AGL_test_FW_guider

Ver 0.0.1

2020/4/13

Reviewer	Executor
Li Xiaoming 2020/4/13	Qiu Tingting, Zhong Lu 2020/4/13

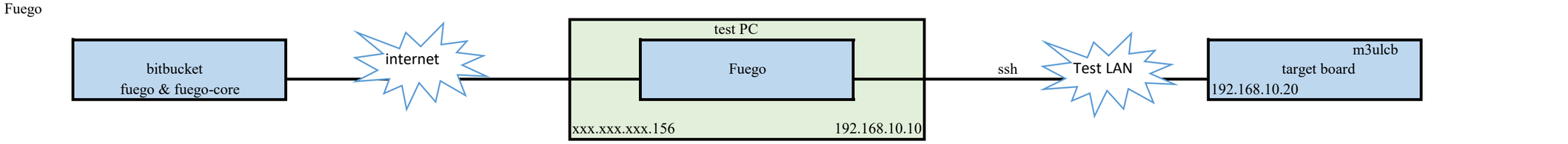
Revision History			
Version	**Updated Date**	**Contents**	**Remarks**
Ver 0.0.1	2020/4/13	Init version of AGL TEST FW GUIDER	

Construction of test framework

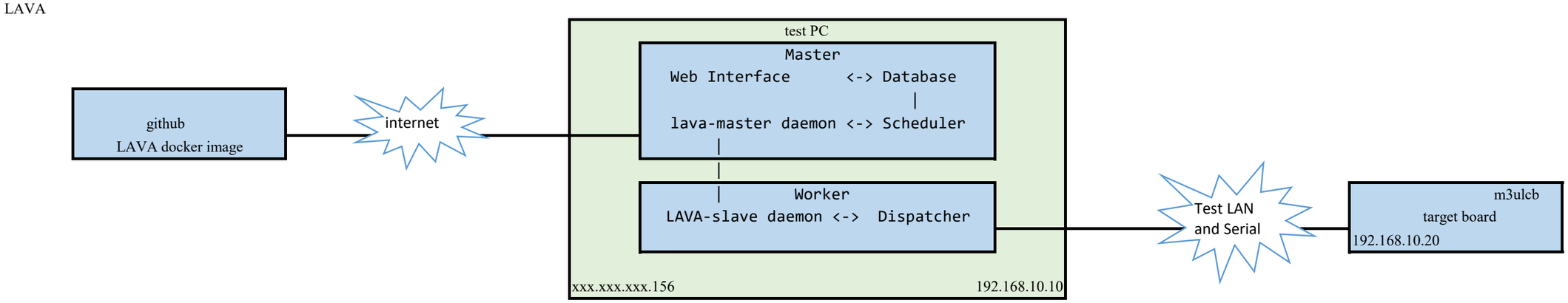
■ Contents

HardWare

Fuego



LAVA



Construction of Fuego

1. introduction of Fuego

Fuego is the official automated test framework for the LTSI project.
The project's web site is at: http://fuegotest.org/

2. prepare for Fuego

To retrieve the Fuego software and create the docker image for it, you need to have git and docker installed on your system.

```
$ sudo apt-get install git
$ sudo apt-get install docker.io
```

3. Installing Fuego to a docker container

Code for the test framework is available in 2 git repositories:

fuego
fuego-core

3-1. download the repository of Fuego v1.5.0 from bitbucket

```
$ git clone https://bitbucket.org/fuegotest/fuego.git
$ cd fuego
```

3-2. Run install.sh from the top directory.

This launches the ``docker build'' command to create a docker image - named 'fuego' by default.

```
$ ./install.sh
```

It also creates a container from this image, called 'fuego-container'. This container's port is 8090.
You can change the name of the docker image and the port used by Jenkins by passing the corresponding parameters to install.sh.

```
$ ./install.sh fuego-8092 8092
```

3-3. Update "fuego-core"

The repository fuego-core is mounted as a external docker volume.
The fuego-core repository should be auto installed inside the fuego directory, at the top level of that repository's directory structure (parallel to fuego-ro and fuego-rw).

```
root@debian-test:/work2/work/fuego/fuego# ls
CHANGELOG  Dockerfile            Dockerfile.test  frontend-install  fuego-host-scripts  fuego-rw        install.sh  LICENSE  start.sh
CREDITS    Dockerfile.nojenkins  docs             fuego-core        fuego-ro            install-debian.sh  jenkins     README   VERSION
```

4. Start "fuego-container"

To start your Fuego container, issue the following command top directory:

```
$ ./start.sh fuego-container
```

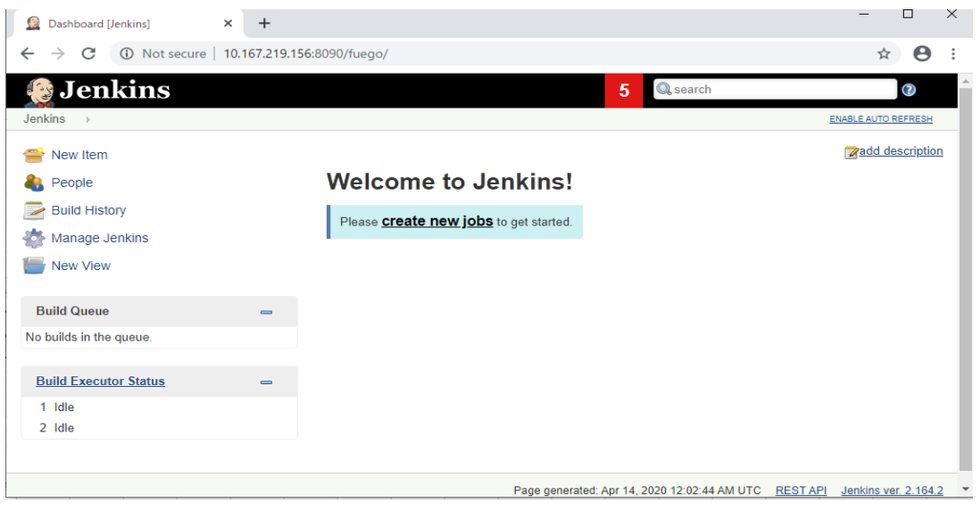
(or, use the name of whatever container was created in the install step. For example, ./start.sh fuego-8092-container).
Jenkins and network are automatically started in docker container.

```
[ ok ] Starting Jenkins Automation Server: jenkins.
[ ok ] Starting network benchmark server.
```

5. Access the Fuego Jenkins web interface

```
$ firefox http://localhost:8090/fuego
```

or



6. Add a board

6-1. Set up communication to the target board

In order for Fuego to test a board, it needs to communicate with it from the host machine where Fuego is running.
The most common way to do this is to use 'ssh' access over a network connection.
The target board needs to run an ssh server, and the host machine connects to it using the 'ssh' client.
For example:

6-2. Decide on user account for testing (creating one if needed)

