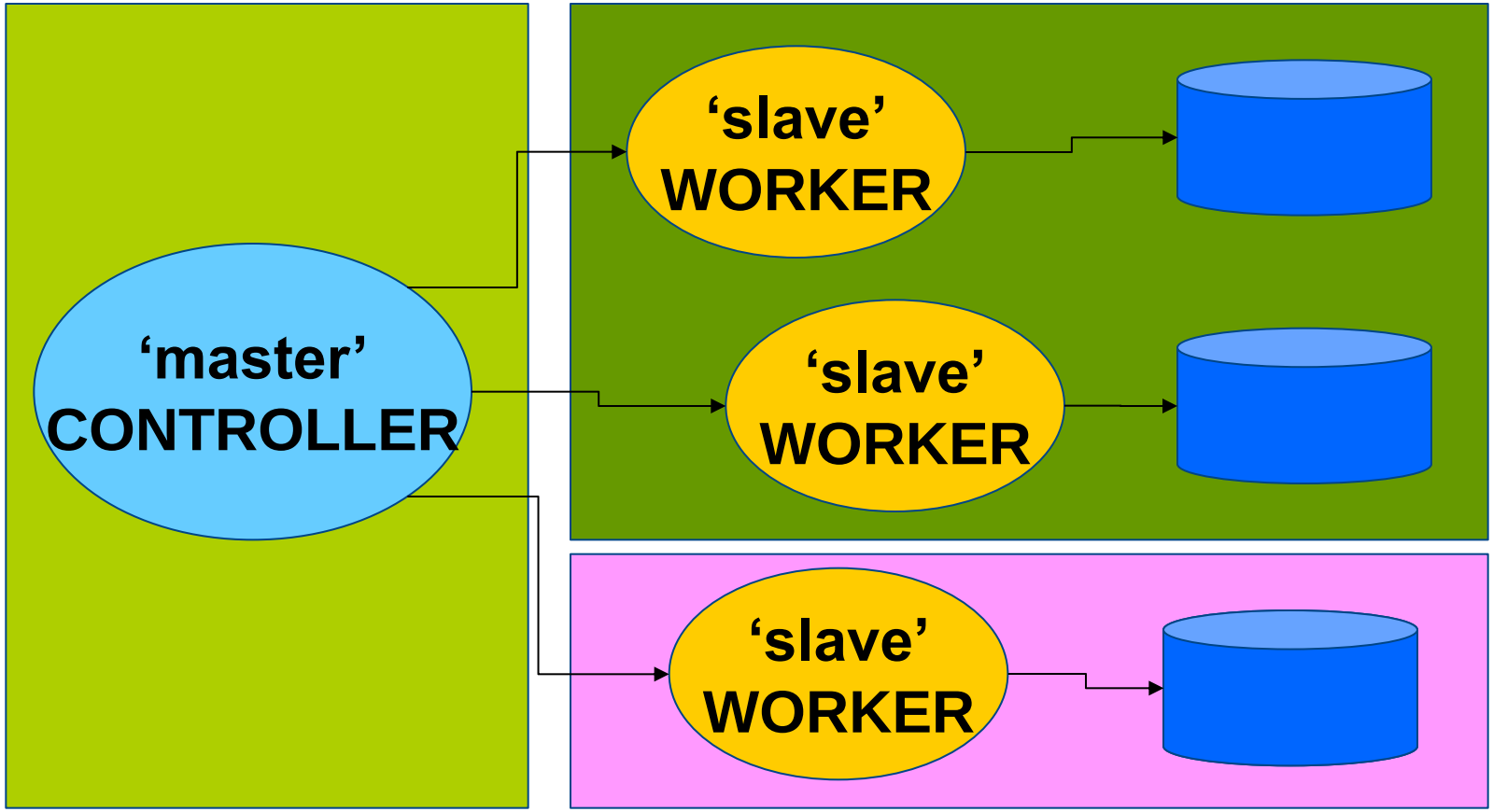
CI overview

- Typical CI on a single machine



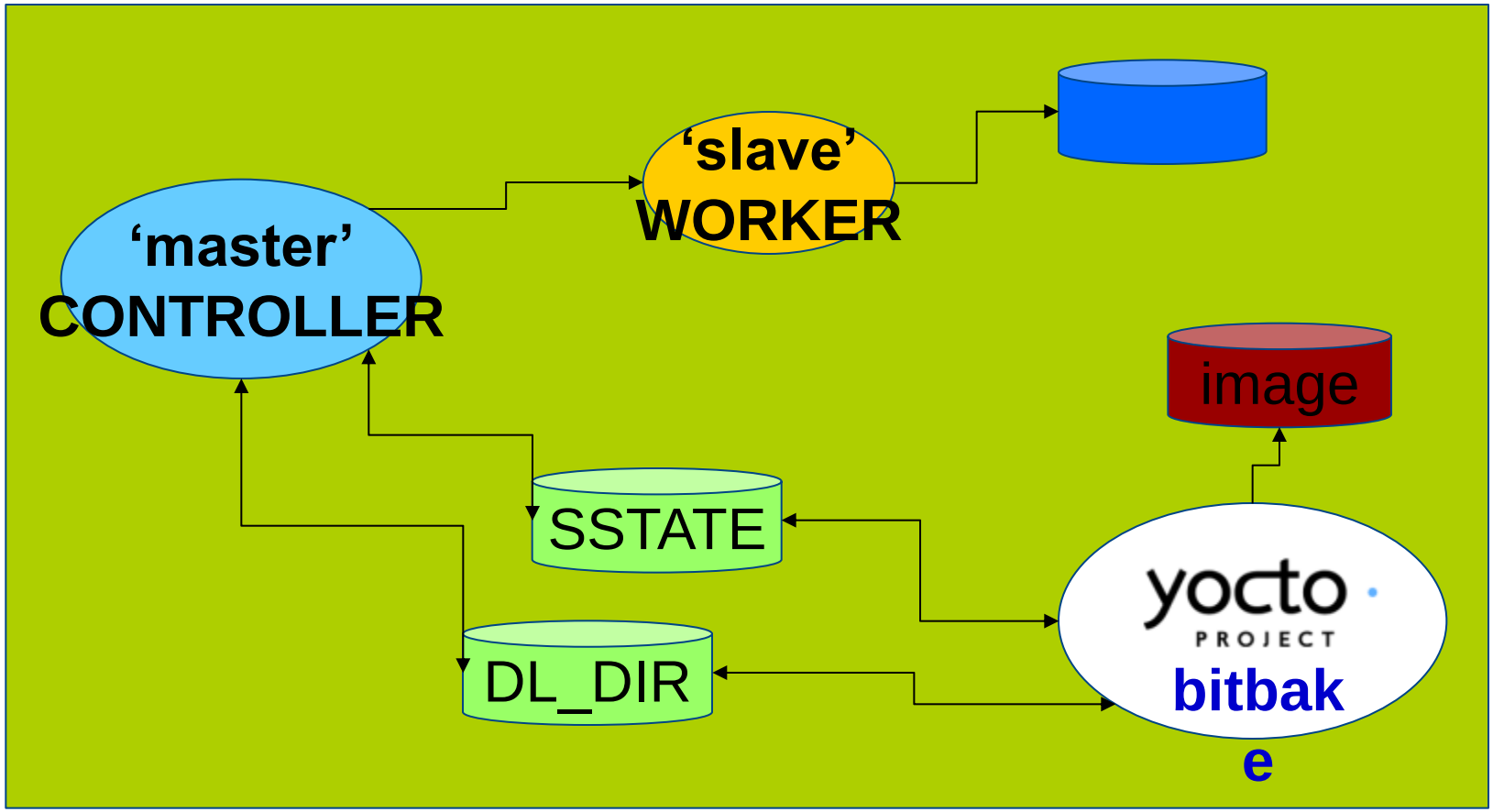
CI overview

- Typical CI on a distributed system

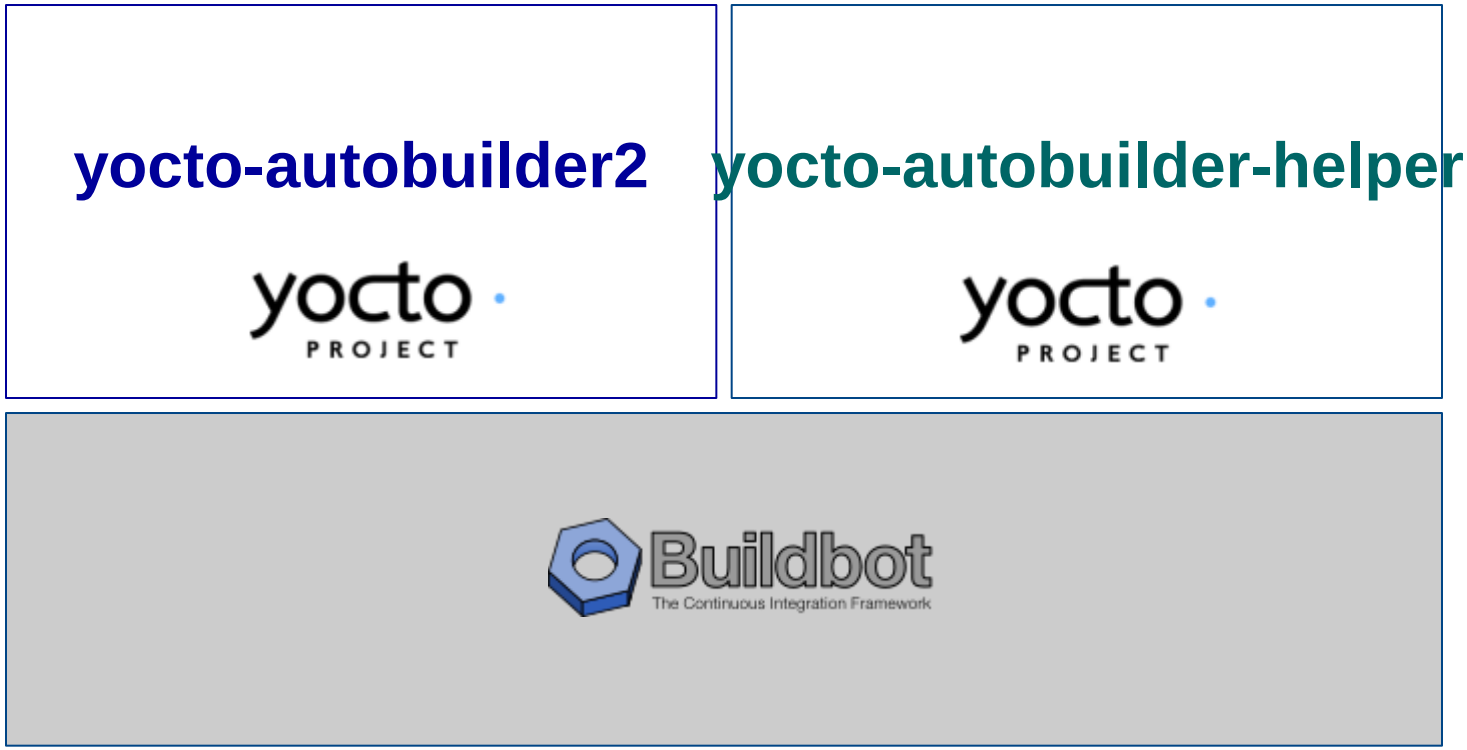


Our goal

- Speed Yocto builds by populating premirrors with Autobuilder2



Autobuilder overview



Autobuilder history *

- **Creation of Autobuilder**
  - The autobuilder started life as something OpenedHand used for testing Poky linux.

- **Yocto-autobuilder**
  - It became "yocto-autobuilder" under Beth's stewardship in December 2012 when it was totally re-implemented.

- **Autobuilder2**
  - In Feburary 2018 it was rewritten again, in particular to move from the long obsolete "buildbot eight" codebase to the "buildbot nine" one but also to fix many long running issues and get back to using an upstream codebase.

** Thanks to Richard Purdie who provided these information*

People behind Autobuilder *

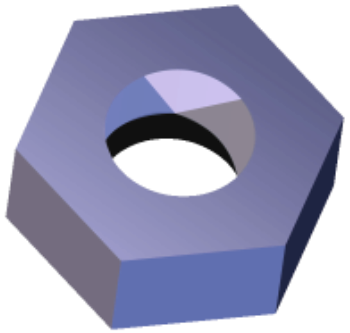
- **Project Autobuilder**
  - Richard Purdie, Elizabeth Flanagan, Joshua Lock as well as contributions from Tracy Graydon, Anibal Limón and Bill Randle.

- **Project Autobuilder2**
  - Richard Purdie and Joshua Lock.
  - Michael Halstead is the project sysadmin who maintains the infrastructure it all runs on top of.

** Thanks to Richard Purdie who provided these information*

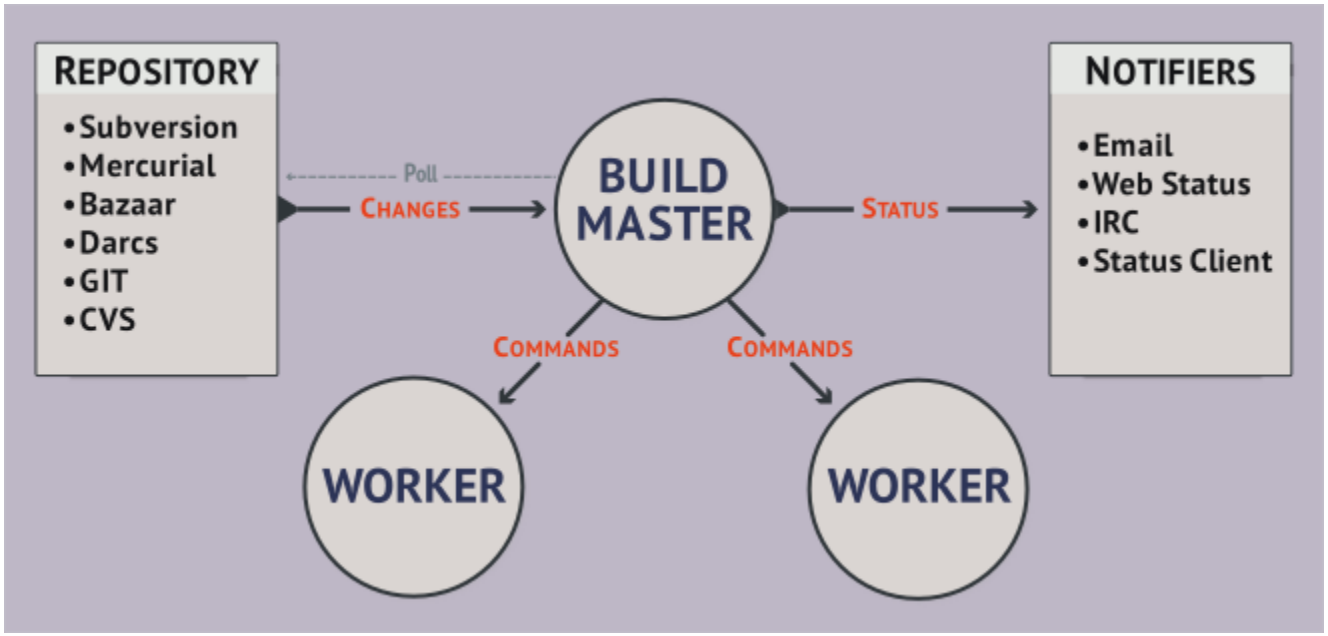
Buildbot

Buildbot, a CI framework for Python



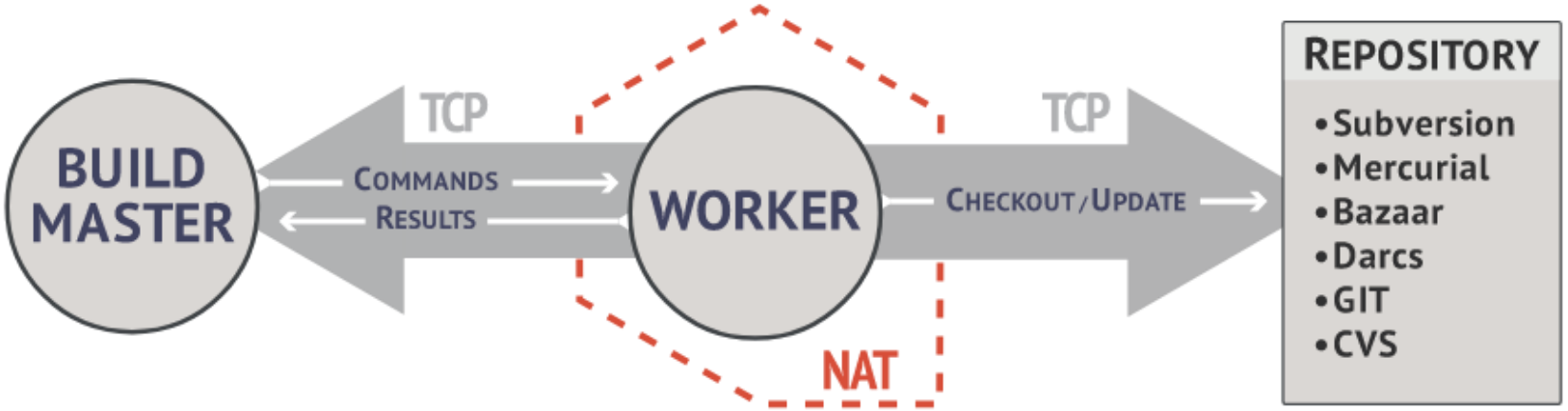
Buildbot basics

- **Buildbot is a job scheduling system**
  - it queues jobs, executes the jobs when the required resources are available, and reports the results

