Bootstraping a Local KernelCI

Bootstraping a Local KernelCI

...in less than a day ;)

COLLABORA

Michał Gałka

Consultant Senior Software Engineer

KernelCI

Open First

COLLABORA

KernelCI 101

KernelCI

- A system dedicated to test upstream Linux kernel
- A single place to store, view, compare and track the test results
- Distributed test automation system

My first days in KernelCI

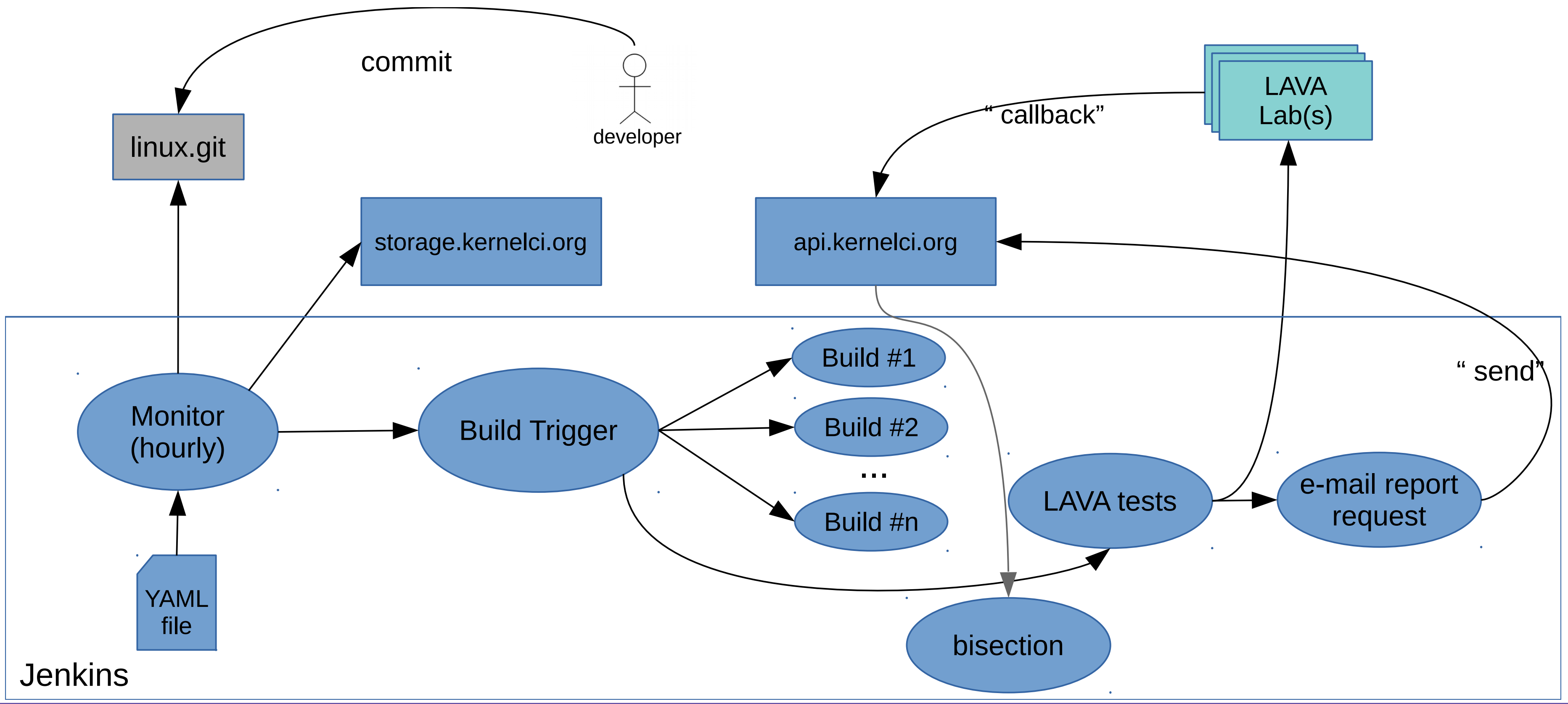
KernelCI beginner’s goals

- Understand how it works
- Understand what’s under the hood
- Build my own local dev environment
- Start development