On your target board, a user account is required in order to run tests.
For example:

use the root account

6-3. Create test directory on target

First, log in to your target board, and create a directory where tests can be run.
Usually, you do this as root, and a commonly used directory for this is "/home/fuego".
For example:

```
$ ssh root@your_target
<target>$ mkdir /home/fuego
<target>$ exit
```

6-4. Create board file

Create board file by copy an existing one.

```
$ cd fuego-ro/boards
$ cp template-dev.board m3ulcb.board
$ vi m3ulcb.board
```

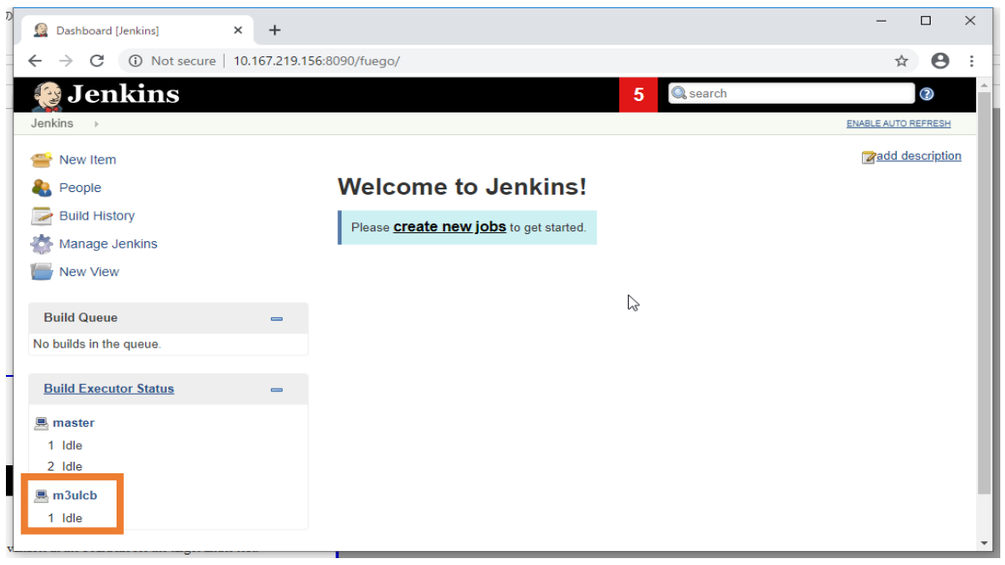
A board file has parameters which define how Fuego interacts with your board.
For example:

```
IPADDR="192.168.10.20"          <- ip of target board(See 6-1)
LOGIN="root"                    <- user account for testing(See 6-2)
PASSWORD=""                     <- password of user account for testing(See 6-2)
BOARD_TESTDIR="/home/fuego"             <- test directory on target(See 6-3)
TOOLCHAIN="m3ulcb"              <- toolchain of target(See 7-3)
TRANSPORT="ssh"
ARCHITECTURE="arm64"            <- the architcture of target(See 6-1)
```

6-5. Add node to Jenkins interface

In the Jenkins interface, boards are referred to as "Nodes".

```
$ ftc add-nodes -b m3ulcb
```



7. Add a toochain for board

You can skip this step when there is no cross-compiling in testsuites.

7-1. Install the SDK in the docker container

To allow fuego to use the SDK, you need to install it into the fuego docker container.

For example:

Copy m3ulcb's SDK into docker container.

SDK: /poky-agl-glibc-x86_64-agl-demo-platform-crosssdk-aarch64-toolchain-9.0.1.sh

Compile m3ulcb's SDK: AGL Image and SDK

Install SDK:

```
$ sudo docker exec -it fuego-container bash
root@ubuntu-ESPRIMO-P720:/# ls /fuego-rw/poky-agl-glibc-x86_64-agl-demo-platform-crosssdk-aarch64-toolchain-9.0.1.sh
/fuego-rw/poky-agl-glibc-x86_64-agl-demo-platform-crosssdk-aarch64-toolchain-9.0.1.sh
root@ubuntu-ESPRIMO-P720:/# sh /fuego-rw/poky-agl-glibc-x86_64-agl-demo-platform-crosssdk-aarch64-toolchain-9.0.1.sh
/bin/sh: warning: setlocale: LC_ALL: cannot change locale (en_US.UTF-8)
Automotive Grade Linux SDK installer version 9.0.1
==================================================
Enter target directory for SDK (default: /opt/agl-sdk/9.0.1-aarch64):
You are about to install the SDK to "/opt/agl-sdk/9.0.1-aarch64". Proceed[Y/n]? Y
Extracting SDK.....................................................................................................done
Setting it up.../bin/sh: warning: setlocale: LC_ALL: cannot change locale (en_US.UTF-8)
done
/bin/sh: warning: setlocale: LC_ALL: cannot change locale (en_US.UTF-8)
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script e.g.
 $ . /opt/agl-sdk/9.0.1-aarch64/environment-setup-aarch64-agl-linux
root@ubuntu-ESPRIMO-P720:/# ls /opt/agl-sdk/9.0.1-aarch64/
environment-setup-aarch64-agl-linux  site-config-aarch64-agl-linux  sysroots  version-aarch64-agl-linux
```

7-2. Create a '*-tools.sh' file for the toolchain

Now, fuego needs to be told how to interact with the toolchain.

During test execution, the fuego system determines what toolchain to use based on the value of the TOOLCHAIN variable in the board file for the target under test.

The TOOLCHAIN variable is a string that is used to select the appropriate '<TOOLCHAIN>-tools.sh' file in /fuego-ro/toolchains.

For example:m3ulcb-tools.sh

```
$ cd fuego-ro/toolchains
$ vi m3ulcb-tools.sh
```

Inside the -tools.sh file, you execute instructions that will set the environment variables needed to build software with that SDK.

```
SDKROOT=/opt/agl-sdk/9.0.1-aarch64/sysroots/aarch64-agl-linux      <-  the dir in docker container
ORIG_PATH=$PATH
PREFIX=aarch64-agl-linux
source /opt/agl-sdk/9.0.1-aarch64/environment-setup-aarch64-agl-linux                <-  the file in docker container
HOST=aarch64-agl-linux
```

8. Add test jobs

There are two ways of adding test jobs, individually, and using testplans. In both cases, you use the 'ftc add-jobs' command.

the 'ftc' command should be run in docker container.