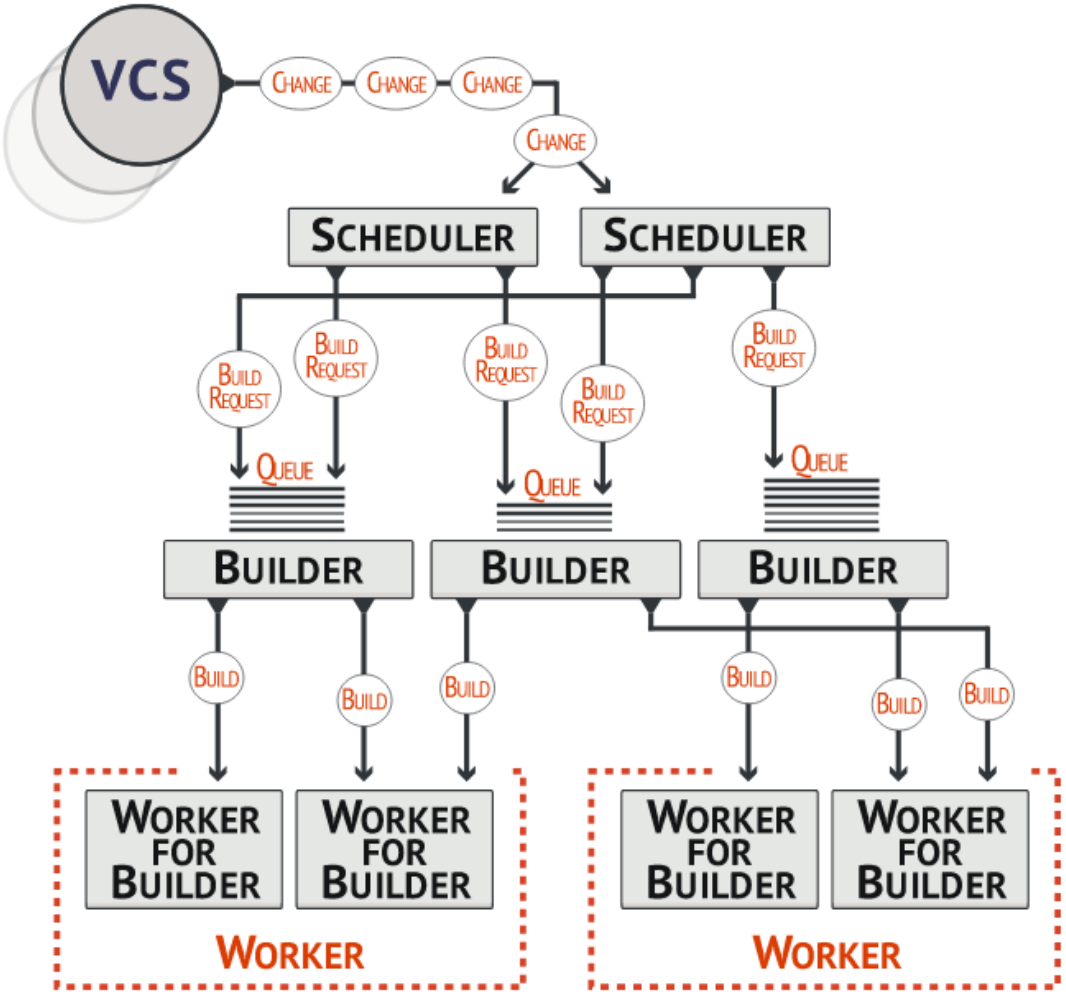


Buildbot basics

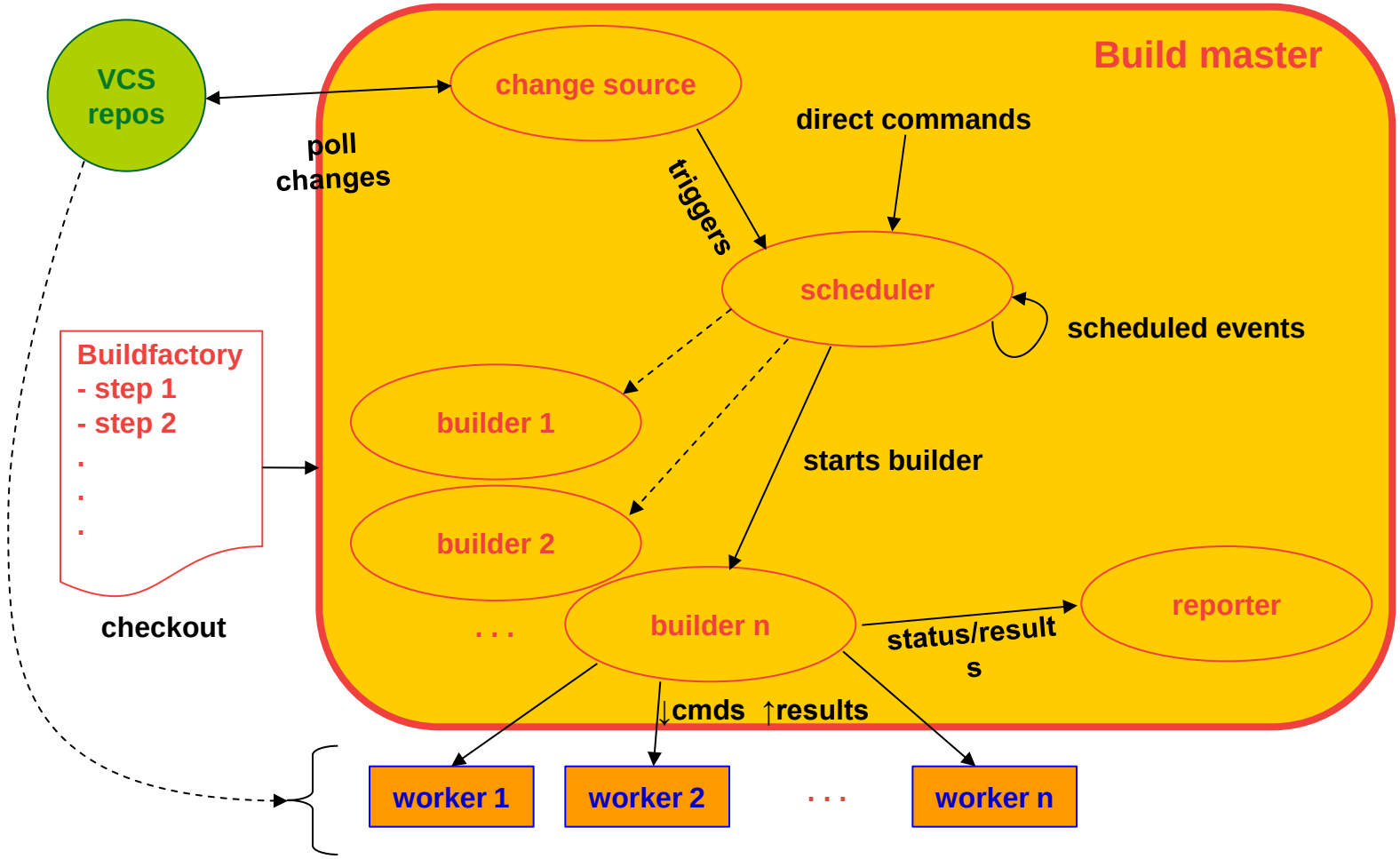
- **Workers are typically run on separate machines**



Buildbot basics



What happens inside the master



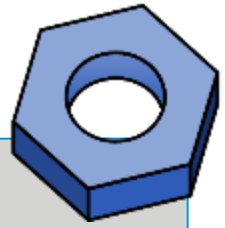
** artwork by Mauro Salvini*

Buildbot installation

- **On a native system**
  - Probably the fastest solution
- **In a Python sandbox**
  - Isolates it from the host system
  - Using ***pip***
- **In a Docker container**
  - Isolates it from the host system

https://docs.buildbot.net/2.4.0/full.html

Buildbot installation (in a Python sandbox) [1/3]



- Create a sandbox

```
mkdir abot-sandbox
cd abot-sandbox

python3 -m venv sandbox
source sandbox/bin/activate
```

- Install master

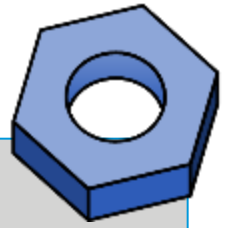
```
pip install --upgrade pip
pip install 'buildbot[bundle]'
```

- Install worker

```
pip install --upgrade pip
pip install buildbot-worker
```

Buildbot installation (in a Python sandbox) [2/3]

- **Create the master**



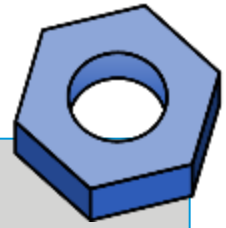
```
buildbot create-master master
mv master/master.cfg.sample master/master.cfg
```

- **Create the worker**

```
pip install setuptools-trial
buildbot-worker create-worker worker localhost example-worker pass
```

Buildbot installation (in a Python sandbox) [3/3]

- **Content of the sandbox**



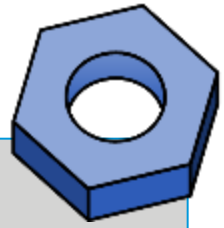
```
(sandbox) koan@amonra:~/abot-sandbox$ tree -L 2
.
├── master
│   ├── buildbot.tac
│   ├── master.cfg          →   c['workers'] = [worker.Worker("example-worker", "pass")] (*)
│   └── state.sqlite
├── sandbox
│   ├── bin
│   ├── include
│   ├── lib
│   ├── lib64 -> lib
│   ├── pip-selfcheck.json
│   ├── pyvenv.cfg
│   └── share
└── worker
    ├── buildbot.tac        → workername = 'example-worker' (*)
    └── info
```

(*) ftp://ftp.koansoftware.com/public/opensource/buildbot/

Buildbot execution

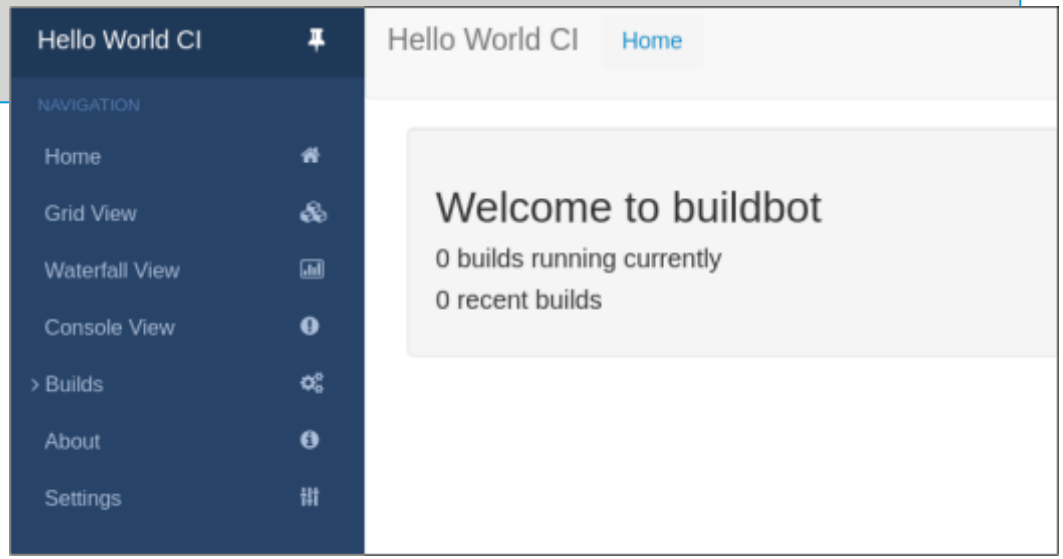
- **Execution of master**

```
buildbot start master
```



- **Control of the build system (using your browser)**

```
http://localhost:8010/
```



Buildbot

- To be continued in a dedicated session…
- Now let's have a look at Autobuilder

Autobuilder2

Buildbot vs. Autobuilder2 lexicon

Buildbot	Autobuilder2
Master	Controller
Worker	Worker

Autobuilder2 installation

- **After you created the sandbox**
- **Create the master and worker directories**

```
buildbot create-master -r yocto-controller

buildbot-worker create-worker -r /
                  --umask=0o22 yocto-worker localhost example-worker pass
```

- yocto-controller is the directory for master
- --umask sets the proper permissions
- yocto-worker is the directory for worker
- localhost is the network address of the master
- example-worker is the name of the worker
- pass is the password (master.cfg)

Autobuilder2 installation

- **Clone yocto-autobuilder2**

```
cd yocto-controller
git clone https://git.yoctoproject.org/git/yocto-autobuilder2 yoctoabb
ln -rs yoctoabb/master.cfg master.cfg
```

  - yoctoabb is the mandatory Autobuilder2 directory name

- **Clone yocto-autobuilder-helper**

```
cd ..
git clone https://git.yoctoproject.org/git/yocto-autobuilder-helper
```

Autobuilder2 tree

- **Content of the Autobuilder2 sandbox**

** Simplified tree list*

```
(sandbox) koan@amonra:~/ab2-sandbox$ tree -L 3
.
├── autobuilder
├── git
├── sandbox
├── yocto-autobuilder-helper
├── yocto-controller
│   └── yoctoabb
│       └── master.cfg
└── yocto-worker
    └── buildbot.tac
```

Autobuilder2 installation

- **Complete installation instructions**

http://git.yoctoproject.org/cgit.cgi/yocto-autobuilder2/tree/README-Guide.md

ftp://ftp.koansoftware.com/public/talks/YoctoSummit-2019/autobuilder2/Autobuilder2-README-Guide.pdf

Autobuilder2

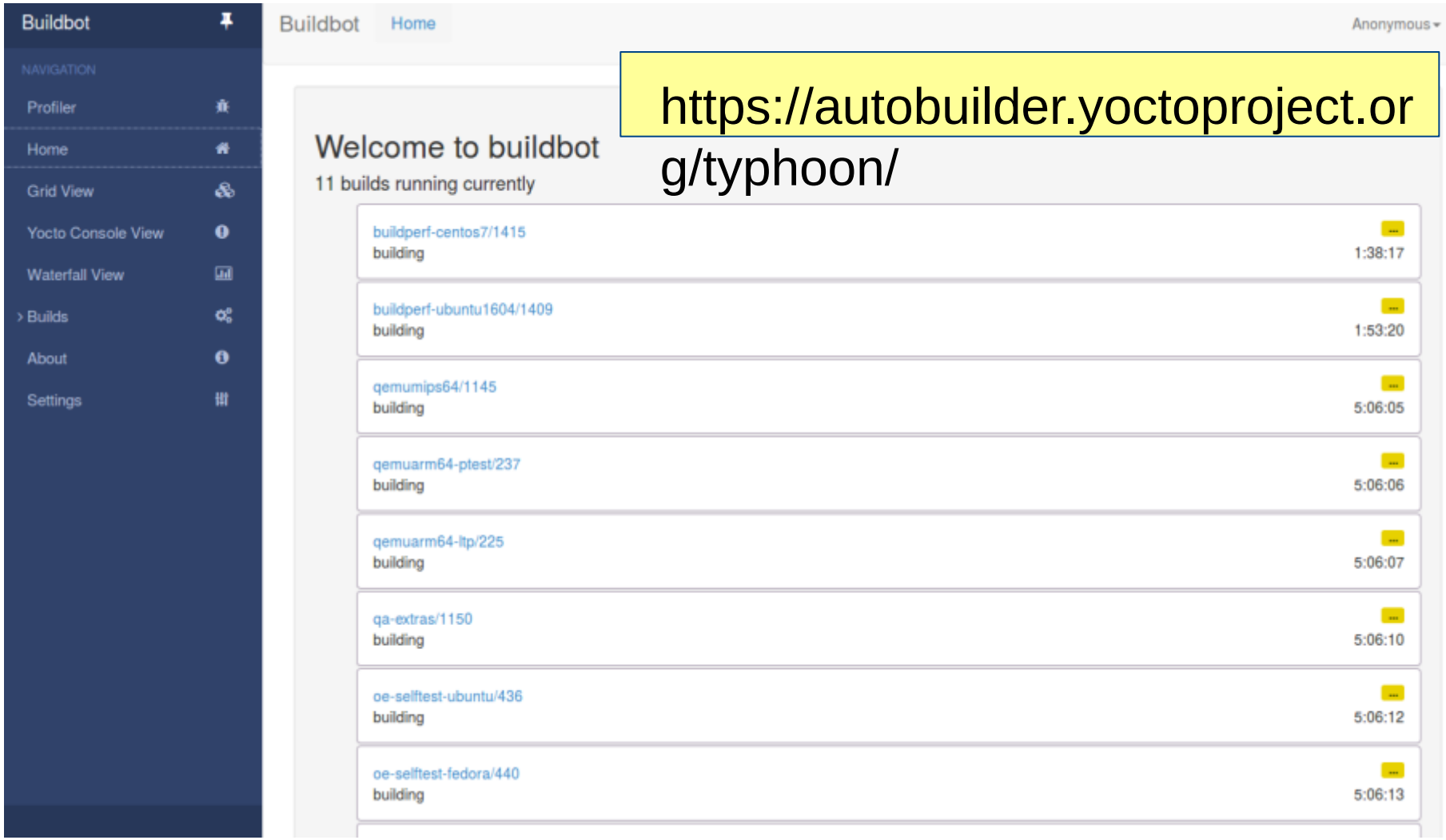
Autobuilder default configuration

Autobuilder2 configuration

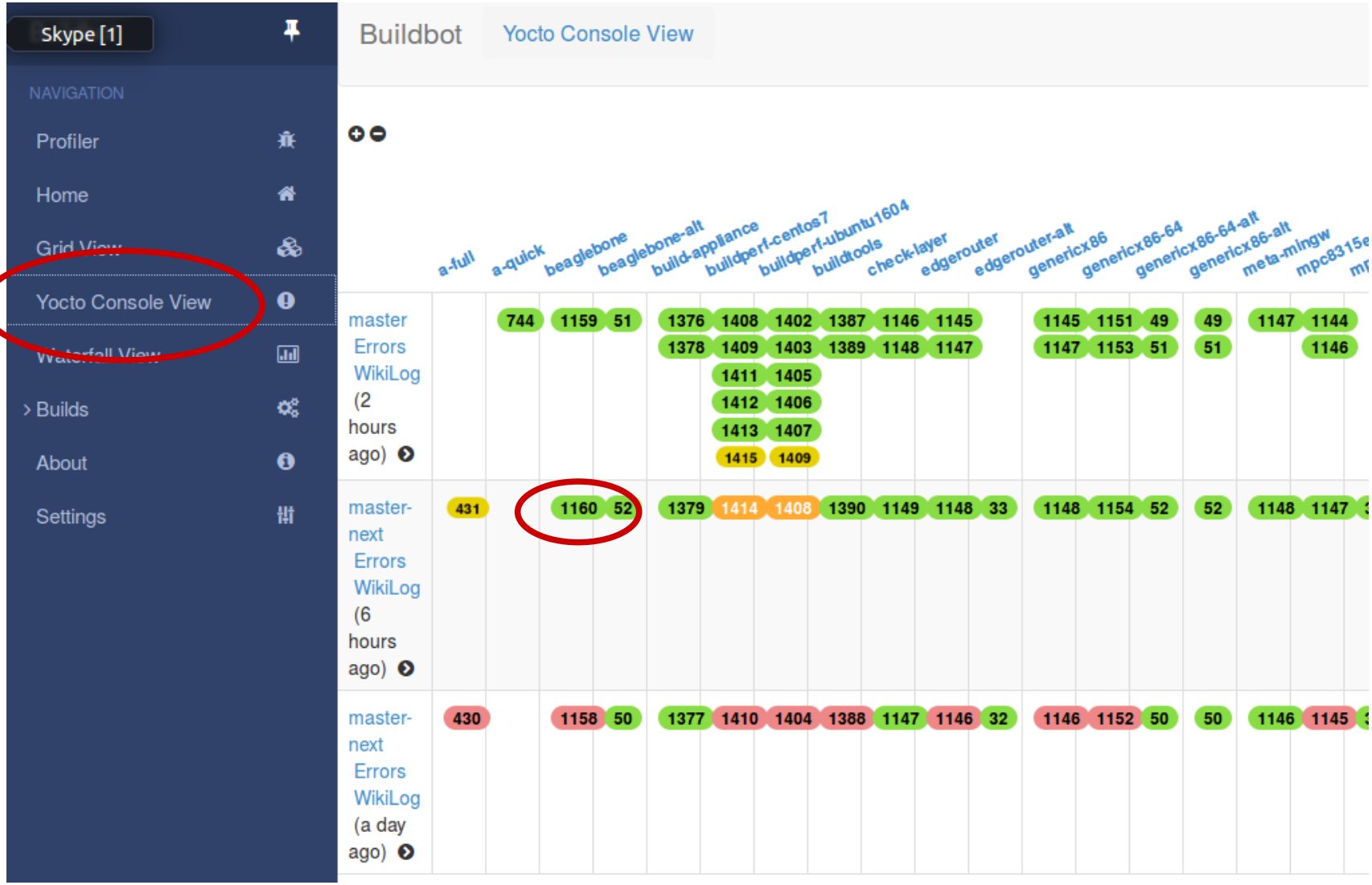
- **Ab2 default configuration**
  - The default configuration of Ab2 uses a lot of **workers** to generate **images** for several **MACHINES**

```
# List of workers in the cluster
workers_ubuntu = ["ubuntu1904-ty-1", "ubuntu1804-ty-1", "ubuntu1804-ty-2", "ubuntu1804-ty-3", "ubuntu1604-ty-1"]
workers_centos = ["centos7-ty-1", "centos7-ty-2", "centos7-ty-3", "centos7-ty-4"]
workers_fedora = ["fedora29-ty-1", "fedora30-ty-1", "fedora30-ty-2"]
workers_debian = ["debian8-ty-1", "debian9-ty-2", "debian10-ty-1", "debian10-ty-2", "debian10-ty-3"]
workers_opensuse = ["tumbleweed-ty-1", "tumbleweed-ty-2", "tumbleweed-ty-3", "opensuse151-ty-1", "opensuse150-ty-1"]
```

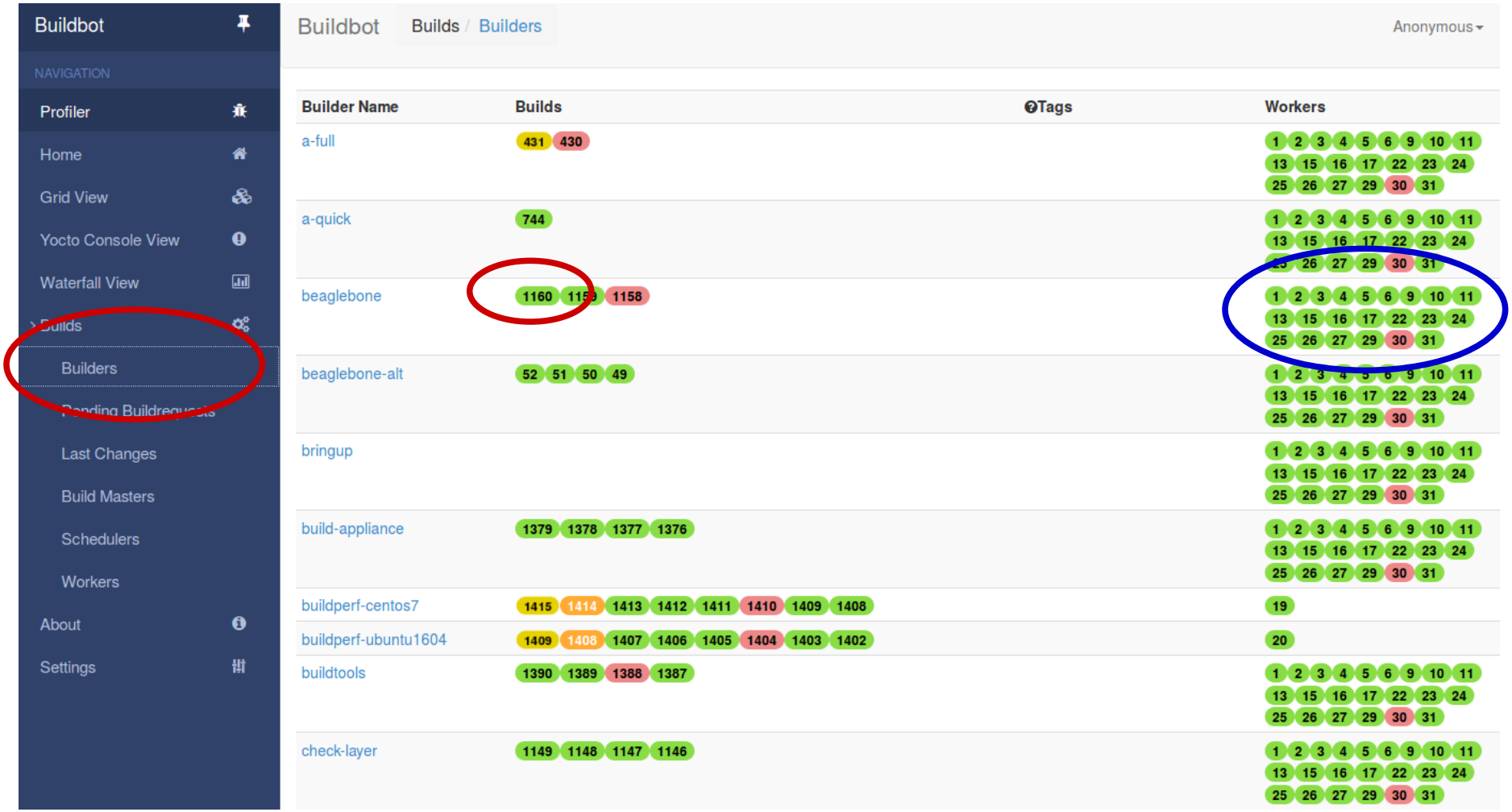
Autobuilder2 official website



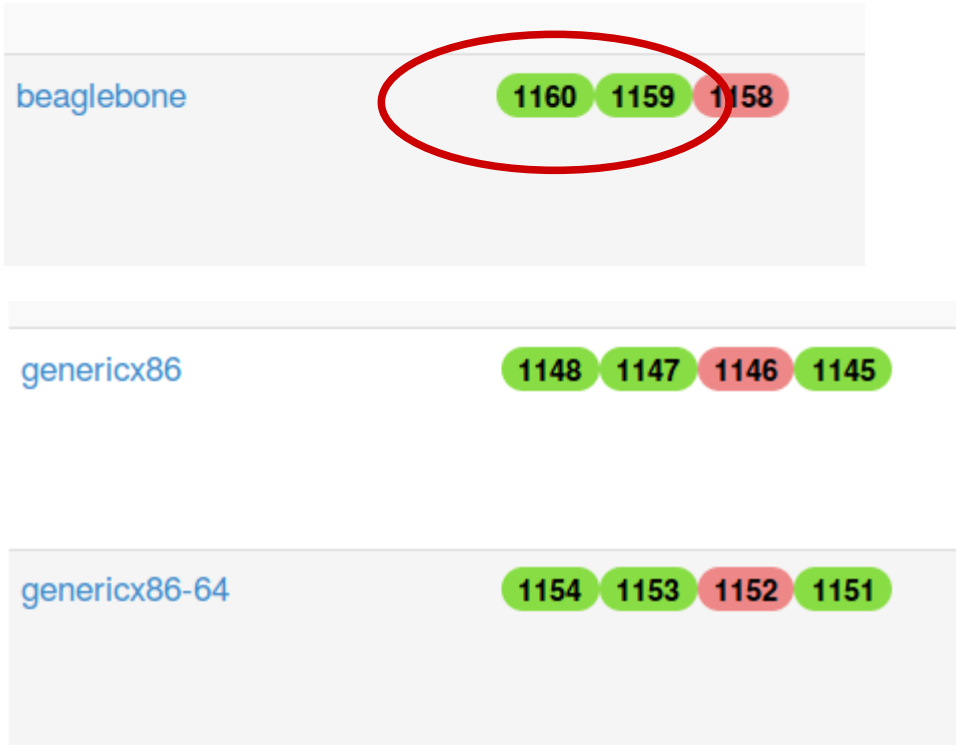
Autobuilder2 website navigation



Autobuilder2 website navigation



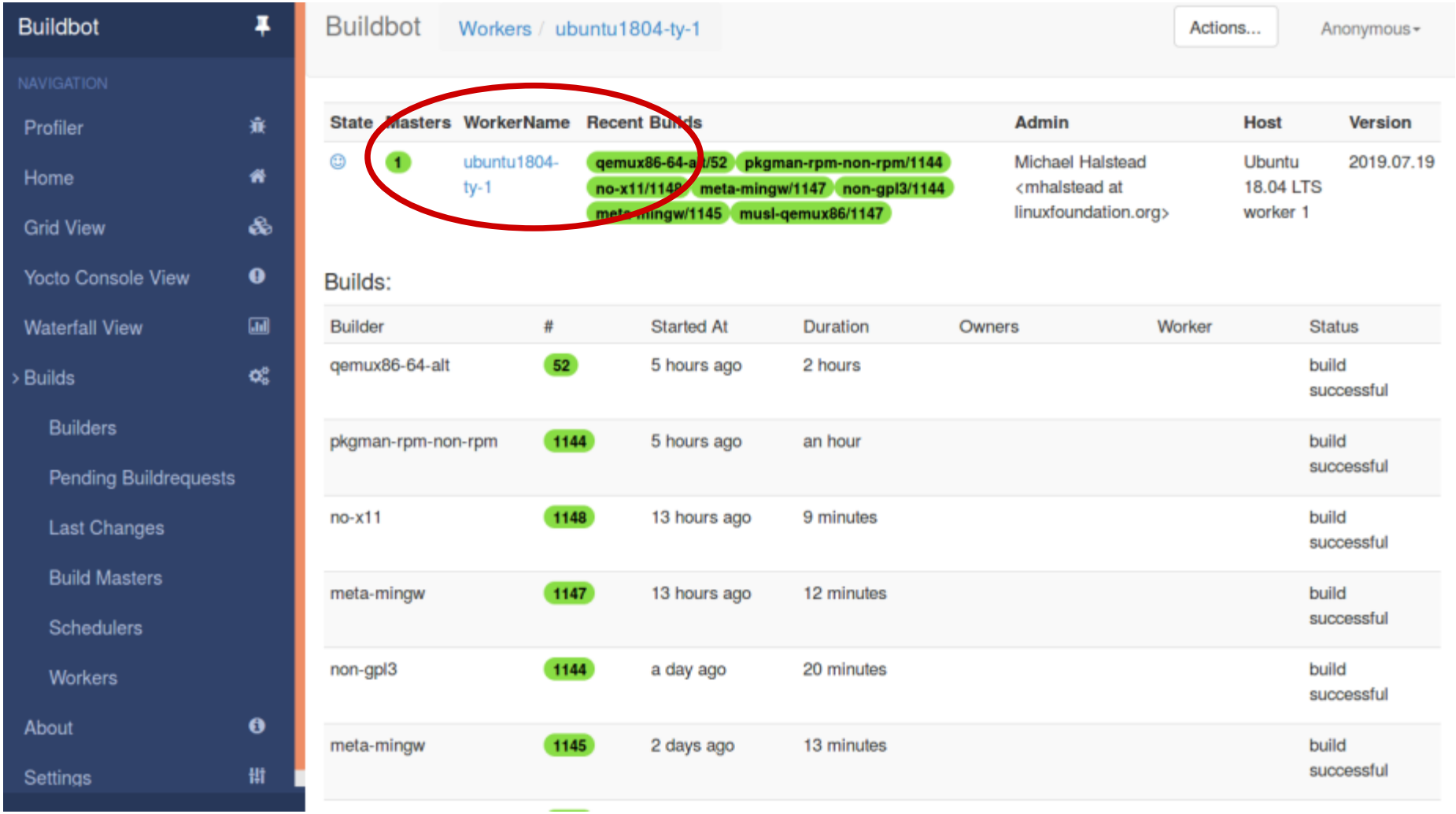
Autobuilder2 - builds



Autobuilder2 - workers

		25 26 27 29 30 31
beaglebone	1160 1159 1158	1 2 3 4 5 6 9 10 11 13 15 16 17 22 23 24 25 26 27 29 30 31
beaglebone-alt	52 51 50 49	1 2 3 4 5 6 9 10 11 13 15 16 17 22 23 24 25 26 27 29 30 31
bringup		1 2 3 4 5 6 9 10 11 13 15

Autobuilder2 – worker details

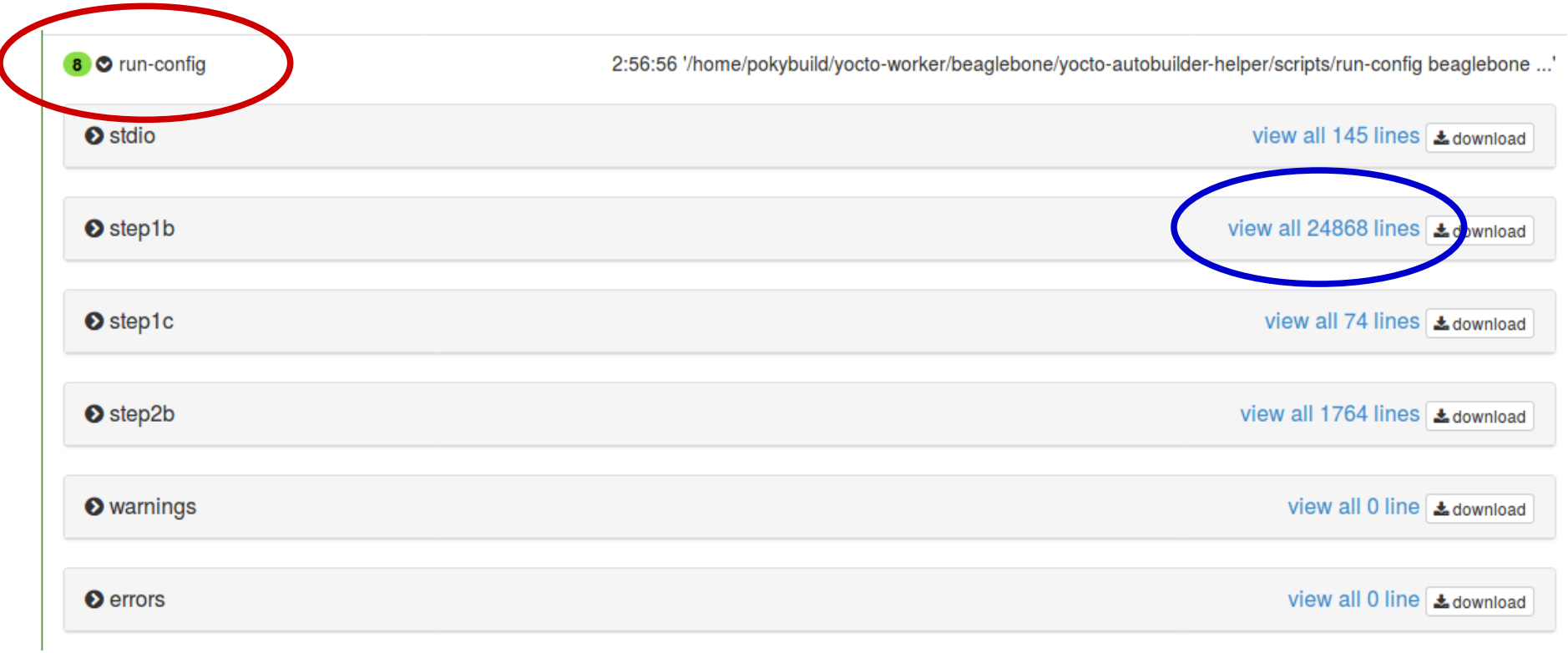


Autobuilder2 – build details

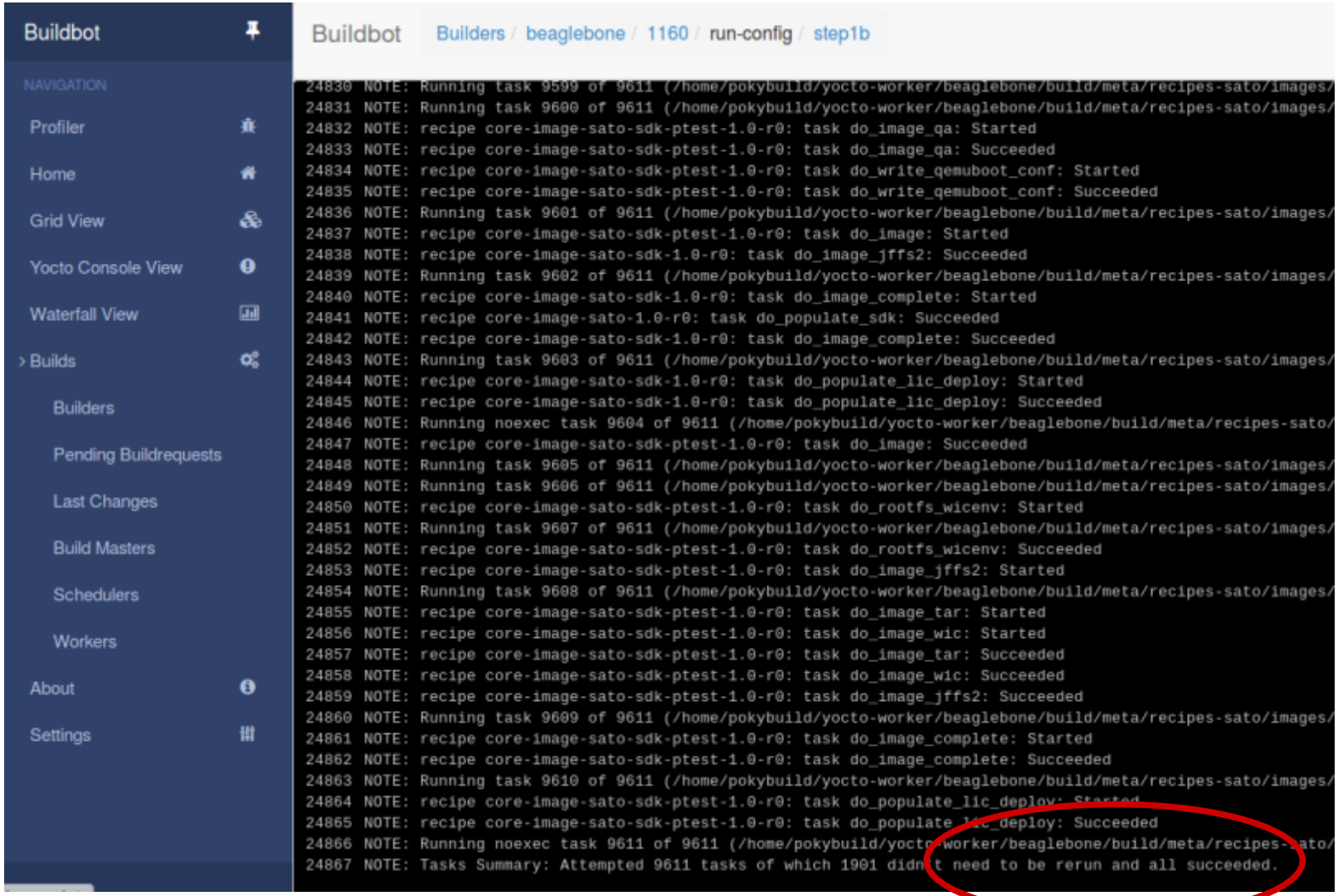
Build summary

	beaglebone/1160	2:57:18 build successful SUCCESS
0	worker_preparation	5 s worker ready
1	Clobber build dir	2 s '/home/pokybuild/yocto-autobuilder-helper/janitor/clobberdir /home/pokybuild/yocto-worker/beaglebone/'
2	Fetch yocto-autobuilder-helper	3 s update
3	SetProperties	1 s Properties Set
4	Write main layerinfo.json	1 s 'printf '{ ...'
5	5 s '/home/pokybuild/yocto-worker/beaglebone/yocto-autobuilder-helper/scripts/shared-repo-unpack /home/pokybuild/yocto-worker/beaglebone/layerinfo.json ...' Unpack shared repositories	
6	Set build revision	1 s property 'yp_build_revision' set
7	Set build branch	1 s property 'yp_build_branch' set
8	run-config	2:56:56 '/home/pokybuild/yocto-worker/beaglebone/yocto-autobuilder-helper/scripts/run-config beaglebone ...'