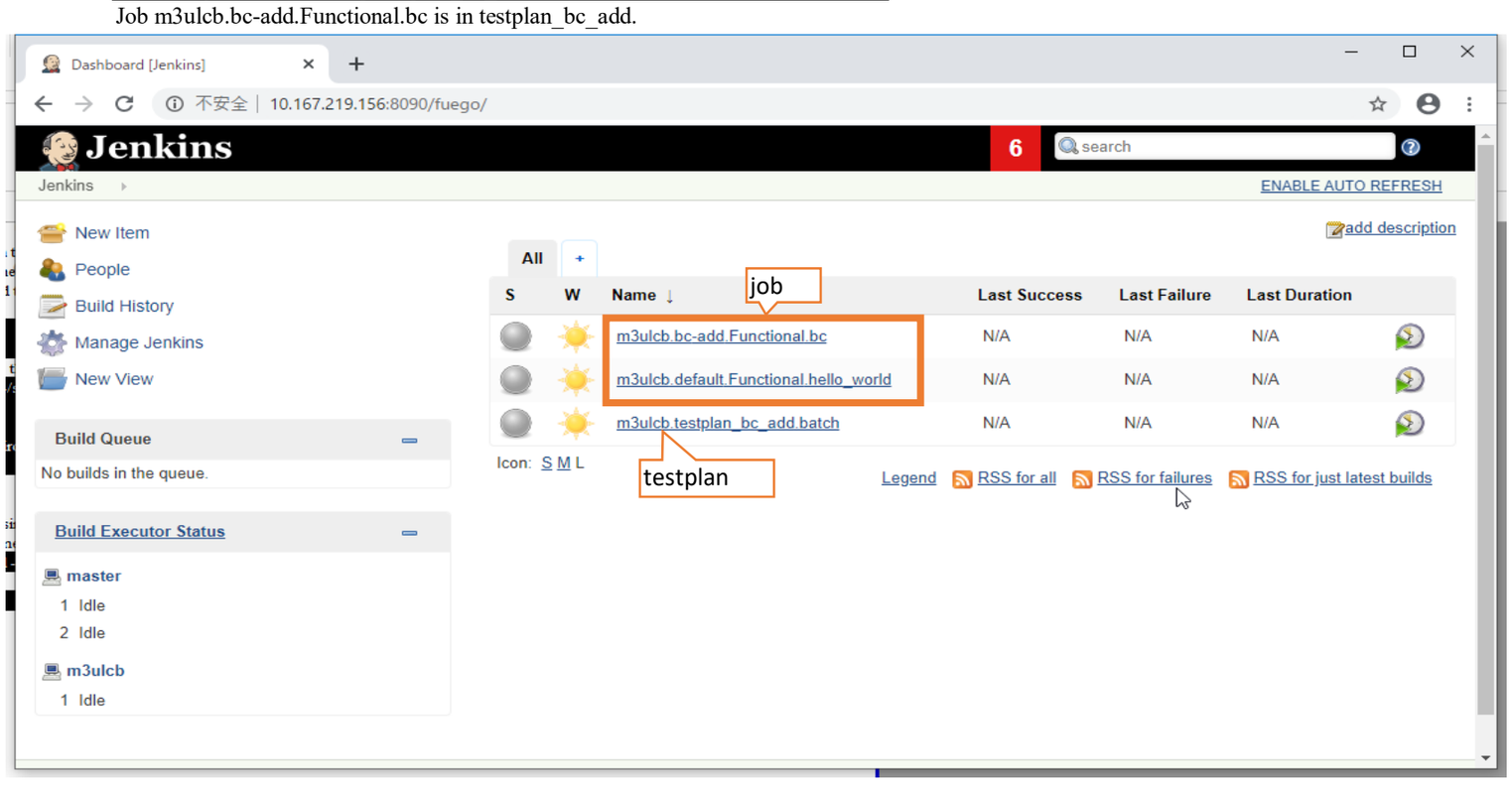
For example:

```
# ftc add-job -b m3ulcb -t Functional.hello_world -s default
```

or

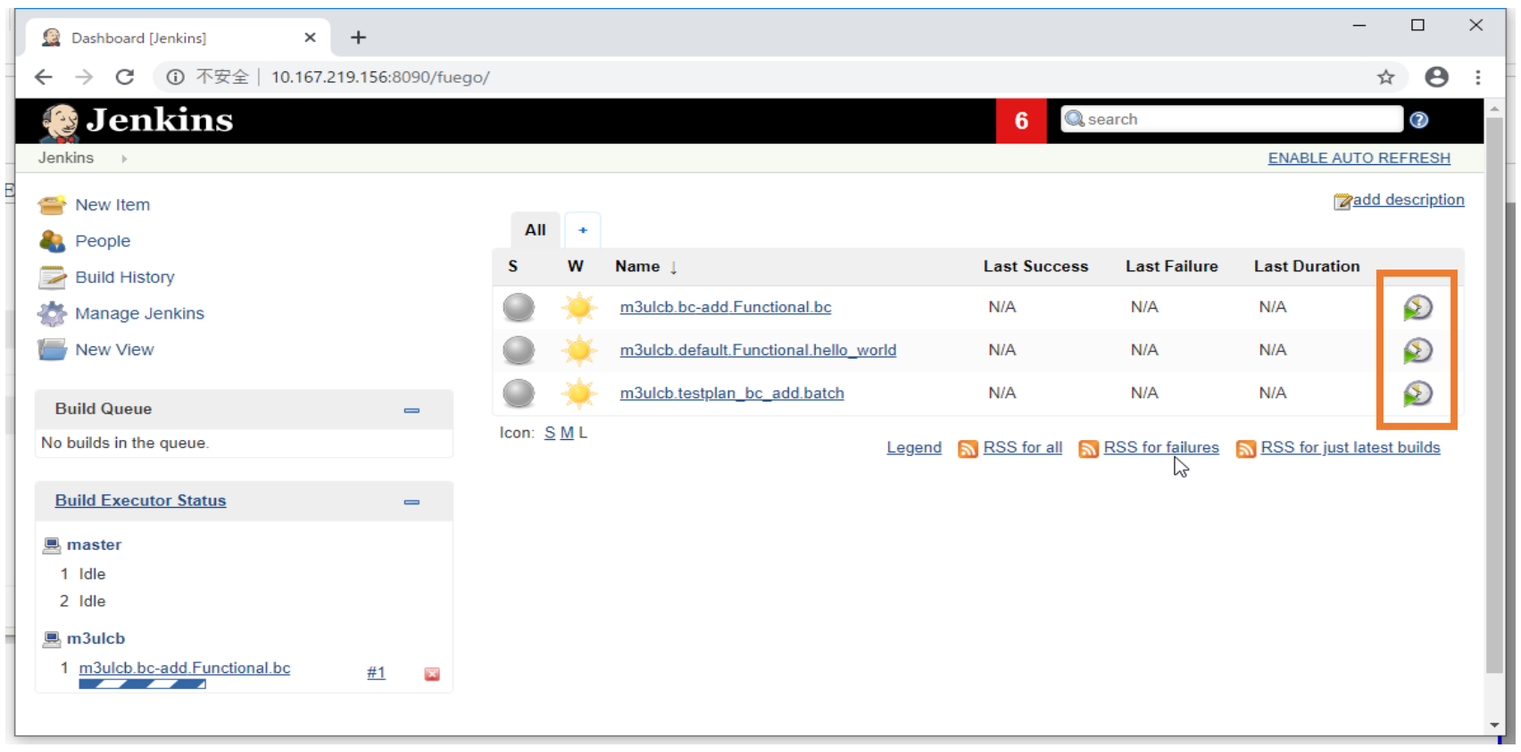
```
# ftc add-jobs -b m3ulcb -p testplan_bc_add
```

Job m3ulcb.bc-add.Functional.bc is in testplan_bc_add.

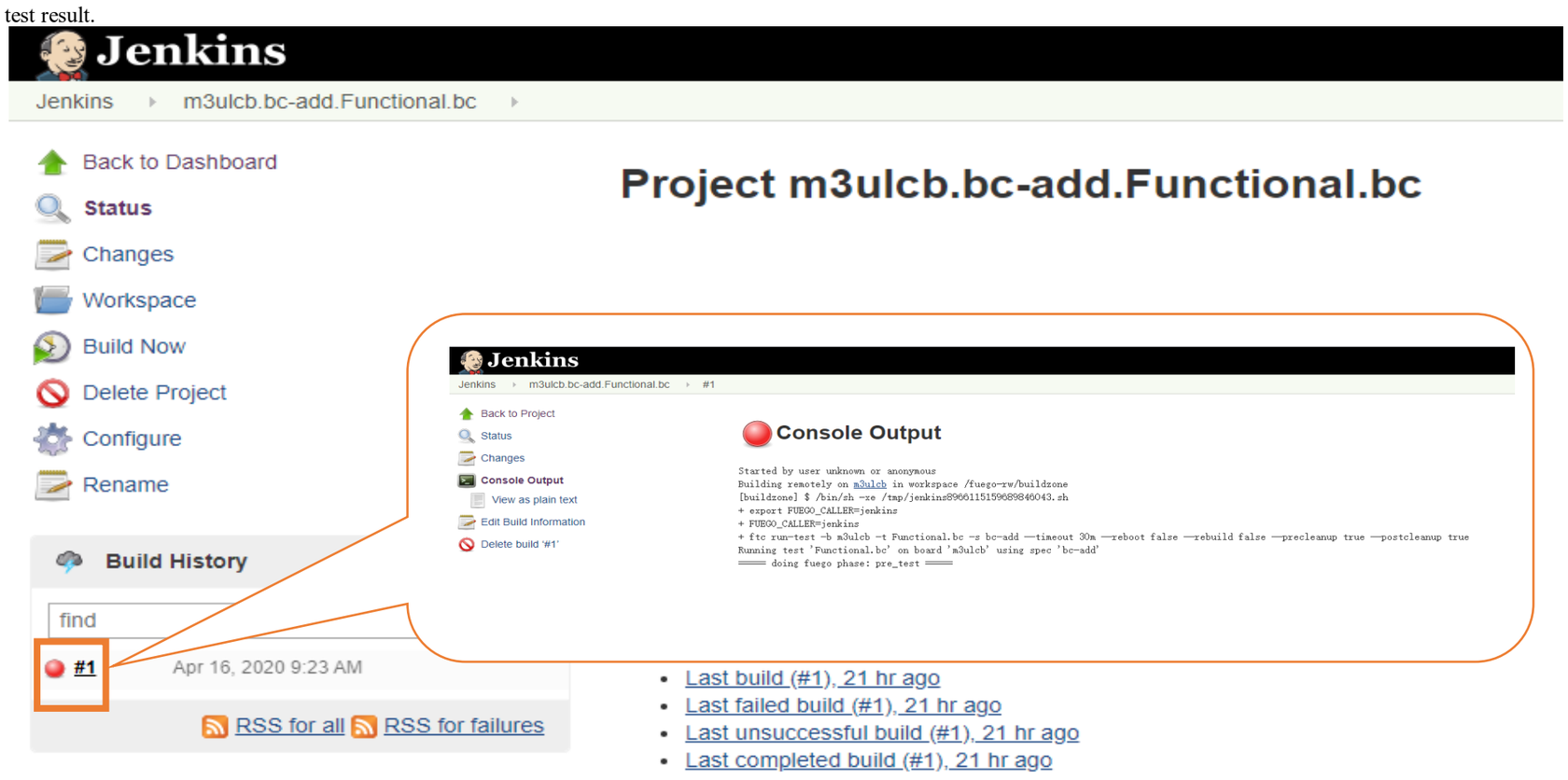


9.Run test jobs

click  button



10. Check test result.



refer to

1. http://fuegotest.org/wiki/FrontPage
2. https://bitbucket.org/fuegotest/fuego/src/master/

Construction of LAVA

1. introduction of LAVA

An automated validation architecture primarily aimed at testing deployments of systems based around the Linux kernel on ARM devices
The project's web site is at: https://validation.linaro.org/

2. prepare for Fuego

To retrieve the Fuego software and create the docker image for it, you need to have git and docker installed on your system.

```
$ sudo apt-get install git
$ sudo apt-get install docker.io
```

3 Installing LAVA with docker

3-1. Git clone this repo "https://github.com/kernelci/lava-docker"

```
$ git clone https://github.com/kernelci/lava-docker.git
```

3-2. Generate configuration files for LAVA, udev, serial ports, etc. from boards.yaml via

```
$ sudo apt-get install python-yaml
$  ./lavalab-gen.py
```

3-3 Go to output/local directory

```
$ cd output/local/
```

3-4 Build docker images via

```
$ sudo apt  install docker-compose
$ sudo docker-compose build
```

3-5 Start all images via

```
$ sudo docker-compose up -d
```

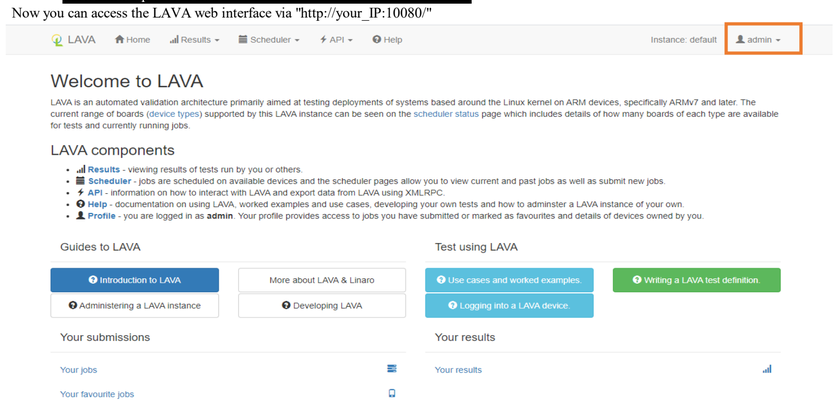
4. Access the LAVA web interface

4-1 You can access the LAVA web interface via "http://localhost:10080/". With the default users, you can login with "admin:admin".