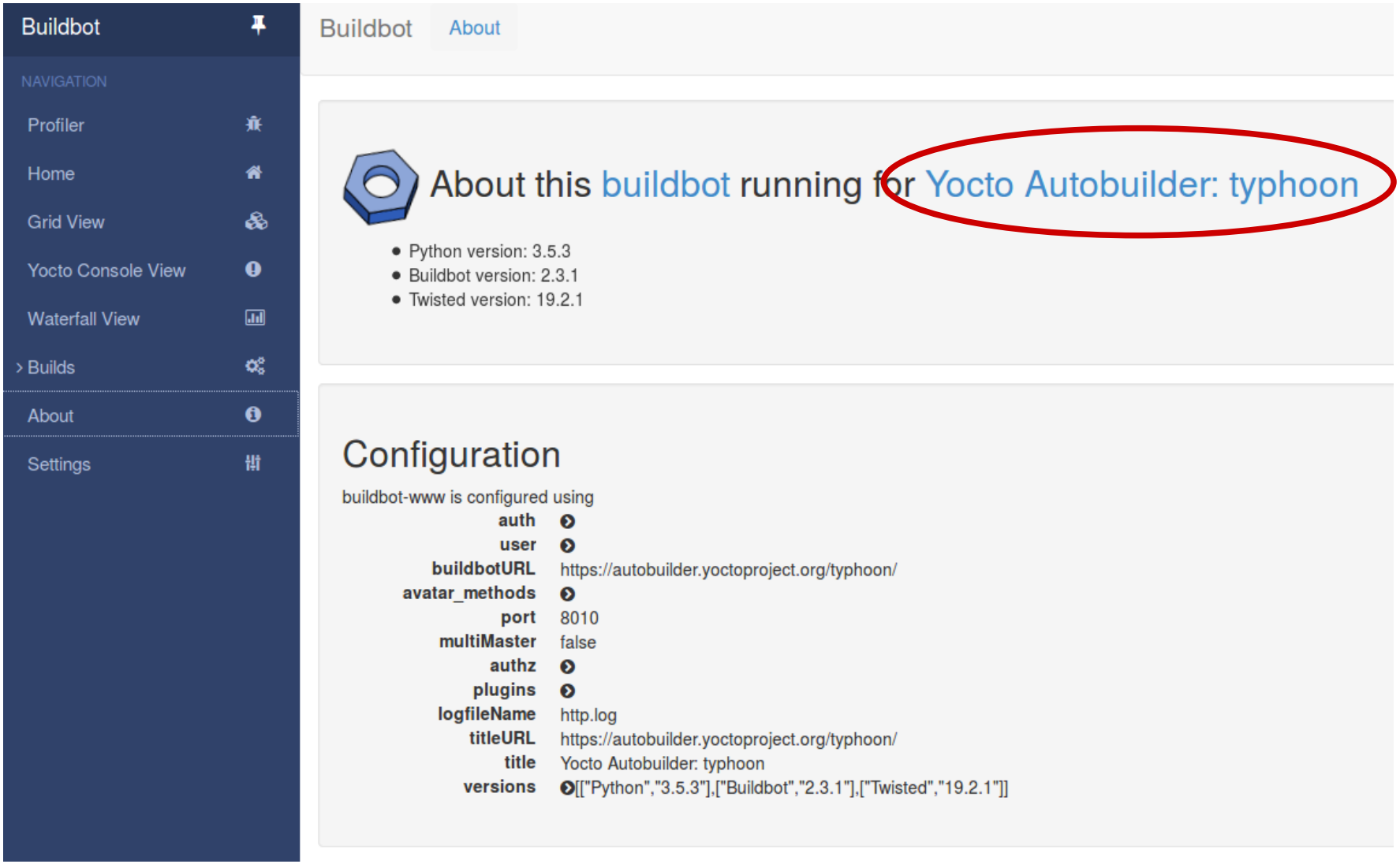
Autobuilder2 – build details



Autobuilder2 – build details



Autobuilder2 – build details



Autobuilder2

Autobuilder custom ‘lighter’ configuration

Autobuilder2 reduced configuration

- **Reduce complexity**
  - The goal is to setup a configuration for an **image** for a single **MACHINE** only
  - This will help yo strenghten the deployments thanks to recurring builds, tipically nightly

Autobuilder2 reduced configuration

- **Files to be modified**

```
.
├── yocto-autobuilder-helper
│   └── config.json
├── yocto-controller
│   └── yoctoabb
│       ├── builders.py
│       ├── config.py
│       ├── master.cfg
│       └── schedulers.py
└── yocto-worker
    └── buildbot.tac
```

Autobuilder2 configuration

- **In yocto-autobuilder-helper**
  - Edit yocto-autobuilder-helper/config.json

```
        "BASE_HOMEDIR" : "/home/koan/ab2-sandbox",
        "BASE_SHAREDDIR" : "${BASE_HOMEDIR}/autobuilder",
        "BASE_PUBLISHDIR" : "${BASE_HOMEDIR}/downloads",
```

- **In yocto-controller**
  - Edit yocto-controller/yoctoabb/master.cfg

```
    c['title'] = "KOAN lite Yocto Autobuilder"
    c['titleURL'] = "http://localhost:8010/"
    c['buildbotURL'] = "http://localhost:8010/"
```

Autobuilder2 configuration

- **In yocto-controller (again)**
  - Edit yocto-controller/yoctoabb/config.py

```
        workers_koan = ["example-worker"]
        workers = workers_koan
        all_workers = workers,
```

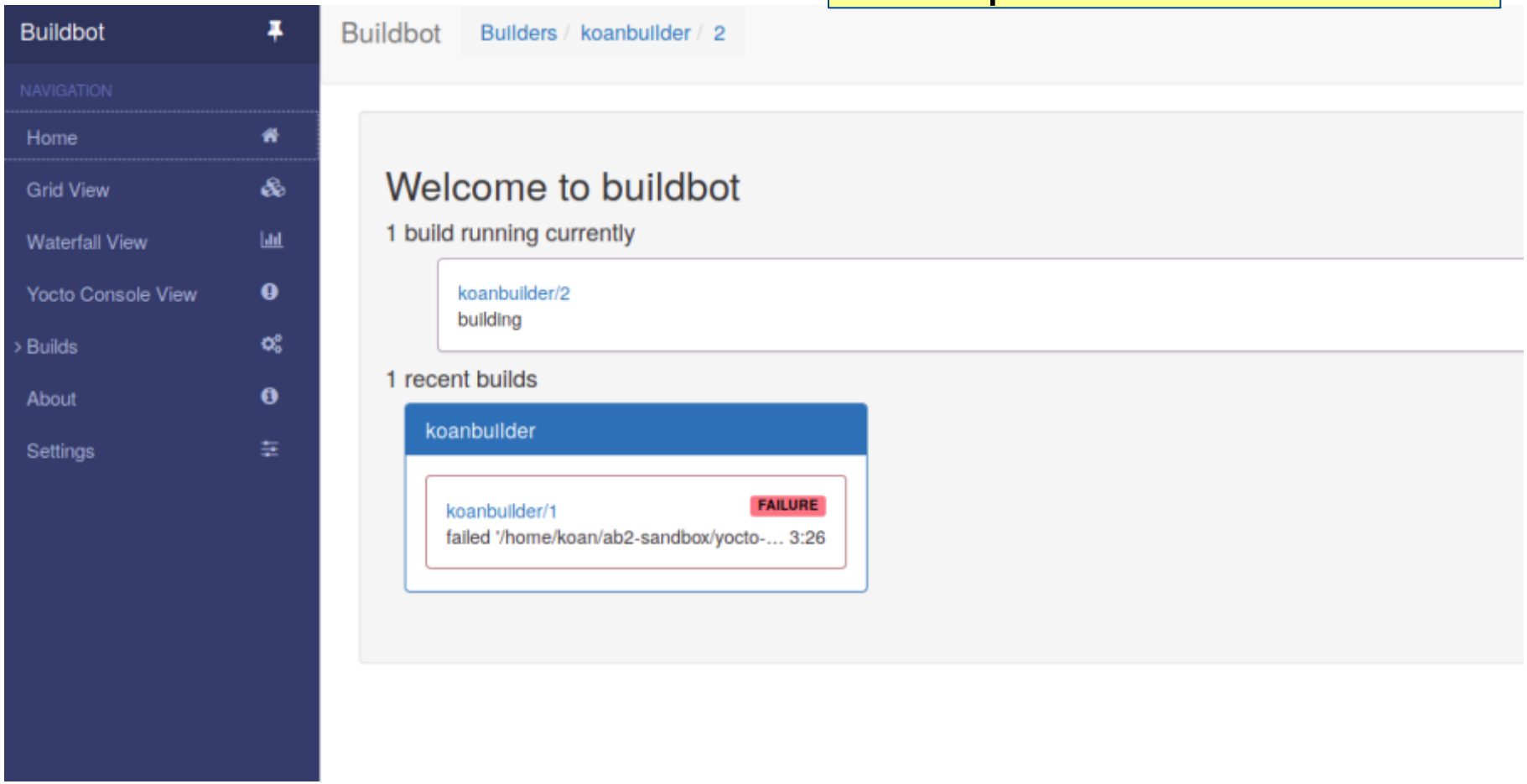
```
        sharedrepodir = "/home/koan/ab2-sandbox/repos"
        publish_dest = "/home/koan/ab2-sandbox/pub"
```

  - Specify the helper directory

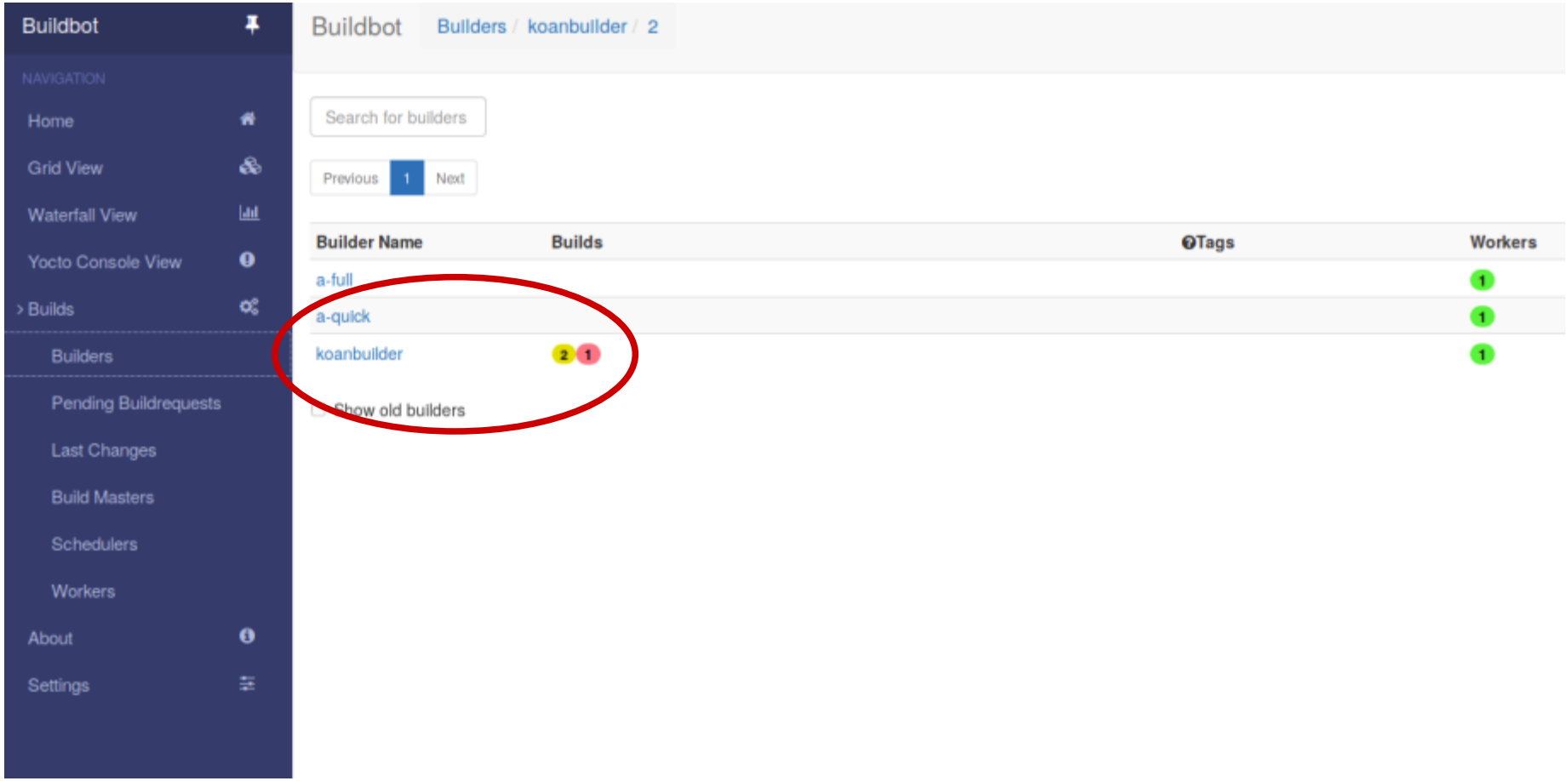
```
    repos = {
        "yocto-autobuilder-helper":
                ["file:///home/koan/ab2-sandbox/yocto-autobuilder-helper",
                 "master"],
```

Autobuilder2 ‘lite’ customized

http://localhost:8010/



Autobuilder2 ‘lite’ customized



Autobuilder2 to speed up Yocto build

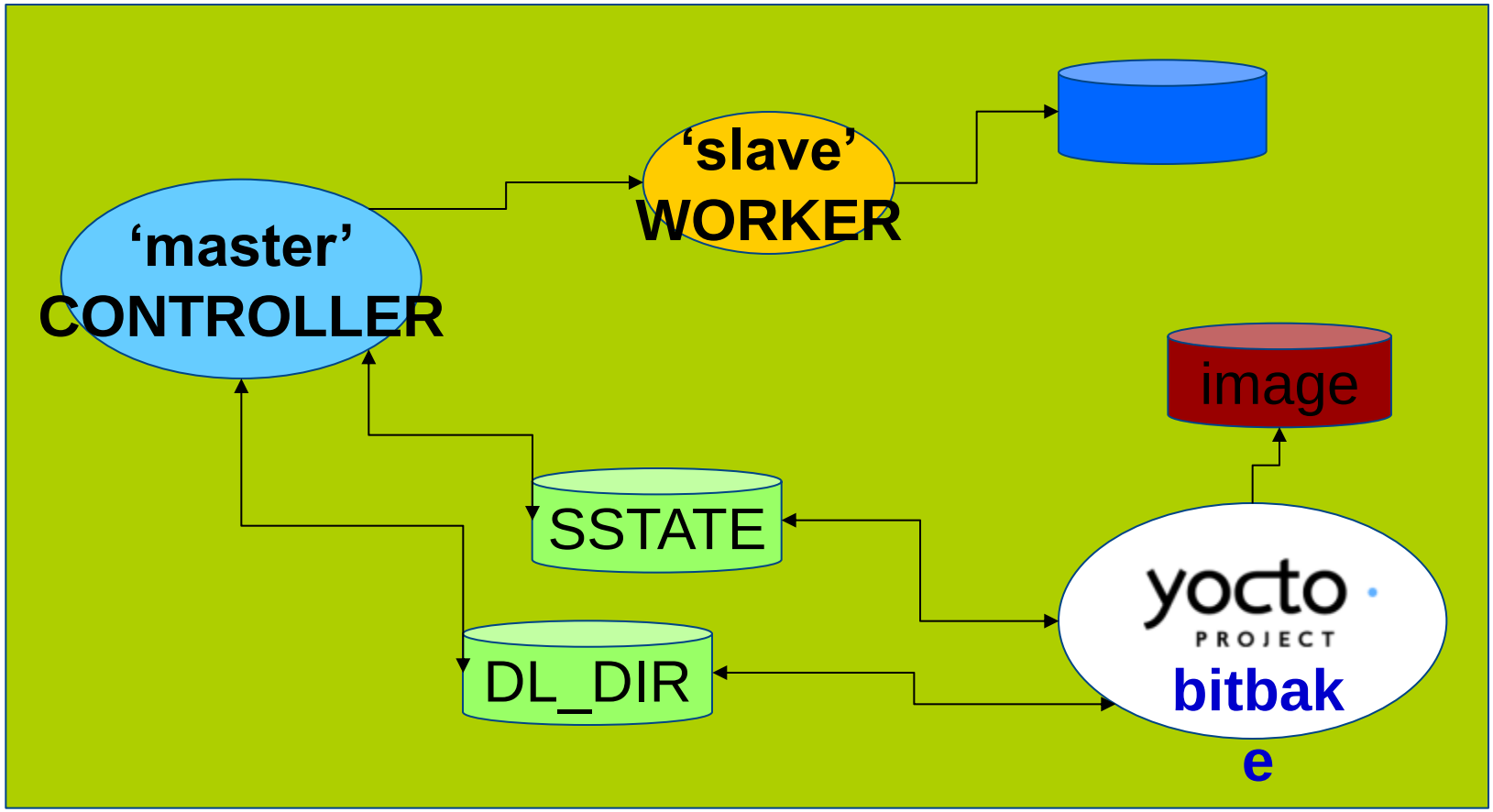
- **Share the artefacts with Yocto**
  - Edit local.conf
  - Share downloads

```
PREMIRRORS_prepend = "\
 git://.*/.* file:///home/koan/ab2-sandbox/autobuilder/current_sources/ \n \
 ftp://.*/.* file:///home/koan/ab2-sandbox/autobuilder/current_sources/ \n \
 http://.*/.* file:///home/koan/ab2-sandbox/autobuilder/current_sources/ \n \
 https://.*/.* file:///home/koan/ab2-sandbox/autobuilder/current_sources/ \n \
 "
```

  - Share SSTATE

```
SSTATE_MIRRORS = "file://.* \
  file:///home/koan/ab2-sandbox/autobuilder/pub/sstate/PATH"
```

Autobuilder2 to speed up Yocto build



Questions?

https://yoctoproject.org

https://koansoftware.com

2. Working with NVIDIA Tegra BSP and Supporting Latest CUDA Versions

Leon Anavi

Download the slides from here:

- **https://wiki.yoctoproject.org/wiki/File:Yocto-dev-summit-leon-anavi-2019.pdf**

3. Sstate-cache Magic!

Jaewon Lee

Presented by Mark Hatle

Abstract

From-scratch builds, even using server grade machines (with 40+ cores) will take just under an hour to complete. Additionally this estimate is just for minimal, stripped down images; Bigger images that bring up more than just core functionality and support things like web browsers/multimedia would take much longer (on the order of several hours).

Use of the sstate cache drastically cuts down on build times, especially for fresh projects. Xilinx makes full use of the sstate cache to speed up builds for its customers by hosting a comprehensive sstate cache (for all packages for different types of architectures) and allowing users to point their builds to this prebuilt and maintained sstate cache.

Abstract

There are different ways of distributing the sstate. When building an esdk (An extensible software development kit), the sstate of all non-native components is packaged so that any build using the esdk will happen in the blink of an eye. However, when building an sdk from within another sdk, the sstate for the native components were missing , hence making the sdk build disproportionately long compared to regular builds.

We introduced a patch into core that allows users to toggle the inclusion of nativesdk packages into the esdk by correctly handling sstate cache artifacts themselves as well as the corresponding signatures that are used to reference if anything has changed. With this change, a bigger esdk will be built, when required, that will skip rebuilding native components.

Agenda

Agenda

- What is sstate-cache and how is it used
- Tips and tricks
- How is sstate-cache used in Xilinx
- Upstreamed native sdk patch

What is sstate-cache and how is it used?

What is the sstate-cache?

- **All of this and more can be found in the sstate-cache section of the mega-manual**

  - https://www.yoctoproject.org/docs/latest/mega-manual/mega-manual.html#shared-state-cache

What is the sstate-cache?

- **The sstate-cache allows incremental builds**
  - Checksums are calculated on a per-task basis to minimize rebuilding unnecessarily
  - If the hash of any task is changed, task will be re-run
  - Configuration (local.conf, distro.conf, etc)
  - Recipe (.bb / .bbappend / dependency / function)
  - Files (src_uri)
  - If all inputs remained the same, build artifacts will be copied from the sstate-cache to the destination
  - This task as well all tasks this was dependent on, will be skipped

What is the sstate-cache?

- **To illustrate how useful the sstate-cache can be:**
  - **Build, no external sstate-cache: ~1.5 Hours**

    How long it takes to parse all of the files, execute all of the tasks.
  - **Re-build with no changes: ~10 sec**

    How long it takes to check all the hashsums and figure out there's nothing to do, parsing is already cached.
  - **Removing tmp/ and rerunning build: ~1.5 min**

    How long it takes to restore minimum necessary build artifacts from the sstate-cache, build image, etc.

    Note: this is approximately how long it would take customers to do a build if provided with a full sstate-cache, and no additional transfer time was required.