4-2 If you want to use "http://your_ip:10080/"(like http://xxx.xxx.xxx.156:10080/) to accsee lLAVA web interface, use following cmds:

```
$ sudo docker exec -it local_master1_1  bash
$ apt-get update
$ apt-get install vim
$ vi /etc/apache2/sites-available/lava-server.conf
      ProxyPreserveHost On  -->  ProxyPreserveHost Off
$ /etc/init.d/apache2 restart
```

Now you can access the LAVA web interface via "http://your_IP:10080/"



5 Add user to LAVA

The LAVA frontend is developed using the Django web application framework and user authentication and authorization is based on the standard Django auth subsystems.
Local django user accounts can be created with the manage users command(all commands are run in the master docker):

5-1 If you want to add a new user.

```
$ sudo lava-server manage users add <username>  --passwd <password>
```

5-2 Set superuser rights

For example: username : admin

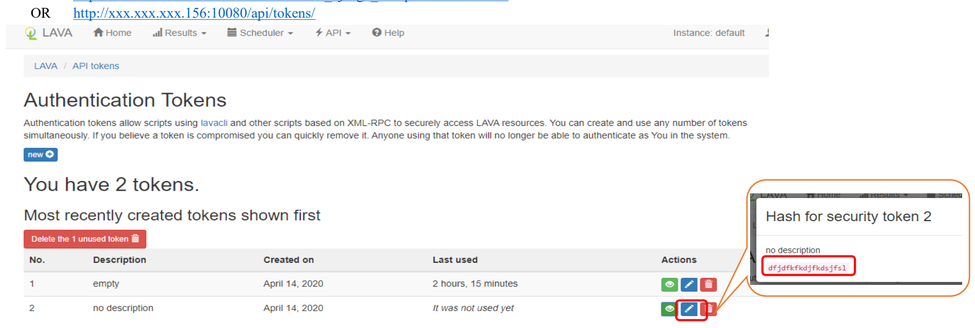
```
$ sudo lava-server manage authorize_superuser --username admin
  User "admin" granted superuser rights
```

6 Add auth token

6-1 Find an unused token(e.g. dfjdfkfkdjfkdsjfsl) from the following url:

http://xxx.xxx.xxx.156:10080/admin/linaro_django_xmlrpc/authtoken/add/

OR http://xxx.xxx.xxx.156:10080/api/tokens/



6-2 Add auth token for user admin(all commands are run in the master docker)

```
$ apt install lava-tool
$ lava-tool auth-add http://admin@xxx.xxx.xxx.156:10080/RPC2/
  Paste token for http://admin@xxx.xxx.xxx.156:10080/RPC2/:dfjdfkfkdjfkdsjfsl
  Token added successfully for user admin.
```

```
$ lava-tool auth-list
  Endpoint URL: http://xxx.xxx.xxx.156:10080/RPC2/
  Tokens found for users: admin
  ----------
```

6-3 Now set the user for this authentication as the default user for this endpoint, and set a shortcut for http://admin@xxx.xxx.xxx.156:10080/RPC2/ as admin

```
$ lava-tool auth-config --default-user http://admin@xxx.xxx.xxx.156:10080/RPC2/
  Auth configuration successfully updated on endpoint http://xxx.xxx.xxx.156:10080/RPC2/.
$ lava-tool auth-config --endpoint-shortcut admin http://admin@xxx.xxx.xxx.156:10080/RPC2/
  Auth configuration successfully updated on endpoint http://xxx.xxx.xxx.156:10080/RPC2/.
$ lava-tool auth-list
  Endpoint URL: http://xxx.xxx.xxx.156:10080/RPC2/
  endpoint-shortcut: admin
  default-user: admin
  Tokens found for users: admin
  ----------
```

7 Use 'ser2net' daemon

ser2net provides a way for a user to connect from a network connection to a serial port, usually over telnet.
http://ser2net.sourceforge.net/
ser2net is a dependency of lava-dispatcher, so will be installed automatically.
Example config as below(in /etc/ser2net.conf):

```
$/etc/ser2net.conf
  7001:telnet:600:/dev/ttyUSB0:115200 8DATABITS NONE 1STOPBIT banner
```

- 7001 is TCP/IP port to be monitored by the TCP server, binded to m3ulcb.
- /dev/ttyUSB0 is the serial port name of the m3ulcb board connected to the LAVA server
- 115200 is the serial baud rate.

8 Add a board(e.g. m3ulcb)

All commands are run in the master docker.

8-1 Using the command line to list all known device types:

```
$ lava-server manage device-types list --all
```

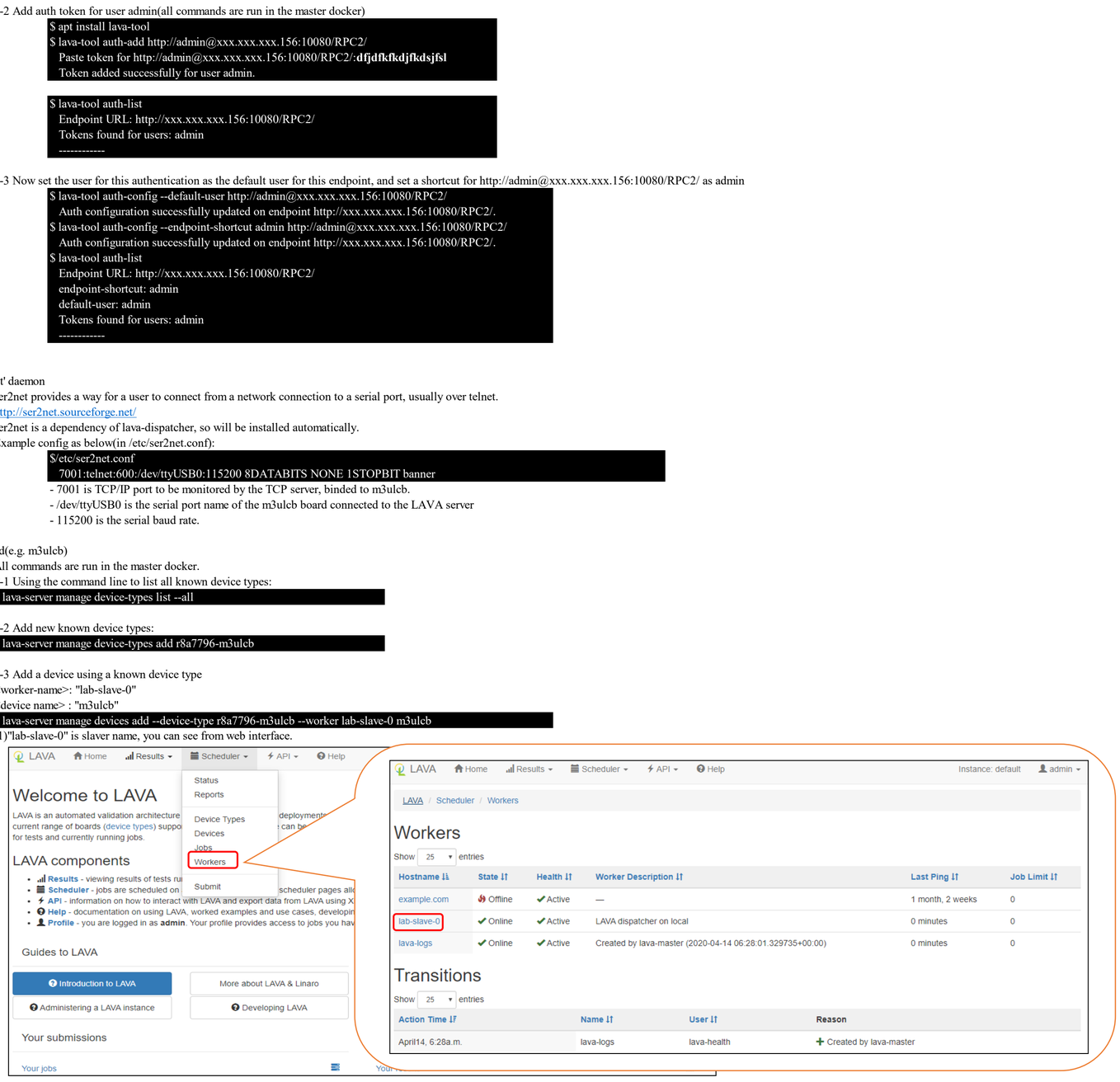
8-2 Add new known device types:

```
$ lava-server manage device-types add r8a7796-m3ulcb
```

8-3 Add a device using a known device type
<worker-name>: "lab-slave-0"
<device name> : "m3ulcb"

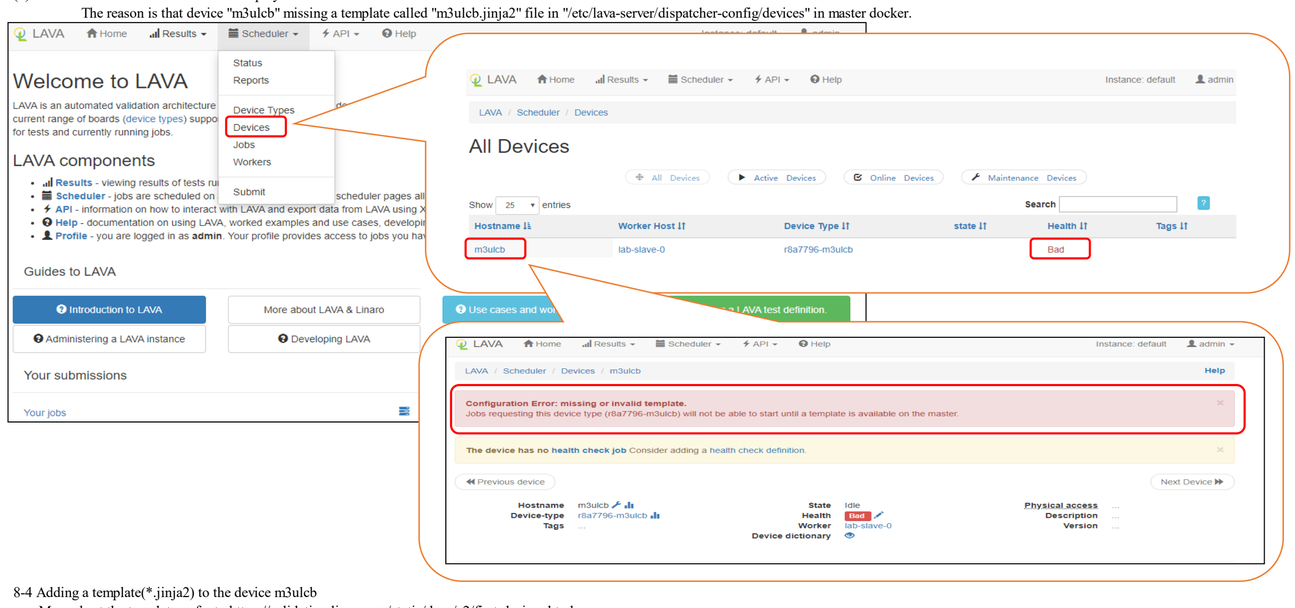
```
$ lava-server manage devices add --device-type r8a7796-m3ulcb --worker lab-slave-0 m3ulcb
```

(1)"lab-slave-0" is slaver name, you can see from web interface.



(2)"m3ulcb" is the device name will be displayed in LAVA interface. We can find that the status of m3ulcb is "bad". Click "m3ulcb" to see details.

The reason is that device "m3ulcb" missing a template called "m3ulcb.jinja2" file in "/etc/lava-server/dispatcher-config/devices" in master docker.



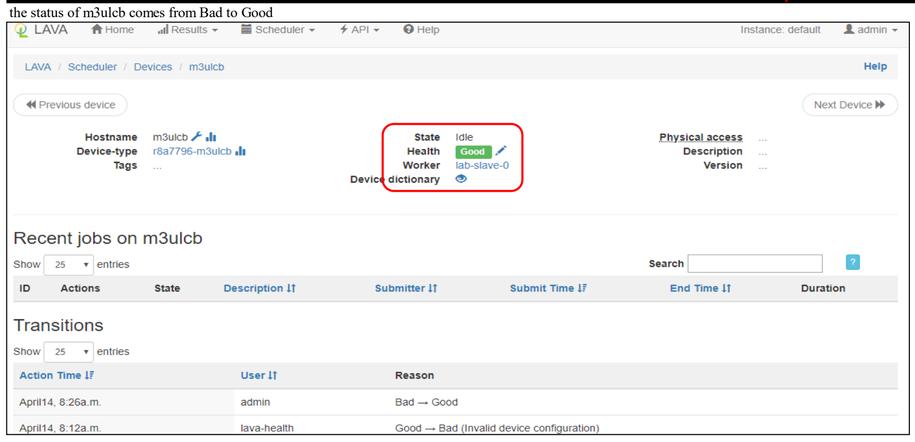
8-4 Adding a template(*.jinja2) to the device m3ulcb

More abort the templete, refer to https://validation.linaro.org/static/docs/v2/first-devices.html

https://validation.linaro.org/static/docs/v2/simple-admin.html#overriding-device-configuration

```
$ cd /etc/lava-server/dispatcher-config/devices
$ vi m3ulcb.jinja2
        {% extends 'r8a7796-m3ulcb.jinja2' %}
        {% set connection_command = 'telnet 172.17.0.1 7001' %}                  #use “telnet 172.17.0.1 7001” to connect board.
                                                                                   "7001" is setted in step 7 and "172.17.0.1" is ipaddr of test PC
        {% set hard_reset_command = '/bin/sh -c /newdisk/LAVA/relay_serial/reboot_m3.sh' %}   #reset board command
        {% set power_on_command = '/bin/sh -c /newdisk/LAVA/relay_serial/power_on_m3.sh' %}   #power on  board command
        {% set power_off_command = '/bin/sh -c /newdisk/LAVA/relay_serial/power_off_m3.sh' %} #power off  board command
        {% set uboot_ipaddr_cmd = 'setenv serverip 192.168.10.10; setenv ipaddr 192.168.10.20' %}  #set board ipaddr=192.168.10.20
                                                                                    set lava server ipaddr=192.168.10.10
```