Sstate-cache details

- **Sstate-cache wiring**
  - The _setscene task is the final wiring needed that will check if the sstate-cache can be used to skip this task
  - Flags: sstate-inputdirs, sstate-outputdirs, sstate-plaindirs, sstate-lockfile, sstate-lockfile-shared, sstate-interceptfuncs, sstate-fixmedir

- **Example:**

  package.bbclass

```
SSTATETASKS += "do_package"
do_package[cleandirs] = "${PKGDEST} ${PKGDESTWORK}"
do_package[sstate-plaindirs] = "${PKGD} ${PKGDEST}
${PKGDESTWORK}"
do_package_setscene[dirs] = "${STAGING_DIR}"

python do_package_setscene () {
    sstate_setscene(d)
}
addtask do_package_setscene
```

Sstate task flags

- sstate-inputdirs – where function places it's output to be cached
- sstate-outputdirs – where the output sstate cache copies to
- sstate-plaindirs – use when input/output is the same
- sstate-lockfile, sstate-lockfile-shared – special locks
- sstate-interceptfuncs – post processing sstate funcs
- sstate-fixmedir – directory to scan "fixme" ops

Sstate-cache details (cont)

- **sstate-cache is stored under build/sstate-cache (default but can be changed)**
  - As an example, In the following:

sstate-cache/01/sstate:sstate:make::4.2.1:r0::3:01397ee06dba53ce572b63b9242fe29c_populate_lic.tgz

- **The build artifacts that would be copied into the cache, then placed into the outputdir:**

do_populate_lic[sstate-inputdirs] = ${LICSSTATEDIR}:

./license-destdir/make/generic_GPLv3
./license-destdir/make/recipeinfo
./license-destdir/make/COPYING
./license-destdir/make/generic_LGPLv2
./license-destdir/make/COPYING.LIB

do_populate_lic[sstate-outputdirs] = ${LICENSE_DIRECTORY}

Tips and Tricks

Tips and tricks

- **You can point to an external sstate-cache (either on a server or local host)**

```
SSTATE_MIRRORS ?= "\
    file://.* http://someserver.tld/share/sstate/PATH;downloadfilename=PATH \n \
    file://.* file:///some/local/dir/sstate/PATH"
```

  - Using SSTATE_MIRRORS is preferred (read only access)
  - Using a single shared directory (read write access)

- **If unfamiliar with mirroring, note the ‘\n’ and ‘\’!**

Tips and tricks

- **There are cases where a scratch build is preferred**
  - Often useful when debugging, or verifying deterministic builds
  - Clean the sstate-cache for individual packages by running: **bitbake $PN –c cleansstate**
  - To invalidate a specific task and rerun everything starting from that task (For ex. If you just want to recompile without rerunning the do_configure task)

    bitbake $PN –C compile (Note the capital –C)

    you will see: "*NOTE: Tainting hash to force rebuild of task*"
  - Since the hash has been tained, output is not shareable!

Tips and tricks

- **Check differences in sstate-cache**
  - bitbake-dumpsig “/path to .siginfo file”
  - Dump everything that makes up the inputs of the sstate-cache (all variables, dependencies, hashsums, etc)
  - For example:

    Variable TARGET_CXXFLAGS value is ${TARGET_CFLAGS}

    basehash: 033325ee84d07cd82674a6827c1ea4a7b398da10430f41cffcaa488c4d0b3947

    List of dependencies for variable EXTRA_OEMAKE is {'BUILD_CFLAGS', 'BUILD_CC’, 'BUILD_LDFLAGS', 'BUILD_CPP'}

    Hash for dependent task /workspaces/jaewon/CORE/poky/meta/recipes-kernel/linux/linux-yocto_5.2.bb:do_kernel_configcheck is 36dad4284470c4ca656cfc981630cecbe4afac71a26a26b3ca706381be4f92cc

Tips and tricks

- **Find out why something rebuilt**
  - bitbake-diffsigs “/path to first .siginfo file” “/path to second .siginfo file”
  - Example output after rebuilding with comment appended to compile task:

```
bitbake-diffsigs "sstate-cache/7e/sstate:linux-yocto:qemux86_64-poky-linux:5.2.17+gitAUTOINC+b867b78b50_255a750d28:r0:
qemux86_64:3:7efd8f22bd3248da406d6e78c6b4983a045f4bf3ffe97ba35aa3e0d8041ba5ea_compile.tgz.siginfo" "sstate-cache/9a/ss
tate:linux-yocto:qemux86_64-poky-linux:5.2.17+gitAUTOINC+b867b78b50_255a750d28:r0:qemux86_64:3:9a095d3900f91c570e4b7fe
024d1b7f2122c488acc2774bd6d83e4e0c516ffcc_compile.tgz.siginfo"
NOTE: Starting bitbake server...
basehash changed from 033325ee84d07cd82674a6827c1ea4a7b398da10430f41cffcaa488c4d0b3947 to a594806008e24ecb63ef68d55d53
a370ebd484c23131bb62ae199369669d22dd
Variable do_compile value changed:
@@ -1,5 +1,4 @@
     kernel_do_compile
-#TESTINGDIFFSIG
         for dtbf in ${KERNEL_DEVICETREE}; do
                 dtb=`normalize_dtb "$dtbf"`
                 oe_runmake $dtb
```

Tips and tricks

- **Ignore variables:**
  - Often there are variables in the hash that don't actually affect the output…
  - For example:

  BB_HASHBASE_WHITELIST_append = "TOPDIR"

  PACKAGE_ARCHS[vardepsexclude] = "MACHINE"

- **Manually add variables:**
  - Sometimes there are variables you want to trigger a rebuild.. Often used with vardepsexclude.
  - For example:

  PACKAGE_ARCHS[vardeps] = "MACHINE"

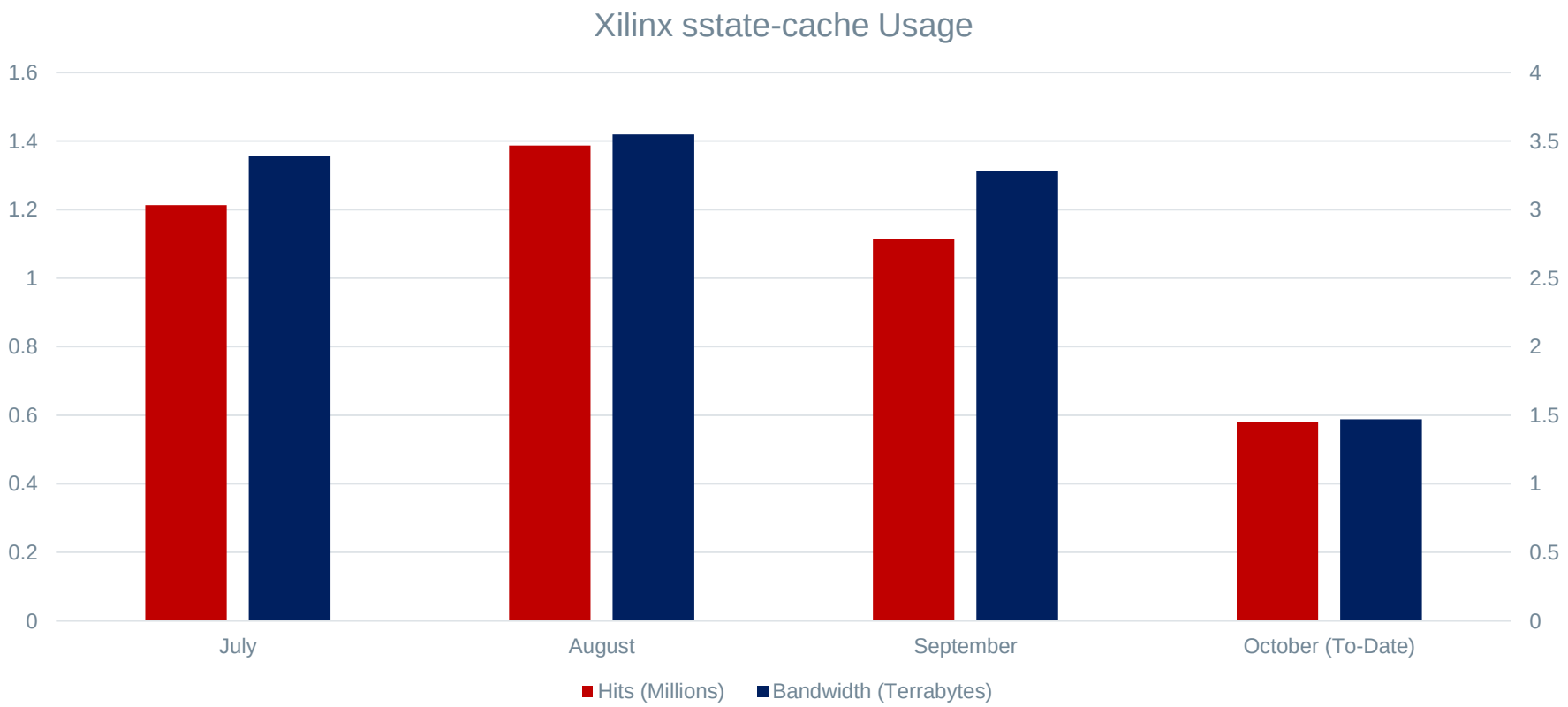
How is sstate-cache used at Xilinx

How sstate cache is used at Xilinx

- **A full image esdk (Extensible SDK) is built on a daily basis**
  - The sstate-cache packaged within this esdk is extracted and synced daily to an internal NFS mount point and tools internally points builds to this sstate-cache through the SSTATE_MIRRORS variable.
  - For every release, the final full sstate-cache is also synced http://petalinux.xilinx.com for external use

Sstate cache usage stats

External usage from petalinux.xilinx.com



Upstreamed native sdk patch

Include native sdk in esdk

- By default, native components (tools needed on the host) are not packaged into the esdk sstate-cache (originally because an esdk is targeted for a specific host).
- We introduced a mechanism to enable including native components so developers could make changes within an esdk, and package that up again to distribute by creating another derivative sdk, in a much shorter time.

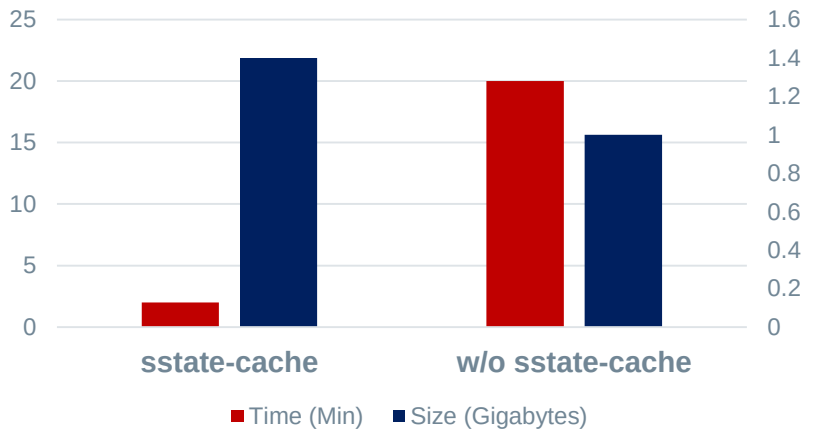
Include native sdk in esdk

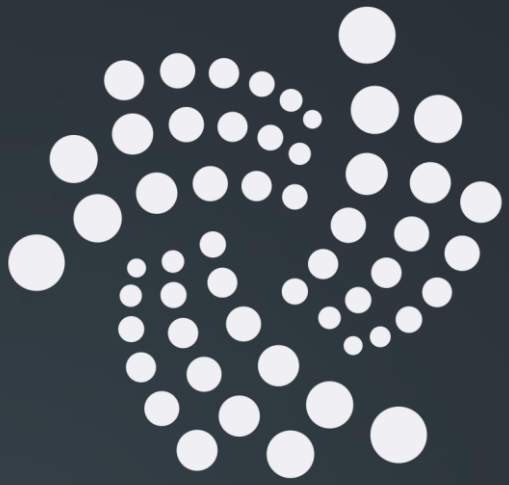
- **Users would need to enable the flag, i.e. SDK_INCLUDE_NATIVESDK = "1“**
  - And then build the esdk by running:

    bitbake $IMAGE_NAME –c populate_sdk_ext

Include native sdk in esdk

- **The result is a bigger esdk that enables a much quicker derivative sdk build**
  - Default:
  - Esdk size of **~1G**
  - Sdk build time of **~20 min**
  - By enabling SDK_INCLUDE_NATIVESDK
  - Esdk size of **~1.4G**
  - Sdk build time of **~2 min**





4. Bringing IOTA Distributed Ledger Technology (DLT) into Yocto/OpenEmbedded

Bernardo A. Rodrigues

Presenters

- Bernardo A. Rodrigues
- meta-iota Maintainer
- bernardoaraujor@gmail.com

- Philipp Blum
- Developer Advocate (IOTA Foundation)
- philipp.blum@iota.org

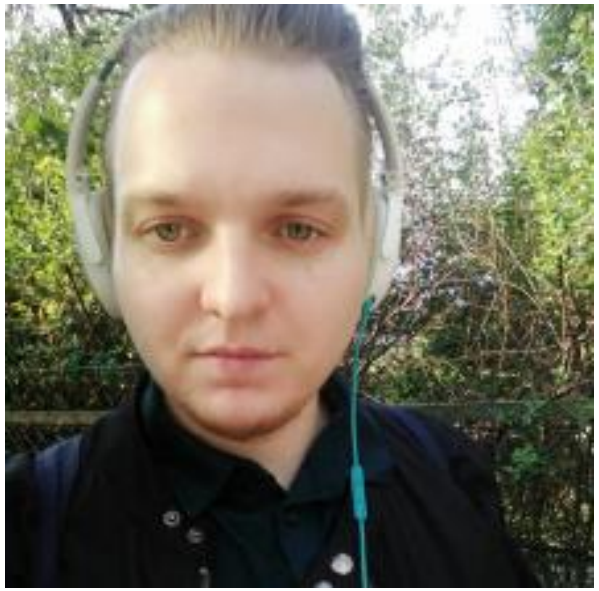
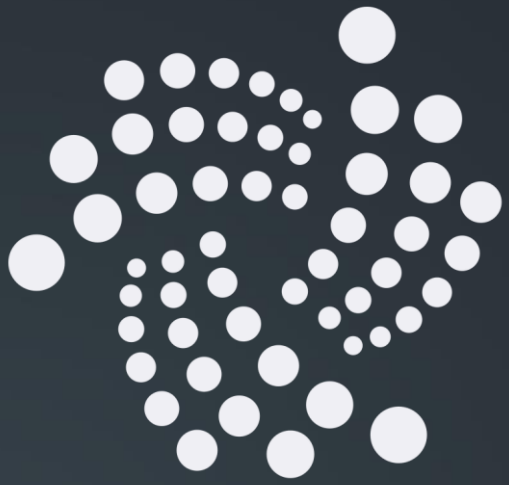


Table of Contents

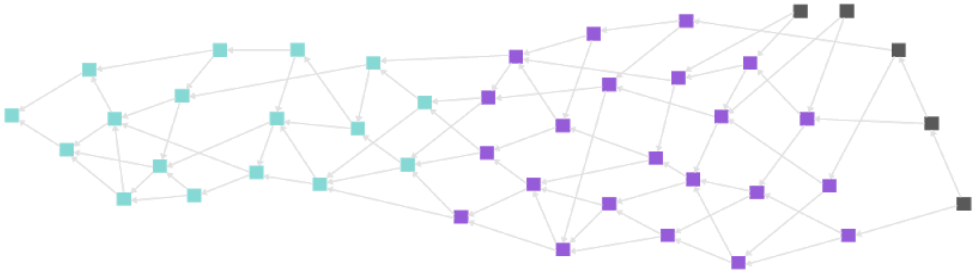


What is IOTA?

Context

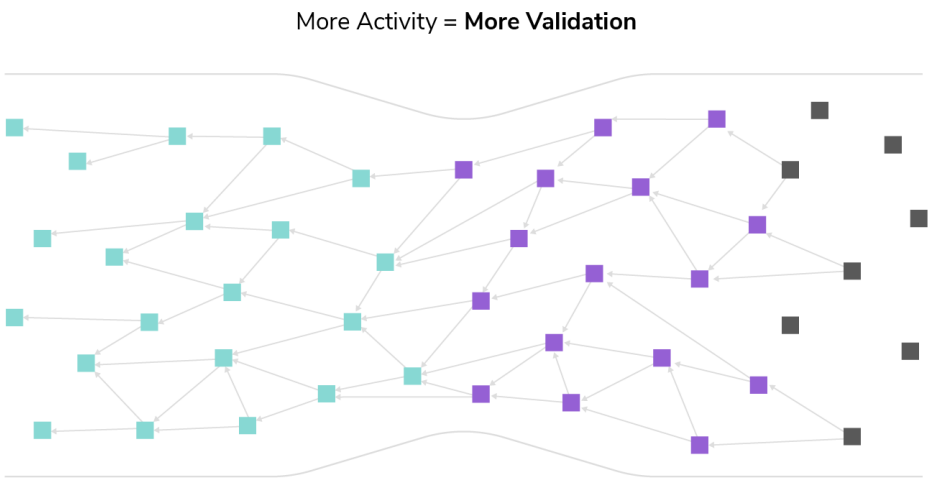
Distributed Ledger Technologies

Blockchain



Tangle
(DAG - Directed Acyclic Graph)

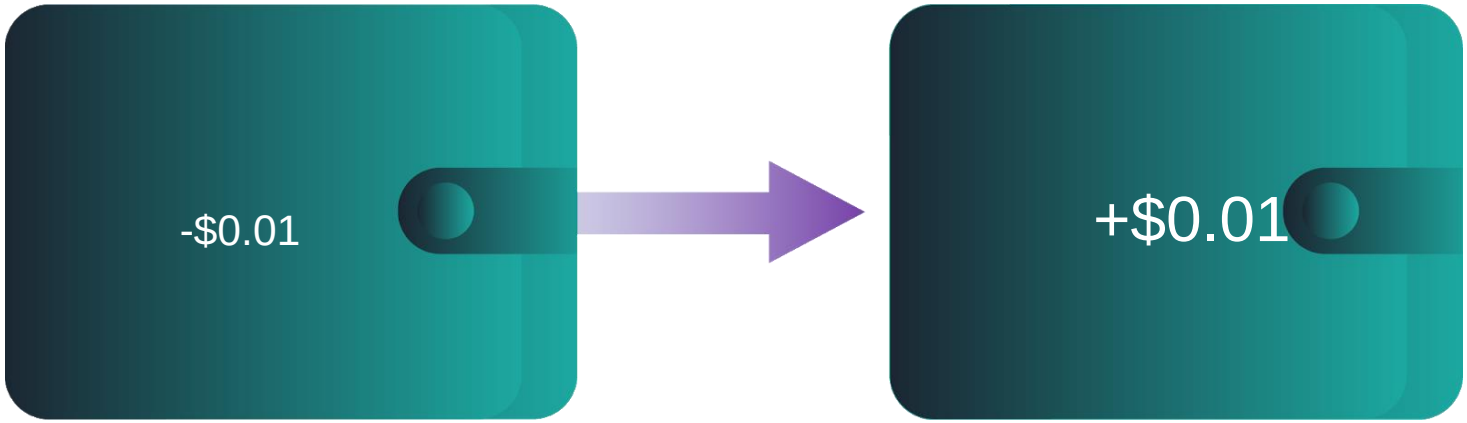
Tangle (DAG)



Each Vertex represents a transaction (squares)

Each Edge represents na approval (lines)