the status of m3ulcb comes from Bad to Good



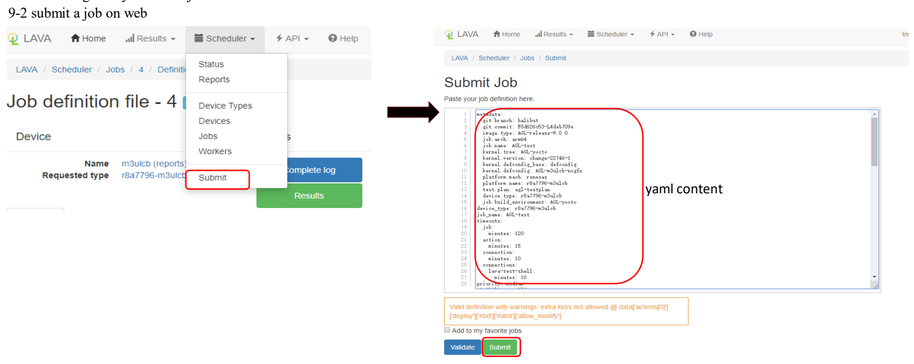
9 Submit a job

9-1 submit a job by cmd(all commands are run in the master docker):

```
$ lava-tool submit-job admin  ./agl-test.yaml
```

agl-test.yaml is the job definition

9-2 submit a job on web

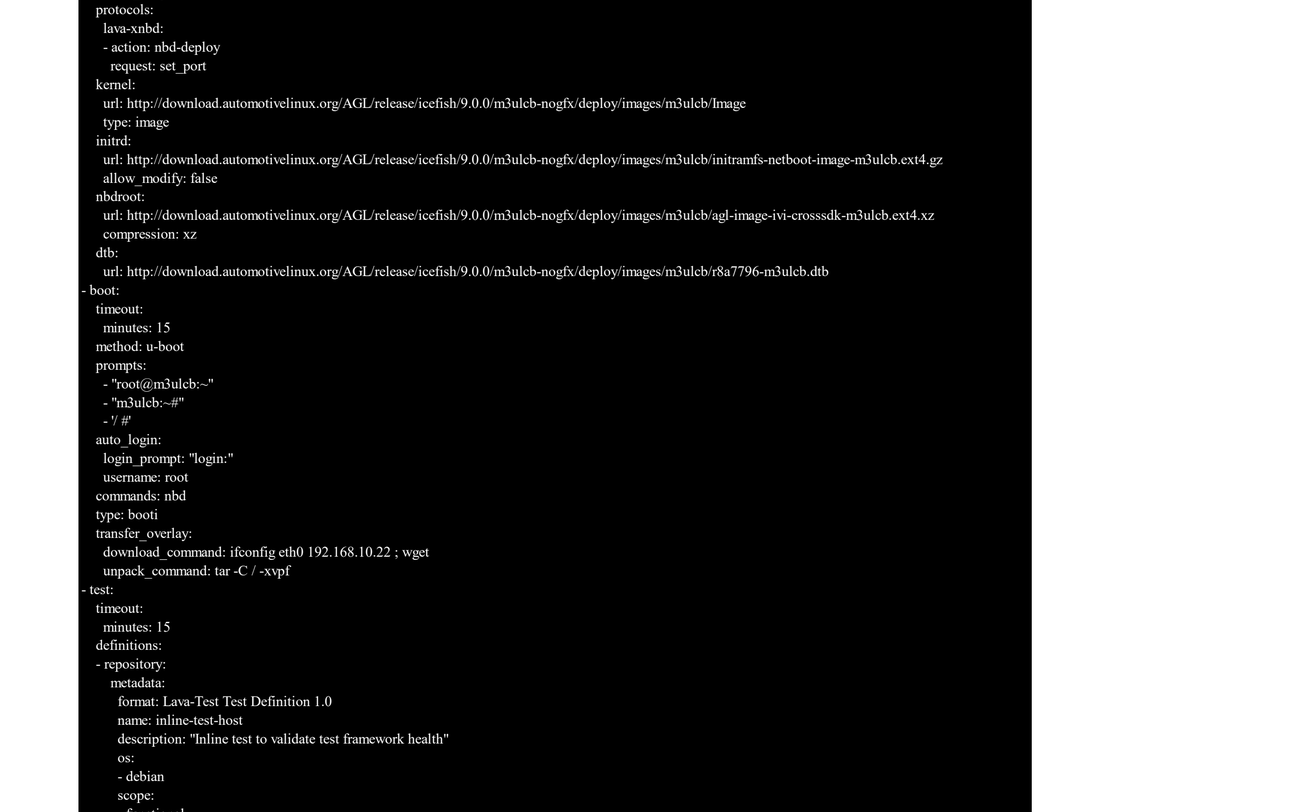


9-3 Yaml file is the job definition, more details refer to https://validation.linaro.org/static/docs/v2/first-job.html

agl-test.yaml:

```
metadata:
  git.branch: halibut
  git.commit: F8d626c53-L4deb709e
  image.type: AGL-release-9.0.0
  job.arch: arm64
  job.name: AGL-test
  kernel.tree: AGL-yocto
  kernel.version: change-22746-1
  kernel.defconfig_base: defconfig
  kernel.defconfig: AGL-m3ulcb-nogfx
  platform.mach: renesas
  platform.name: r8a7796-m3ulcb
  test.plan: agl-testplan
  device.type: r8a7796-m3ulcb
  job.build_environment: AGL-yocto
device_type: r8a7796-m3ulcb
job_name: AGL-test
timeouts:
  job:
    minutes: 120
  action:
    minutes: 15
  connection:
    minutes: 10
  connections:
    lava-test-shell:
      minutes: 10
priority: medium
visibility: public
protocols:
  lava-xnbd:
    port: auto
actions:
- deploy:
    timeout:
      minutes: 60
    to: nbd
    os: oe
    failure_retry: 2
    protocols:
      lava-xnbd:
      - action: nbd-deploy
        request: set_port
    kernel:
      url: http://download.automotivelinux.org/AGL/release/icefish/9.0.0/m3ulcb-nogfx/deploy/images/m3ulcb/Image
      type: image
    initrd:
      url: http://download.automotivelinux.org/AGL/release/icefish/9.0.0/m3ulcb-nogfx/deploy/images/m3ulcb/initramfs-netboot-image-m3ulcb.ext4.gz
      allow_modify: false
    nbdroot:
      url: http://download.automotivelinux.org/AGL/release/icefish/9.0.0/m3ulcb-nogfx/deploy/images/m3ulcb/agl-image-ivi-crosssdk-m3ulcb.ext4.xz
      compression: xz
    dtb:
      url: http://download.automotivelinux.org/AGL/release/icefish/9.0.0/m3ulcb-nogfx/deploy/images/m3ulcb/r8a7796-m3ulcb.dtb
- boot:
    timeout:
      minutes: 15
    method: u-boot
    prompts:
      - "root@m3ulcb:~"
      - "m3ulcb:~#"
      - '/ #'
    auto_login:
      login_prompt: "login:"
      username: root
    commands: nbd
    type: booti
    transfer_overlay:
      download_command: ifconfig eth0 192.168.10.22 ; wget
      unpack_command: tar -C / -xvpf
- test:
    timeout:
      minutes: 15
    definitions:
    - repository:
        metadata:
          format: Lava-Test Test Definition 1.0
          name: inline-test-host
          description: "Inline test to validate test framework health"
          os:
          - debian
          scope:
          - functional
        run:
          steps:
          - ls /
          - uname -a
          - df -h
      from: inline
      name: health-test
      path: inline/health-test.yaml
```