Zero Fee Transactions



No mining = No fees = Zero fee micro-transactions

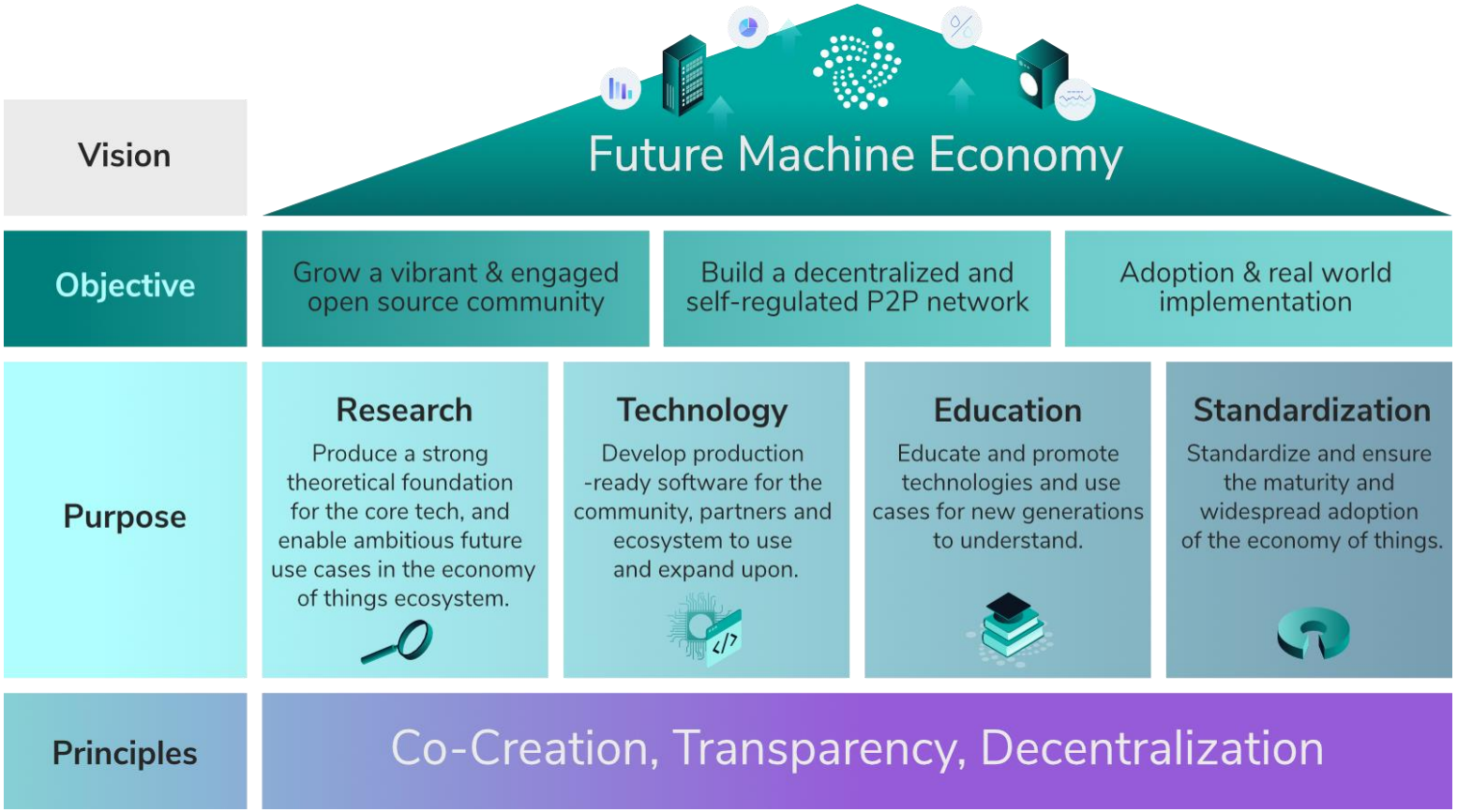
IOTA Foundation

IOTA FOUNDATION

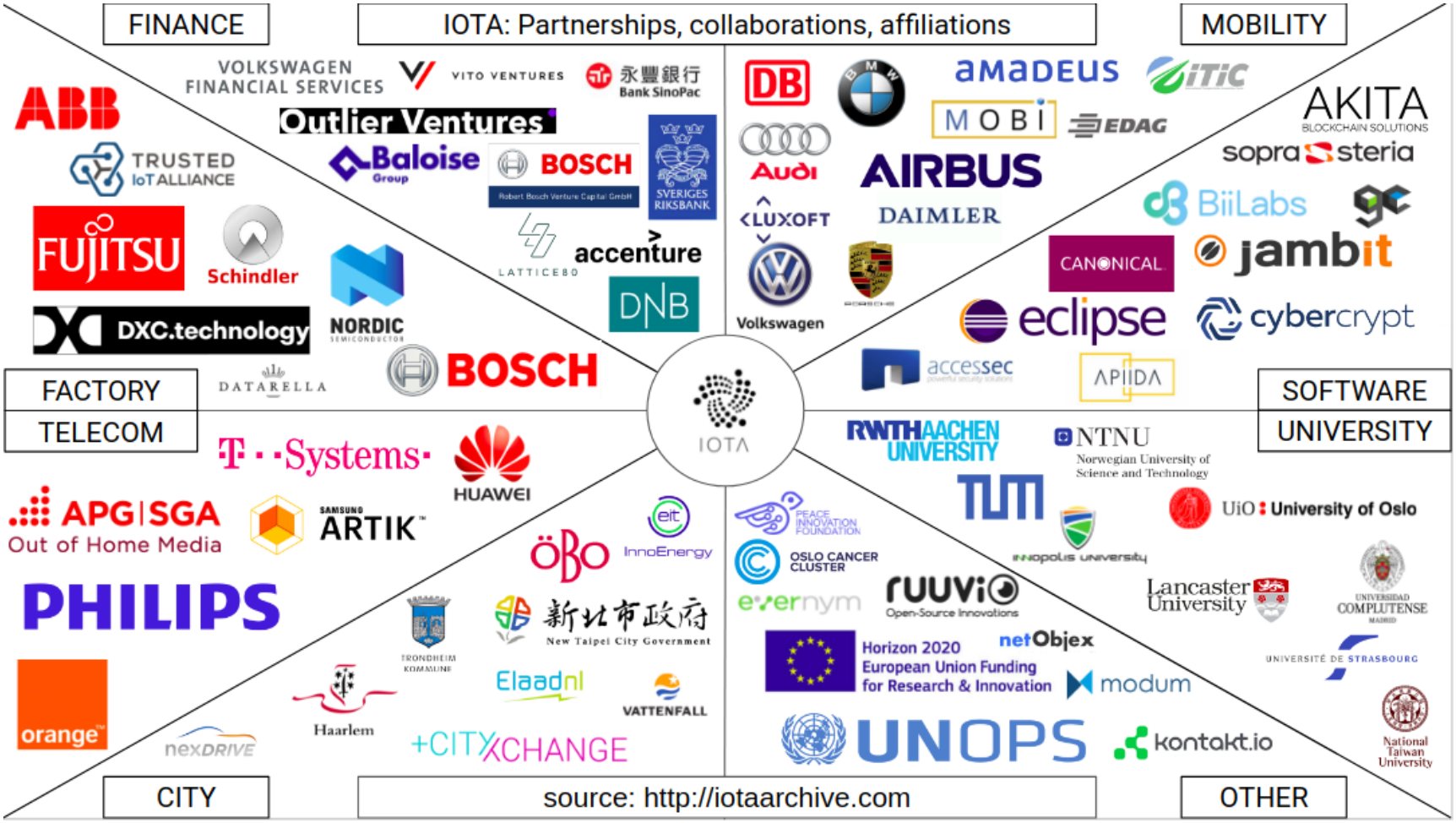
- **Non-Profit Foundation registered in Berlin**
- **~100 employees in 17 countries**
- **Funded through donations from IOTA Token holders, Research Grants and Project-based corporate financial support**

IOTA Foundation

IOTA FOUNDATION



IOTA Foundation: Collaborations & Partnerships



IOTA Nodes

Pre Coordicide vs Post Coordicide

IOTA Nodes

- DLT Node:
  - transaction relay
  - ledger copy

IOTA Nodes

- Ethereum, Bitcoin, etc: Nodes on the Cloud (↑ hw resources)
- IOTA: Nodes on the Edge (↓ hw resources)

https://blog.iota.org/towards-open-collaboration-1926e94514b8

Coordicide

- To make it possible for the network to grow and protect it against certain attacks, IOTA currently relies on a coordinator.
- The coordinator checkpoints valid transactions, which are then validated by the entire network.
- The coordinator is being run by the IOTA Foundation.
- Removing the Coordinator from the IOTA network will realize a long sought after goal in the field of DLT: scalability without centralization.
- Coordicide: the death of the Coordinator.

Pre Coordicide vs Post Coordicide

- Pre-Coordicide Node implementation:
  - IRI (Java)
  - cIRI (C)
- Coordicide Proof of Concept Node implementation:
  - GoShimmer (Go)
- Post-Coordicide Node implementation:
  - Bee (Rust)
  - Hornet (Go)

Since Coordicide is still a topic under R&D, meta-iota focuses on Pre Coordicide (for the moment).

meta-iota

Recipes

cIRI

- low level implementation of an IOTA node in C
- Users to become part of the IOTA network:
  - transaction relay
  - network information provider
- JSON-REST HTTP interface
- Suited for Embedded (SoC, SoM):
  - RAM: down to ~140MB RAM for solid node, ~500MB while syncing

cIRI: Bazel

- IF development team chose Bazel as build system for cIRI
- I borrowed the Bazel recipe and bbclass from meta-tensorflow
- Plans to switch to CMake

ciri_0.1.0.bb

- https://github.com/bernardoaraujor/meta-iota/blob/master/recipes-iota/ciri/ciri_0.1.0.bb

Let's ping the cIRI node on the BBB

```
$ curl http://104.155.135.221:14265/ \
        -X POST \
        -H 'Content-Type: application/json' \
        -H 'X-IOTA-API-Version: 1' \
        -d '{"command": "getNodeInfo"}'
```

CClient

- IOTA client library implementation in C.
- Recipe exports libcclient.a into the target rootfs/sysroot.
- CMake support
- Patch CMakeLists.txt to avoid the ExternalProject_add feature of CMake
- Recipe for c-iota-workshop repository as an example of how to integrate with libcclient

libcclient_1.0.0.bb

- https://github.com/bernardoaraujor/meta-iota/blob/master/recipes-iota/cclient/libcclient_1.0.0.bb
- https://github.com/bernardoaraujor/meta-iota/blob/master/recipes-iota/cclient/c-iota-workshop_git.bb

Playing around with c-iota-workshop

- Install Bazel:
  https://docs.bazel.build/versions/master/install.html
- Clone repo:
  ```
  $ git clone https://github.com/iota-community/c-iota-workshop
  ```
- Run an example:
  ```
  $ cd c-iota-workshop
  $ bazel run -c opt examples:[EXAMPLE_NAME]
  ```
- Following examples are available:

```
hello_world
send_hello
receive_hello
generate_address
check_balances
send_tokens
```

iota.go

- IOTA Go API Library allows:
  - Create transactions
  - Sign transactions
  - Interact with an IRI node
- Recipe written, although more testing is needed for validation.
- Recipe lists all golang package dependencies explicitly.
- Recipe for go-iota-workshop repository as an example of how to integrate with iota.go library

iota.go_1.0.0.bb

- https://github.com/bernardoaraujor/meta-iota/blob/go-dev/recipes-iota/iota.go/iota.go_1.0.0.bb
- https://github.com/bernardoaraujor/meta-iota/blob/go-dev/recipes-iota/iota.go/go-iota-workshop_git.bb

Playing around with go-iota-workshop

- Install Golang (1.10+)
  https://golang.org/doc/install
- Clone repo and download dependencies:
  ```
  $ git clone https://github.com/iota-community/go-iota-workshop
  $ cd go-iota-workshop; go mod download
  ```
- Run an example:
  ```
  $ go run iota_go_[EXAMPLE_NAME]/main.go
  ```
- Following examples are available:

```
helloworld
send_data
receive_data
create_address
check_balance
```

```
send_tx
receive_tx
zmq
```

iota.lib.py / PyOTA

- Official Python library for the IOTA Core.
- Implements both the official API, as well as signing, bundles, utilities and conversion.
- Python 3.6, 3.5 and 2.7.
- inherit setuptools
- Integration is planned for the near future
- https://github.com/iotaledger/iota.lib.py

Playing around with python-iota-workshop

- Install Python 3 and PIP
  https://www.python.org/downloads/
  https://pip.pypa.io/en/stable/installing/

- Clone repo and donwload dependencies:
  ```
  $ git clone https://github.com/iota-community/python-iota-workshop
  $ cd python-iota-workshop; pip install -r requirements.txt
  ```

- Run an example:
  ```
  $  python code/[EXAMPLE_NAME].py
  ```

- Following examples are available:
  ```
  e01_hello_world.py    e04_generate_address.py  e07_send_data.py
  e02_send_hello.py     e05_check_balance.py     e08_receive_data.py
  e03_receive_hello.py  e06_send_tokens.py       e09_zmq_listen.py
  ```

IOTA CLI App

- Command Line wallet and node management tool.
- It is implemented in nodejs, and it’s available as a npm package.
- To be integrated with the help of devtool npm functionality.
- Integration planned for the near future.
- https://github.com/iotaledger/cli-app
- https://wiki.yoctoproject.org/wiki/TipsAndTricks/NPM

recipes-support

- In order to fulfill dependencies, I had to write a few support recipes.
  - **nanopb_0.3.9.3.bb**: small code-size Protocol Buffers implementation in ansi C. Especially suitable for use in microcontrollers, but fits any memory restricted system
  - **keccak_git.bb**: keccak sponge function family including SHA3 implementation. Recipe needs improvement to support more architectures)
  - **logger_4.0.0.bb**: simple logging facility for the C language)
  - **libzmq_4.3.2.bb**: ZeroMQ core engine in C++
- Extra contribution to the OE community.

Future of meta-iota (2020-21)

- Bee
  - Post Coordicide Reference Implementation
  - Official IOTA Foundation
  - Rust (meta-rust and meta-rust-bin?)
- Hornet
  - Post Coordicide Implementation
  - Community based (EDF)
  - Go

Ecosystem Development Fund

Boards for Proof-of-Concept

IOTA Ecosystem Development Fund

- The IOTA EDF will allow me to validate Proof-of-Concepts on a few different boards with potential for IOTA Industrial applications.
- There is a big interest for FPGA projects in the IOTA Community. This is due to the Quorum Based computations, as well as accelerated Proof-of-Work (PoW), Address Generation and Signing.

Board	Manufacturer	Comment	OpenEmbedded BSP Layer
STM32MP157C-DK2	STMicroelectronics	The discovery SBC for STMicroelectronics STM32MP1 Series microprocessors	meta-st-stm32mp
Colibri iMX6 Solo SoM + Viola Carrier	Toradex	Toradex is a swiss manufacturer of Industrial-grade System on Modules.	meta-freescale-3rdparty
Zynq-7000 SoC ZC702 Evaluation Kit	Xilinx	The Zynq-7000 is a SoC+FPGA with great potential to accelerate PoW, Mini-PoW, Address Generation and Signing, as well as future Qubic implementations.	meta-xilinx

IOTA Ecosystem Development Fund

Board	Manufacturer	Comment	OpenEmbedded BSP Layer
DE10-Nano Development Kit	Terasic Technologies	The E10-Nano Development Kit is built around the Intel/Altera CycloneV SoC+FPGA. Also great potential to accelerate PoW, Mini-PoW, Address Generation and Signing, as well as future Qubic implementations.	meta-de10-nano meta-altera
BeagleBone Black	Texas Instruments	The most popular SBC in the Yocto Community.	meta-yocto-bsp meta-ti meta-beagleboard meta-bbb
DragonBoard 410c	96Boards	SBC with a Qualcomm Snapdragon 400	meta-qcom
Orange Pi Zero	Orange Pi	Popular small SBC with an AllWinner H2 chip.	meta-sunxi meta-allwinner-hx
Raspberry Pi Zero W	Raspberry Pi Foundation	Miniature version of the RPi, with Wireless support.	meta-raspberrypi

Class Account Setup

Yocto Project Dev Day Lab Setup

- **The virtual host's resources can be found here:**
  - Your Project: "/scratch/poky/build-qemux86_64“
  - Extensible-SDK Install: "/scratch/sdk/qemux86_64“
  - Sources: "/scratch/src“
  - Poky: "/scratch/poky"
  - Downloads: "/scratch/downloads"
  - Sstate-cache: "/scratch/sstate-cache“

- **You will be using SSH to communicate with your virtual server.**

FYI: How class project was prepared (1/2)

```
$
$ cd /scratch
$ git clone -b zeus git://git.yoctoproject.org/poky.git
$ cd poky
$
$ bash                                          # set up local shell
$ # Prepare the project
$ ./scratch/poky/oe-init-build-env  build
$ echo "SSTATE_DIR = \"/scratch/sstate-cache\"" >> conf/local.conf
$ echo "DL_DIR = \"/scratch/downloads\"" >> conf/local.conf
$ echo "IMAGE_INSTALL_append = \" gdbserver openssh libstdc++ \
    curl \"" >> conf/local.conf
$
$ # Build the project
$ bitbake core-image-base
$
```

FYI: How class project was prepared (2/2)

```
$ # Build the eSDK
$
$ bitbake core-image-base -c populate_sdk_ext
$ cd /scratch/poky/build/tmp/deploy/sdk/
$ ./poky-glibc-x86_64-core-image-base-qemux86_64-toolchain-ext-*.sh \
      -y -d /scratch/sdk/qemux86_64
$ exit                                          # return to clean shell
$
$
$ bash                                          # set up local shell
$ cd /scratch/sdk/qemux86_64
$ . /scratch/sdk/qemux86_64/environment-setup-qemux86_64-poky-linux-gnue
$ devtool modify virtual/kernel
$ exit                                          # return to clean shell
$
```

NOTE: Clean Shells!

- **We are going to do a lot of different exercises in different build projects, each with their own environments.**
- **To keep things sane, you should have a new clean shell for each exercise.**
- **There are two simple ways to do it:**
  1. Close your existing SSH connection and open a new one -- or –
  2. Do a “bash” before each exercise to get a new sub-shell, and “exit” at the end to remove it, in order to return to a pristine state.

Devtool: Part 1
Kernel recipes and menuconfig

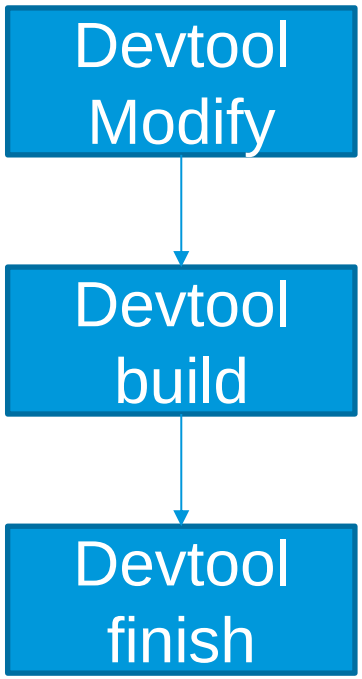
Manjukumar Harthikote Matha, Chandana Kalluri

Presented by Mark Hatle

Summary

- Current devtool flow
- Devtool flow for kernel
- Devtool menuconfig

Initial Devtool flow for kernel (a)



Fetch fresh copy of source from “SRC_URI”
(Takes about 5 mins to complete the command)

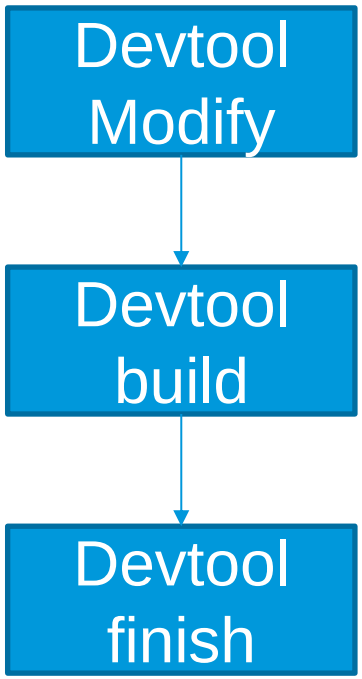
Make manual changes to .config file and save it

Compile and check for breakages

Save changes to the recipe

- The kernel source is fetched and copied to work-shared during the normal workflow either by running bitbake linux-yocto or bitbake <image-name>
- User runs devtool modify linux-yocto

Initial Devtool flow for kernel (b)



Fetch fresh copy of source from “SRC_URI”
(Takes about 5 mins to complete the command)

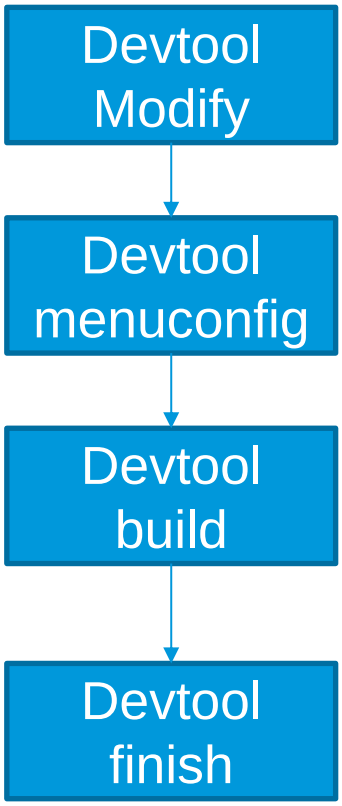
Make manual changes to .config file and save it

Compile and check for breakages

Save changes to the recipe

- The command will take about 5 mins because it will extract a new copy of source into workspace even though a copy of kernel source is present in a shared location from normal flow. Next it will configure the kernel.