10. Check test result.



refer to

1. https://validation.linaro.org/static/docs/v2/contents.html
2. https://github.com/kernelci/lava-docker

AGL Image and SDK

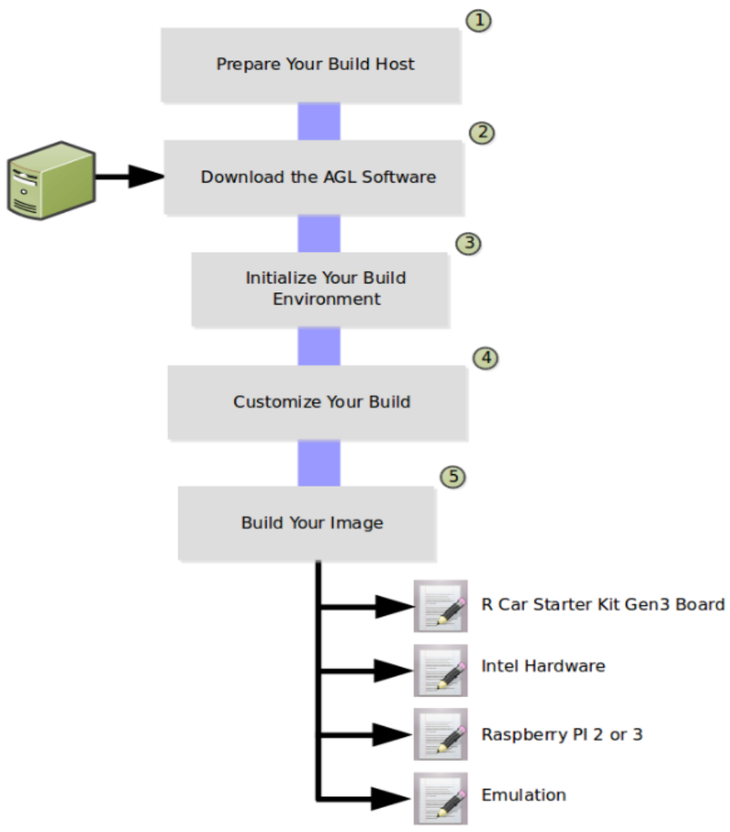
env

board: Renesas R-Car M3

AGL version: Itchy Icefish v9.0.1

overview

The AGL image development workflow consists of setting up the system (i.e. the build host) that builds the image and finishes with using the Yocto Project to create an image targeted towards specific hardware.
The following figure and list overview the AGL image development process.



1. Prepare your build host to be able to use the tools needed to build your image.
Preparing your build host so that it can build an AGL image means making sure your system is set up to use the Yocto Project OpenEmbedded build system

-1)Use a Supported Linux Distribution
-2)Be Sure Your Build Host Has Enough Free Disk Space: Your build host should have at least 50 Gbytes.
-3)Be Sure Tools are Recent: You need to have recent versions for the following tools:
Git 1.8.3.1 or greater
Tar 1.27 or greater
Python 3.4.0 or greater
-4)Install Essential, Graphical, and Eclipse Plug-in Build Host Packages

Usually, a normal ubuntu environment is fine

2. Download the AGL software into a local Git repository on your build host.
Renesas R-Car ivi series need to build manually, others can be downloaded from following:
https://download.automotivelinux.org/AGL/release/
Download the release image/sdk. If do this you can skip all steps below.
image (e.g. UP2 icefish 9.0.1)

```
$ wget https://download.automotivelinux.org/AGL/release/icefish/9.0.1/intel-corei7-64/deploy/images/intel-corei7-64/agl-demo-platform-crosssdk-intel-corei7-64.wic.xz
```

Usually, download *.wic.xz
SDK (e.g. UP2 icefish 9.0.1)

```
$ wget https://download.automotivelinux.org/AGL/release/icefish/9.0.1/intel-corei7-64/deploy/sdk/poky-agl-glibc-x86_64-agl-demo-platform-crosssdk-corei7-64-toolchain-9.0.1.sh
```

Usually, download *.sh

3 Download the latest source code from git
3-1.Prepare Repo:

```
$ mkdir ~/bin
$ export PATH=~/bin:$PATH
$ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
$ chmod a+x ~/bin/repo
```

3-2.download code from master:

```
$ export AGL_TOP=/work/agl-test/icefish-9.0.1
$ cd $AGL_TOP
$ repo init -b icefish -m icefish_9.0.1.xml -u https://gerrit.automotivelinux.org/gerrit/AGL/AGL-repo
$ repo sync
```

4. Run the build environment script to initialize variables and paths needed for the build.
For example: m3ulcb

```
$ export XDG_DOWNLOAD_DIR=$HOME/Downloads
$ export MACHINE=m3ulcb
$ cd $AGL_TOP
$ source meta-agl/scripts/aglsetup.sh -m $MACHINE -b build agl-devel agl-demo agl-netboot agl-appfw-smack agl-localdev
$ grep -w -e "^MACHINE =" $AGL_TOP/build/conf/local.conf
```

5. Make sure your build configuration is defined exactly how you want it for your build.
usually, we need to do nothing during this step

6. Build Images.

```
$ bitbake agl-demo-platform
```

The build process puts the resulting image in the Build Directory:
For example:
$AGL_TOP/build/tmp/deploy/images/$MACHINE/

7. Build SDK
Download proprietary drivers from the R-Car H3/M3 Software library and Technical document site into '/Downloads'.
Refer to the following url to download Proprietary Drivers
https://docs.automotivelinux.org/docs/en/master/getting_started/reference/getting-started/machines/renesas.html

Build

```
$ bitbake agl-demo-platform-crosssdk
```

The SDK installer file (*.sh) is placed in the build directory.
For example:
$AGL_TOP/build/tmp/deploy/sdk/poky-agl-glibc-x86_64-agl-demo-platform-crosssdk-aarch64-toolchain-9.0.1.sh

refer to
https://wiki.automotivelinux.org/agl-distro/release-notes?s[]=repo&s[]=init#itchy_icefish_v901
https://docs.automotivelinux.org/docs/en/master/getting_started/reference/getting-started/machines/renesas.html
https://docs.automotivelinux.org/docs/en/master/getting_started/reference/getting-started/app-workflow-sdk.html
https://docs.automotivelinux.org/docs/en/master/getting_started/reference/getting-started/image-workflow-intro.html