New Devtool flow for kernel – case 1 (a)



Fetch fresh copy of source from “SRC_URI”
(Takes about 15 seconds)

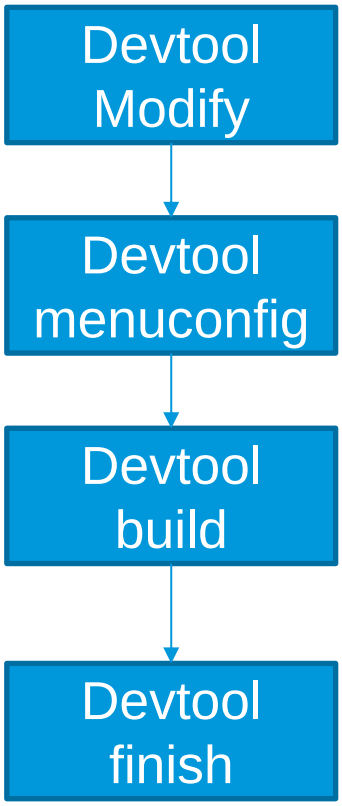
Use menuconfig GUI to make modifications to kernel configs

Compile and check for breakages

Save changes to the recipe

- The kernel source is fetched and copied to work-shared during the normal workflow either by running bitbake linux-yocto or bitbake <image-name>

New Devtool flow for kernel – case 1 (b)



Fetch fresh copy of source from “SRC_URI”
(Takes about 15 seconds)

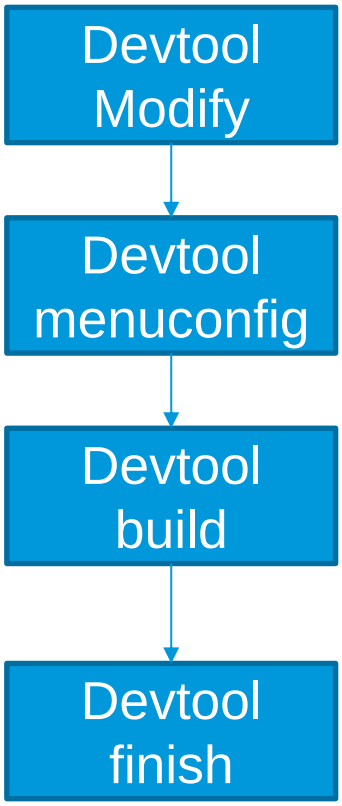
Use menuconfig GUI to make modifications to kernel configs

Compile and check for breakages

Save changes to the recipe

- User runs devtool modify linux-yocto
- The devtool modify command will take about 15 seconds because it will copy source from work-shared and configure the kernel.

New Devtool flow for kernel – case 2 (a)



Fetch fresh copy of source from “SRC_URI”
(Takes about 5 minutes)

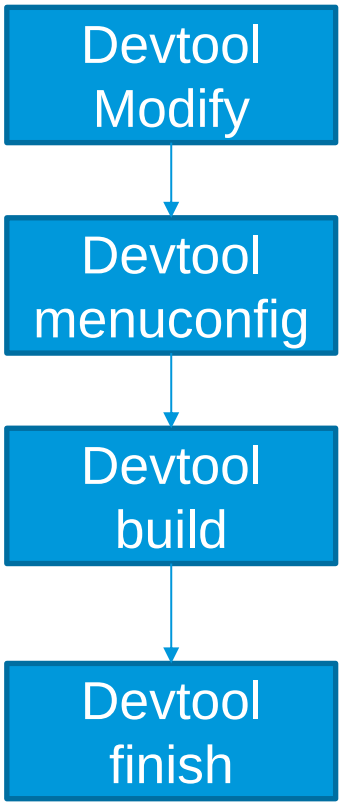
Use menuconfig GUI to make modifications to kernel configs

Compile and check for breakages

Save changes to the recipe

- The kernel source is not fetched and copied to work-shared during the normal workflow either by not running bitbake linux-yocto or bitbake <image-name>

New Devtool flow for kernel – case 2 (b)



Fetch fresh copy of source from “SRC_URI”
(Takes about 5 minutes)

Use menuconfig GUI to make modifications to kernel configs

Compile and check for breakages

Save changes to the recipe

- User runs devtool modify linux-yocto
- The devtool modify command will take about 5 mins because it will fetch new copy of source into workspace and place a copy in work-shared.

Commands

Original Devtool modify flow commands (pre 3.0)

```
$ source oe-init-build-env
$ bitbake linux-yocto <this takes about 5 mins>
$ devtool modify linux-yocto <this takes about 5 mins>
```

New Devtool modify flow commands – Case 1

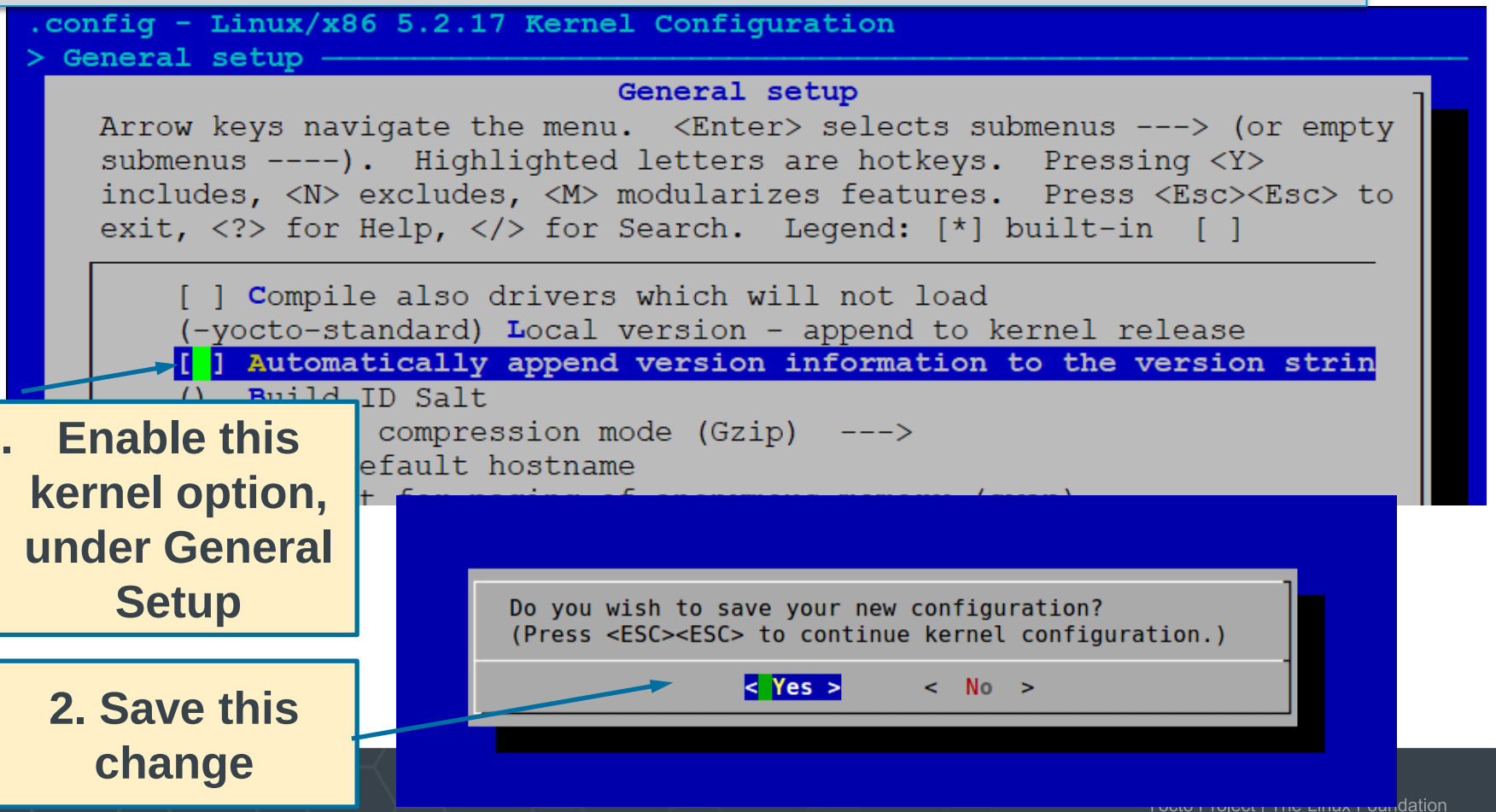
```
$ source oe-init-build-env
$ bitbake linux-yocto <this takes about 5 mins>
$ # Observe that the kernel souce was fetched:
$ ls tmp/work-shared/qemux86-64/kernel-source
$ devtool modify linux-yocto <this takes about 15 seconds>
```

New Devtool modify flow commands – Case 2

```
$ source oe-init-build-env
$ ls tmp/work-shared/qemux86-64/kernel-source  <kernel source not present>
$ devtool modify linux-yocto <this takes about 5 mins>
$ ls tmp/work-shared/qemux86-64/kernel-source  <kernel source copied >
```

Devtool menuconfig:

```
$ source oe-init-build-env
$ devtool modify linux-yocto
$ devtool menuconfig linux-yocto
```



1. Enable this kernel option, under General Setup

2. Save this change

Devtool menuconfig summary:

```
$ devtool menuconfig linux-yocto
…
Sstate summary: Wanted 0 Found 0 Missed 0 Current 43 (0% match, 100% complete)
NOTE: Executing Tasks
NOTE: Setscene tasks completed
Currently  1 running tasks (347 of 347)  99% |################################### |
0: linux-yocto-5.2.17+git999-r0 do_menuconfig - 0s (pid 31817)
NOTE: Tasks Summary: Attempted 347 tasks of which 339 didn't need to be rerun and all
succeeded.
INFO: Updating config fragment /scratch/poky/build/workspace/sources/linux-yocto/
oe-local-files/devtool-fragment.cfg
```

Review the change (config fragment) at the path specified above.

```
CONFIG_LOCALVERSION_AUTO=y
/build/workspace/sources/linux-yocto/oe-local-files/devtool-fragment.cfg (END)
```

Devtool finish

```
$ devtool finish linux-yocto meta-yocto-bsp

$cat ../meta-yocto-bsp/recipes-kernel/linux/linux-yocto_%.bbappend
FILESEXTRAPATHS_prepend := "${THISDIR}/${PN}:"

SRC_URI += "file://devtool-fragment.cfg"

$cat ../meta-yocto-bsp/recipes-kernel/linux/linux-yocto/devtool-fragment.cfg
CONFIG_LOCALVERSION_AUTO=y
```

Devtool: Part 2
Kernel Modules with eSDKs

Marco Cavallini

Kernel modules with eSDKs – Overview

- **The Extensible SDK (eSDK) is a portable and standalone development environment , basically an SDK with an added bitbake executive via devtool.**
- **The “devtool” is a collection of tools to help development, in particular user space development.**
- **We can use devtool to manage a new kernel module:**
  - Like normal applications is possible to import and create a wrapper recipe to manage the kernel module with eSDKs.

Kernel modules with eSDKs – Compiling a kernel module

- **We have two choices**
- **Out of the kernel tree**
  - When the code is in a different directory outside of the kernel source tree
- **Inside the kernel tree**
  - When the code is managed by a KConfig and a Makefile into a kernel directory

Kernel modules with eSDKs – Pro and Cons of a module outside the kernel tree

- **When the code is outside of the kernel source tree in a different directory**
- **Advantages**
  - Might be easier to handle modifications than modify it into the kernel itself
- **Drawbacks**
  - Not integrated to the kernel configuration/compilation process
  - Needs to be built separately
  - The driver cannot be built statically

Kernel modules with eSDKs – Pro and Cons of a module inside the kernel tree

- **When the code is inside the same directory tree of the kernel sources**
- **Advantages**
  - Well integrated into the kernel configuration and compilation process
  - The driver can be built statically if needed
- **Drawbacks**
  - Bigger kernel size
  - Slower boot time

Kernel modules with eSDKs – The source code

```
#include <linux/module.h>
#include <linux/kernel.h>

static int __init hello_init(void)
{
    printk("When half way through the journey of our life\n");
    return 0;
}

static void __exit hello_exit(void)
{
    printk("I found that I was in a gloomy wood\n");
}

module_init(hello_init);
module_exit(hello_exit);

MODULE_LICENSE("GPL");
MODULE_DESCRIPTION("Greeting module from the Divine Comedy");
MODULE_AUTHOR("Dante Alighieri");
```

Kernel modules with eSDKs – The Makefile

```
obj-m += hellokernel.o

SRC := $(shell pwd)

all:
	$(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules

modules_install:
	$(MAKE) -C $(KERNEL_SRC) M=$(SRC) modules_install
```

- ***KERNEL_SRC*** *is the location of the kernel sources.*
- *This variable is set to the value of the STAGING_KERNEL_DIR within the module class (module.bbclass)*

- *Sources avalable on https://github.com/koansoftware/simplest-kernel-module.git* and in **/scratch/src/kmod**

Kernel modules with eSDKs – Devtool setup

- **Start a new Shell!** Otherwise, the existing bitbake environment can cause unexpected results
- Here is how the eSDK was prepared for this class account:

< DO NOT ENTER THE FOLLOWING COMMANDS : ALREADY EXECUTED >

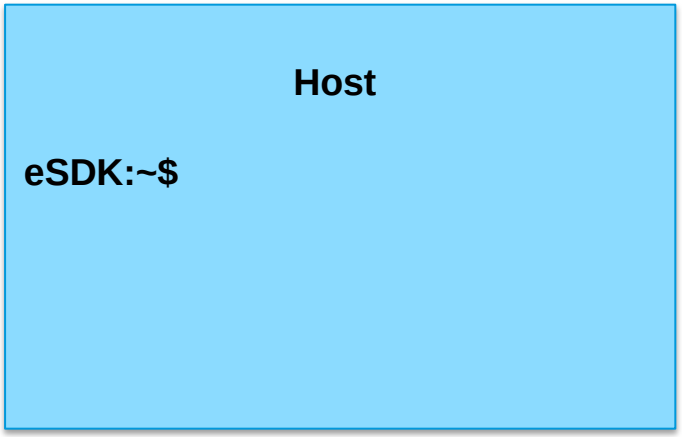
```
$ bitbake core-image-base -c populate_sdk_ext
$ cd /scratch/working/build/tmp/deploy/sdk/
$ ./poky-glibc-x86_64-core-image-base-core2-64-qemux86-64-toolchain-ext-3.0.sh \
        -d /scratch/sdk/qemux86_64 -y
$ cd /scratch/sdk/qemux86_64
$ . environment-setup-core2-64-poky-linux
$ devtool modify virtual/kernel
```

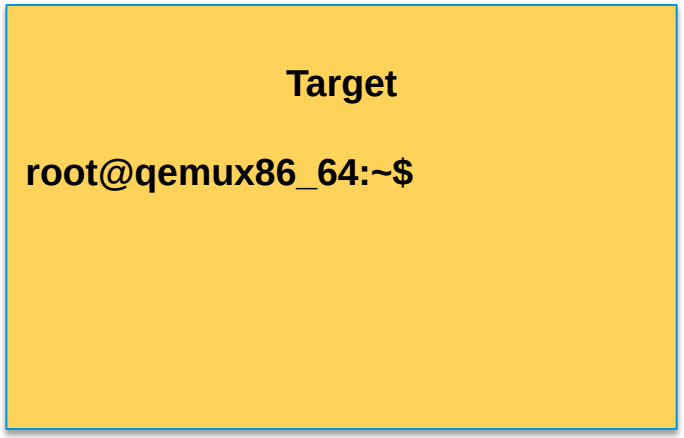
- **This installed the eSDK into:**

  /scratch/sdk/qemux86_64

Kernel modules with eSDKs – Overview

- Starting from now we are using the **eSDK** and not the project
- During this exercise we using two different machines
  - The **HOST** containing the eSDK (providing devtool)
  - The **TARGET** running the final qemux86_64 image





Kernel modules with eSDKs – Globalsetup

- Open two terminal windows and setup the eSDK environment in each one

```
$ cd /scratch/sdk/qemux86_64
$ bash                                    # safe shell
$ source environment-setup-core2-64-poky-linux
…
SDK environment now set up;
additionally you may now run devtool to perform development tasks.
Run devtool --help for further details.
$
```

Kernel modules with eSDKs – build the target image

- After you have setup the eSDK environment, build an image

```
$ devtool build-image
```

- **This will create a new image into:**

  /scratch/sdk/qemux86_64/tmp/deploy/images/qemux86_64

Kernel modules with eSDKs – build the target image

- Run the image to check if everything is OK
- This will run the QEMU machine in the TARGET shell you were using
- Login using user**: root** (no password required)

```
$ runqemu qemux86-64 nographic
```

Kernel modules with eSDKs – Hooking a new module into the build

- **Run the devtool to add a new recipe** (on the HOST side)

```
$ devtool add --version 1.0 simplestmodule \
      /scratch/src/kmod/simplest-kernel-module/
```

  - This generates a minimal recipe in the workspace layer
  - This adds EXTERNALSRC in an workspace/appends/simplestmodule_git.bbappend file that points to the sources
  - In other words, the source tree stays where it is, devtool just creates a wrapper recipe that points to it

- ***Note: this does not add your image to the original build engineer's image, which requires changing the platform project's conf/local.conf***

After the add

Workspace layer layout

```
$ tree /scratch/sdk/qemux86_64/workspace/

/scratch/sdk/qemux86_64/workspace/
├── appends
│   └── simplestmodule_git.bbappend
├── conf
│   └── layer.conf
├── README
└── recipes
    └── simplestmodule
        └── simplestmodule_git.bb
```

Kernel modules with eSDKs – Build the Module

- **Build the new recipe** (on the HOST side)

```
$ devtool build simplestmodule
```

*This will create the **simplestmodule.ko** kernel module*

This downloads the kernel sources (already downloaded for you):
linux-yocto-4.12.12+gitAUTOINC+eda4d18ce4_67b62d8d7b-r0 do_fetch

Kernel modules with eSDKs – Deploy the Module

- ***Get the target's IP address from the target serial console***

  root@qemux86_64:~# ifconfig

- **In the eSDK** (HOST) **shell, deploy the output**
  (*the target's ip address may change)*

```
$ devtool deploy-target -s simplestmodule root@192.168.7.2
```

- *NOTE: the '-s' option will note any ssh keygen issues, allowing you to (for example) remove/add this IP address to the known hosts table*

Kernel modules with eSDKs – Deploy Details

- **In the target** (qemux86_64)**, observe the result of deployment**

```
devtool_deploy.list                    100%   108        0.1KB/s  00:00
devtool_deploy.sh                      100%  1017        1.0KB/s  00:00
./
./lib/
./lib/modules/
./lib/modules/5.2.17-yocto-standard/
./lib/modules/5.2.17-yocto-standard/extra/
./lib/modules/5.2.17-yocto-standard/extra/hellokernel.ko
./usr/
./usr/include/
./usr/include/simplestmodule/
./usr/include/simplestmodule/Module.symvers
./etc/
./etc/modprobe.d/
./etc/modules-load.d/

NOTE: Successfully deployed
/scratch/sdk/qemux86_64/tmp/work/qemux86_64-poky-linux-gnueabi/simplestmodule/
```

Kernel modules with eSDKs – Load the Module

- **In the target (qemux86_64), load the module and observe the results**

```
root@qemux86_64:~# depmod -a


root@qemux86_64:~# modprobe hellokernel
[ 874.941880] hellokernel: loading out-of-tree module taints kernel.
[ 874.960165] When half way through the journey of our life


root@qemux86_64:~# lsmod
Module                  Size  Used by
hellokernel              929  0
nfsd                  271348  11
```

Kernel modules with eSDKs – Unload the Module

- **In the target** (qemux86_64)**, unload the module**

```
root@qemux86_64:~# modprobe -r hellokernel
[ 36.005902] I found that I was in a gloomy wood

root@qemux86_64:~# lsmod
Module                  Size  Used by
nfsd                  271348  11
```

Kernel modules with eSDKs – automatic load of the module at boot

- **In the target** (qemux86_64)**, edit the file below and add a new line containing the module name 'hellokernel'**

```
root@qemux86_64:~# vi /etc/modules-load.d/hello.conf

< insert the following line and save >

hellokernel
```

- **Then reboot the Qemu machine and verify**

```
root@qemux86_64:~# reboot
```

Questions

Devtool: Part 3
Bonus Kernel Lab

Tom Zanuss, Darren Hart, Saul Wold, Richard Griffiths, and YOU!

Bonus Kernel Lab!

- Here is you chance to learn more about kernel development support, plus help contribute tutorial content to Yocto Project!
- There is an important tradition of providing a Kernel Lab to help developers. The problem is that the last one was done for YP-2.6, and it needs an update to YP-3.*.
- Give the lab a try. If you find errors, let us know. If it is missing topic that would like addressed, or parts are unclear, let us know. If you can help provide fixes and improvements, then even better!

Bonus Kernel Lab

- The current document can be found here:
  - https://wiki.yoctoproject.org/wiki/File:Kernel-lab-2.6.pdf
  - https://wiki.yoctoproject.org/wiki/File:Kernel-lab-2.6.odt
- The sample source ZIP file can be found here:
  - TDB
- If you want to share your observations with us and others working on this, email myself and/or Tim and we will connect you:
  - david.reyna@windriver.com

6. User Space Topics

Rudi Streif

Presented by David Reyna

Overview

- Activity Setup
- Users, Groups and Passwords
- Login Shells
- Sudo Configuration
- SSH Server Configuration

Please ask questions.
Your questions might help others too.

Activity Setup

- **Create an activity layer and add it to the build environment**

```
$ cd /scratch/poky
$ source oe-init-build-env build-userspace
$ bitbake-layers create-layer meta-activity3
NOTE: Starting bitbake server…
Add your new layer with 'bitbake-layers add-layer meta-activity3'
$ bitbake-layers add-layer meta-activity3
NOTE: Starting bitbake server…
$ cat conf/bblayers.conf
…
BBLAYERS ?= " \
   /scratch/poky/meta \
   /scratch/poky/meta-poky \
   /scratch/poky/meta-yocto-bsp \
   /scratch/poky/build/meta-activity3 \
   "
```

Activity Setup

- Create an image recipe

```
$ mkdir -p meta-activity3/recipes-core/images
$ pushd meta-activity3/recipes-core/images
$ vi core-image-activity3.bb
```

```
SUMMARY = "Activity 3 Test Image"
DESCRIPTION = "Activity 3 Test Image for Yocto Project Summit"
LICENSE = "MIT"

IMAGE_INSTALL = "packagegroup-core-boot \
                 packagegroup-base-extended \
                 ${CORE_IMAGE_EXTRA_INSTALL} \
                 "

inherit core-image
```

```
$ bitbake core-image-activity3
$ runqemu qemux86-64 nographic
<…boot …>
<close QEMU with CTRL-A,X (typed fast)>
```

Users, Groups and Passwords

- The extrausers class provides a mechanism for managing users, groups and passwords.
- Available commands:
  - useradd
  - usermod
  - userdel
  - groupadd
  - groupmod
  - groupdel
- Commands are added to the EXTRA_USERS_PARAMS variable.
- Passwords must be provided in encrypted form.

Setting root user password and creating a user

```
$ vi core-image-activity3.bb
```

```
SUMMARY = "Activity 3 Test Image"
DESCRIPTION = "Activity 3 Test Image for Yocto Project Summit"
LICENSE = "MIT"

IMAGE_INSTALL = "packagegroup-core-boot \
                 packagegroup-base-extended \
                 ${CORE_IMAGE_EXTRA_INSTALL} \
                 "

inherit core-image
inherit extrausers

ROOT_PASSWORD = "secret"
DEV_PASSWORD = "hackme"

EXTRA_USERS_PARAMS = " \
   groupadd developers; \
   useradd -p `openssl passwd ${DEV_PASSWORD}` developer; \
   useradd -g developers developer; \
   usermod -p `openssl passwd ${ROOT_PASSWORD}` root; \
   "
```

```
$ bitbake core-image-activity3
$ runqemu qemux86-64 nographic
```

Works, but...

- **Changing the image recipe for new users is not really elegant.**
- **It would be better if we could set the users we want to add and their passwords in a configuration file such as local.conf or a distro configuration.**

A little script goes a long way...

```
$ vi user-setup.inc
```

```
# Image post-processing to setup user accounts

inherit extrausers

# Space-delimited list of user:password:<group,group,...> tuples
NEWUSERS ??= ""

# root password
ROOT_PASSWORD ??= ""

python () {
   params = ""

   # add new users
   newusers = (d.getVar("NEWUSERS", True) or "").split()
   if newusers:
      for user in newusers:
         name,password,groups = user.split(":")
         for group in groups.split(","):
            params += "groupadd -f " + group + "; "
         params += "useradd -p `openssl passwd " + password + "` "
         if groups:
            params += "-G " + groups + " "
         params += name + "; "

   # modify root password
   rootpw = d.getVar("ROOT_PASSWORD", True) or ""
   if rootpw:
      params += "usermod -p `openssl passwd " + rootpw + "` root; "

   d.setVar("EXTRA_USERS_PARAMS", params)
}
```

Using the script

```
$ vi core-image-activity3.bb
```

```
SUMMARY = "Activity 3 Test Image"
DESCRIPTION = "Activity 3 Test Image for Yocto Project Summit"
LICENSE = "MIT"

IMAGE_INSTALL = "packagegroup-core-boot \
                 packagegroup-base-extended \
                 ${CORE_IMAGE_EXTRA_INSTALL} \
                 "

inherit core-image

require user-setup.inc
```

```
$ vi /scratch/poky/build-userspace/conf/local.conf
```

```
# Users to be added: space-delimited list of name:password:groups tuples.
# groups is comma-delimited list of additional group names
NEWUSERS = "developer:hackme:developers"

# Root User Password
ROOT_PASSWORD = "secret"
```

```
$ bitbake core-image-activity3
$ runqemu qemux86-64 nographic
```

5. Devtool hands-on Seminar Part 2

Image Post Processing

- Sometimes it is necessary to processing such as adding, modifying files and more after the root file system has been created but before it is packaged into the different formats.
- Through the variable ROOTFS_POSTPROCESS_COMMAND you can specify a list of shell functions to be executed.
- Commonly the variable and the functions are added to the image recipe.
- The functions are executed in the order they appear in the variable.
- The search path for shell commands includes the native system root of the build environment and build host PATH from the user environment.
- The variable IMAGE_ROOTFS points to the directory where the build system assembles the root file system.

Setting Login Shells

```
$ vi core-image-activity3.bb
```

```
SUMMARY = "Activity 3 Test Image"
DESCRIPTION = "Activity 3 Test Image for Yocto Project Summit"
LICENSE = "MIT"

IMAGE_INSTALL = "packagegroup-core-boot \
                 packagegroup-base-extended \
                 ${CORE_IMAGE_EXTRA_INSTALL} \
                 "

inherit core-image

modify_shells() {
   printf "# BAR /etc/shells: valid login shells\n/bin/sh\n/bin/bash\n" \
          > ${IMAGE_ROOTFS}/etc/shells
}
ROOTFS_POSTPROCESS_COMMAND += "modify_shells;"
```

```
$ bitbake core-image-activity3
$ runqemu qemux86-64 nographic
```

Sudo Configuration

```
$ vi core-image-activity3.bb
```

```
SUMMARY = "Activity 3 Test Image"
DESCRIPTION = "Activity 3 Test Image for Yocto Project Summit"
LICENSE = "MIT"

IMAGE_INSTALL = "packagegroup-core-boot \
                 packagegroup-base-extended \
                 ${CORE_IMAGE_EXTRA_INSTALL} \
                 sudo \
                 "

inherit core-image

modify_sudoers() {
   sed 's/# %sudo/%sudo/' < ${IMAGE_ROOTFS}/etc/sudoers > \
       ${IMAGE_ROOTFS}/etc/sudoers.tmp
   mv ${IMAGE_ROOTFS}/etc/sudoers.tmp ${IMAGE_ROOTFS}/etc/sudoers
}
ROOTFS_POSTPROCESS_COMMAND += "modify_sudoers;"
```

```
$ bitbake core-image-activity3
$ runqemu qemux86-64 nographic
```

Note: You have to add a regular user to the sudo group for this to work.

SSH Server Configuration

```
$ vi core-image-activity3.bb
```

```
configure_sshd() {
   # disallow password authentication
   echo "PasswordAuthentication no" >> ${IMAGE_ROOTFS}/etc/ssh/sshd_config

   # create keys in tmp/deploy/keys
   mkdir -p ${DEPLOY_DIR}/keys
   if [ ! -f ${DEPLOY_DIR}/keys/root-sshkey ]; then
      /usr/bin/ssh-keygen -t rsa -N '' \
         -f ${DEPLOY_DIR}/keys/root-sshkey
   fi

   # add public key to authorized_keys for root
   mkdir -p ${IMAGE_ROOTFS}/home/root/.ssh
   cat ${DEPLOY_DIR}/keys/root-sshkey.pub \
       >> ${IMAGE_ROOTFS}/home/root/.ssh/authorized_keys
}
ROOTFS_POSTPROCESS_COMMAND += "configure_sshd;"
```

```
$ bitbake core-image-activity3
$ runqemu qemux86-64 nographic
[in a new ssh shell to your build system]
$ ssh -i \
  /scratch/poky/build-userspace/tmp/deploy/keys/root-sshkey \
  root@192.168.7.2
```

Nice, but once again not very flexible...

```
$ vi sshd-setup.inc
```

```
# Image post-processing to configure sshd

# Setup ssh key login for these users
SSH_USERS ??= ""

configure_sshd() {
   # disallow password authentication
   echo "PasswordAuthentication no" >> ${IMAGE_ROOTFS}/etc/ssh/sshd_config

   # keys will be stored tmp/deploy/keys
   mkdir -p ${DEPLOY_DIR}/keys

   # create the keys for the users
   for user in ${SSH_USERS}; do
      if [ ! -f ${DEPLOY_DIR}/keys/${user}-sshkey ]; then
      /usr/bin/ssh-keygen -t rsa -N '' \
         -f ${DEPLOY_DIR}/keys/${user}-sshkey
      fi

      # add public key to authorized_keys for the user
      mkdir -p ${IMAGE_ROOTFS}/home/${user}/.ssh
      cat ${DEPLOY_DIR}/keys/${user}-sshkey.pub \
          >> ${IMAGE_ROOTFS}/home/${user}/.ssh/authorized_keys
   done
}
ROOTFS_POSTPROCESS_COMMAND += "configure_sshd;"
```

Using the script

```
$ vi core-image-activity3.bb
```

```
SUMMARY = "Activity 3 Test Image"
DESCRIPTION = "Activity 3 Test Image for Yocto Project Summit "
LICENSE = "MIT"

IMAGE_INSTALL = "packagegroup-core-boot \
                 packagegroup-base-extended \
                 ${CORE_IMAGE_EXTRA_INSTALL} \
                 "

inherit core-image

require sshd-setup.inc
```

```
$ vi /scratch/poky/build-userspace/conf/local.conf
```

```
# Users for whom to create ssh login with key
SSH_USERS = "root developer"
```

```
$ bitbake core-image-activity3
$ runqemu qemux86-64 nographic

[in a new ssh shell to your build system]
$ ssh -i \
  /scratch/poky/build-userspace/tmp/deploy/keys/developer-sshkey \
  developer@192.168.7.2
```

EoA (End of Activity)

- **Cleanup**

```
$ cd /scratch/poky/build-userspace
$ bitbake-layers remove-layer meta-activity3
```

- **Thank You!**

Questions and Answers

Thank you for your participation!

Activity Eight

Tools, Toaster, User Experience

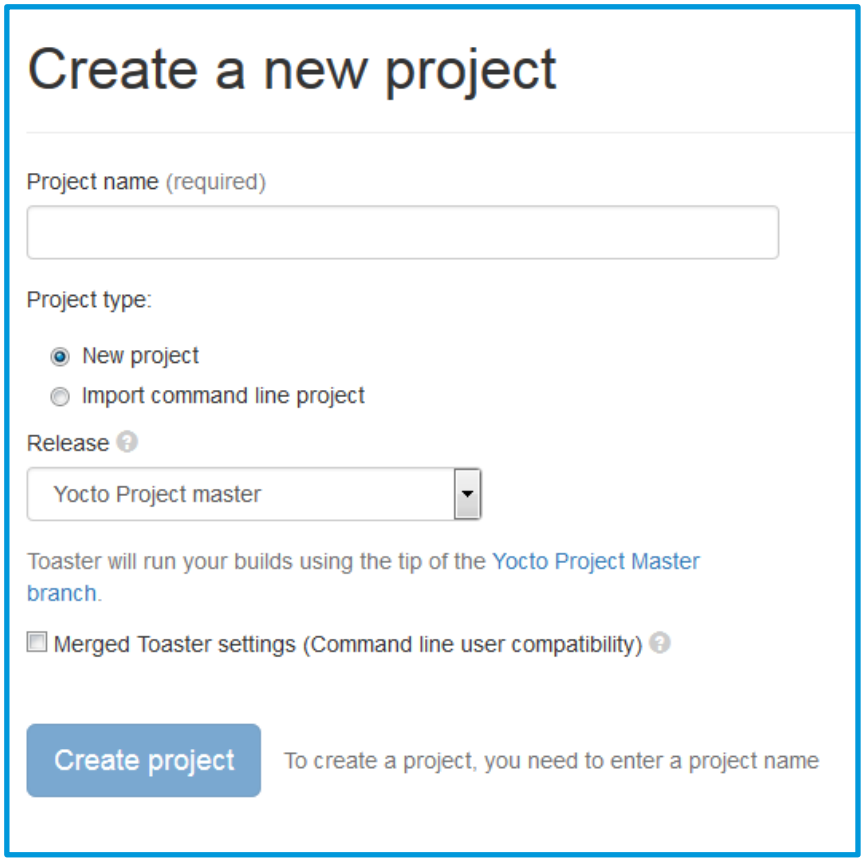
David Reyna

Toaster: Latest Features (1/2)

- **Toaster Documentation**
  - https://www.yoctoproject.org/docs/latest/toaster-manual/toaster-manual.html
- **Toaster Service Without a Web Server (“noweb”)**
  - Good for capturing command line build(s) directly into the db
- **Toaster Service Without Remote Builds (“nobuild”)**
  - Good for sharing build local status, without enabling external people creating projects and starting builds on your host
- **Toaster Service – Build Status within Containers**
  - New REST/JSON API to access the progress and health of bitbake builds via HTTP; very handy for containers
  - Build Status options: “Completed”, “In Progress”, “Specific Status”

Toaster: Latest Features (2/2)

- **Compatibility between Command Line and Toaster builds**
  - New “Import command line build” option
  - New “Merge Toaster Settings” into standard conf files”

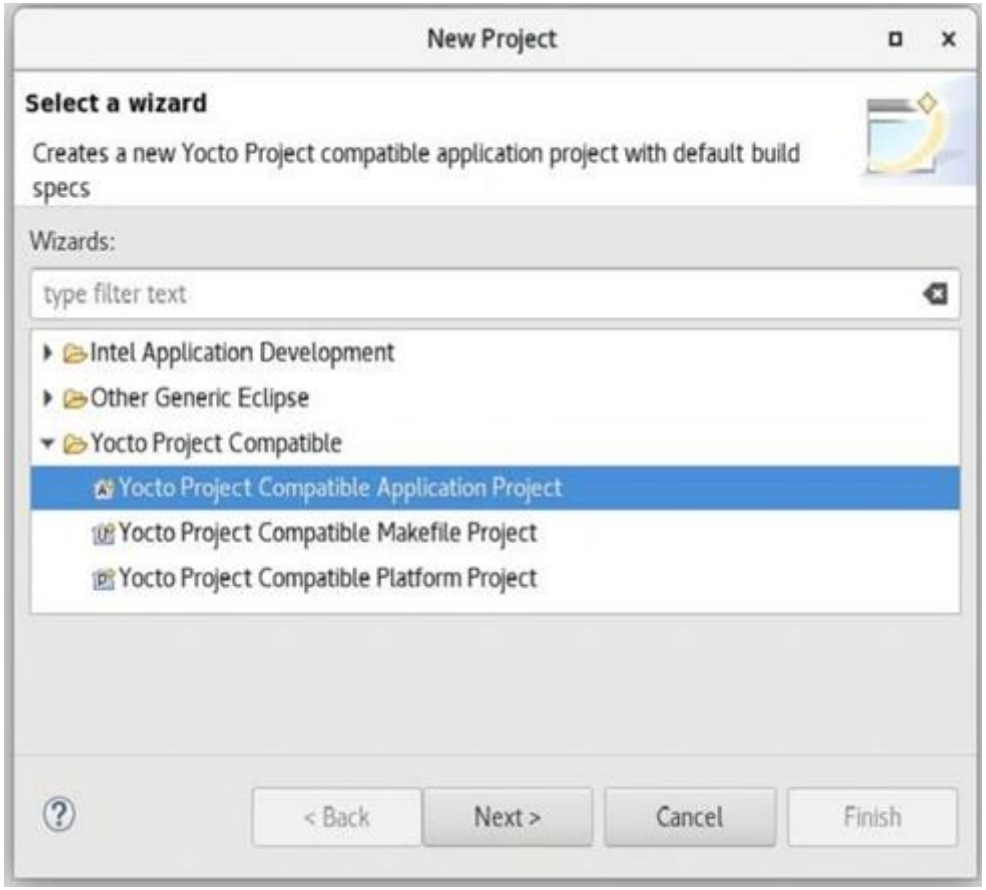


Intel System Studio 2019: Yocto Project Compatible

- **The Wind River Application and Project plug-ins have been shared with Intel System Studio, with the idea of open sourcing them to Eclipse.org**
- **Implementation is architecture agnostic**
- **Application Project Features:**
  - Awareness of YP compatible SDKs/eSDKs
  - Ability to register multiple SDKs
  - Automatic generation of “Build Specs” for each machine variant in each SDK
  - Ability to enable/disable debug flags
  - Debugger deploy and access over GDB/TCF
  - Set of sample applications

Intel System Studio 2019: Yocto Project Compatible

- **Platform Project Features:**
  - Configuration/Updates via Toaster
  - Basic build targets directly from ISS
  - Eclipse-based Kernel Configuration Tool
  - Tree view to browse deploy artifacts



Intel System Studio 2019: Yocto Project Compatible

- **Import:**
  - Existing command line project
  - Existing SDK/eSDK