COLLABORA

The Environment

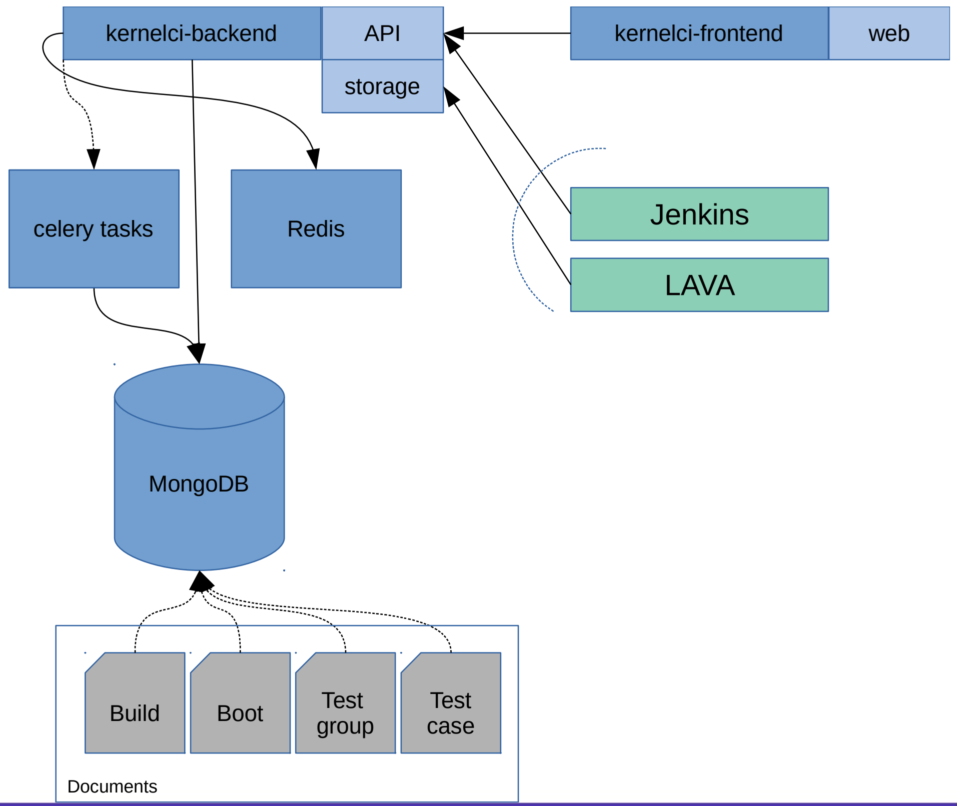
The KernelCI



Work phases

- Probe repository
- Build Kernel
- Upload artifacts to the storage
- Run tests
- Send results to the KernelCI backend
- Prepare and e-mail the test report

Software components



- Several complex software components
- Loosely coupled (mainly REST APIs)
- Needs configuration to properly interact with each other

COLLABORA Open First

COLLABORA

Setup a local environment
Take 1

Plan

- Install KernelCI
  - Use ansible playbooks from:
    - https://github.com/kernelci/kernelci-frontend-config.git
    - https://github.com/kernelci/kernelci-backend-config.git
- Install LAVA
  - Create Debian VM and install debian packaged lava master and dispatcher

Plan

- Install Jenkins
  - Use the Debian package
- Configure system
  - Create necessary tokens via REST APIs
  - Create LAVA Qemu device
  - Create Jenkins jobs

Results

- Pros
  - Method proven to work
  - Very similar to the production KCI config

Results

- Cons
  - Takes a lot of time to setup
    - Setting up VM(s)
    - Jenkins jobs
  - kernelci-backend-config and kernelci-frontend-config INSTALL files contain over 300 lines

COLLABORA

Setup a local environment Take 2

Plan

- Install KernelCI and LAVA dockerized environment
- Configure local environment
- Use kci_build to run builds and submit build artifacts

Containers

- KernelCI
  - kernelci/kernelci-docker is still work in progress and a bit outdated
  - There is a fork of the krenelci-docker used by the Automotive Grade Linux project
    - https://github.com/lucj/kernelci-docker
    - Provides all necessary components: frontend, backend, storage as well as proxy, celery and redis
    - Provides an API token to the frontend
    - Local KernelCI code can be easily plugged into the container

Containers

- LAVA
  - Dan Rue’s lava-docker-compose facilitates LAVA setup process
    - https://github.com/danrue/lava-docker-compose
    - Creates all necessary containers: lava-master, lava-disptacher, PostgreSQL and Squid proxy
    - Pre-configures the lava-master
      - Creates admin account
      - Creates `qemu` device-type and `qemu-01` device

Configuration

- Running KernelCI and LAVA docker containers is fairly simple...
- ...but they need some configuration to interact with each other

Start containers

- Start KernelCI

```
$ ./dev-start.sh
```

```
  -> waiting for backend...
  -> waiting for frontend...
  -> configuring the application...
  -> requesting token from backend...
  -> token returned: 9d413d7b-f9de-4d4e-a801-29f15a8ff9f0
  -> wait while frontend is restarted

  -> application configured
  --> frontend available on port 8080
  --> backend  available on port 8081
  --> storage  available on port 8082
```

KernelCI API master token

KernelCI ports

Start containers

- Run LAVA

```
$ make
```

- `Forwards port 80`
  - `Web panel`
  - `REST API`

Network configuration

- Make sure that KernelCI and LAVA containers see each other

```
$ docker network connect kernelcidocker_default lava_server
$ docker network connect kernelcidocker_default lava_squid
$ docker network connect kernelcidocker_default lava_dispatcher
```

Configuration

- Add a lab to the KernelCI
  - kernelci-admin repo contains a kci tool that facilitates administrative tasks
  - https://github.com/kernelci/kernelci-admin.git
- The `kci` tool facilitates token CRUD operations
  - Leverages the KernelCI REST API
  - Can be used for multiple KernelCI installations
  - Keep KernelCI host configuration in `_settings.py`

kernelci-admin configuration

- Edit the `_settings.py` file

KernelCI instance name

KernelCI backend URL

```
HOSTS = {
    'kci-local': {
        'url': 'http://127.0.0.1:8081',
        'token': '9d413d7b-f9de-4d4e-a801-29f15a8ff9f0',
    },
}
```

API token

COLLABORA

Open First

Configuration

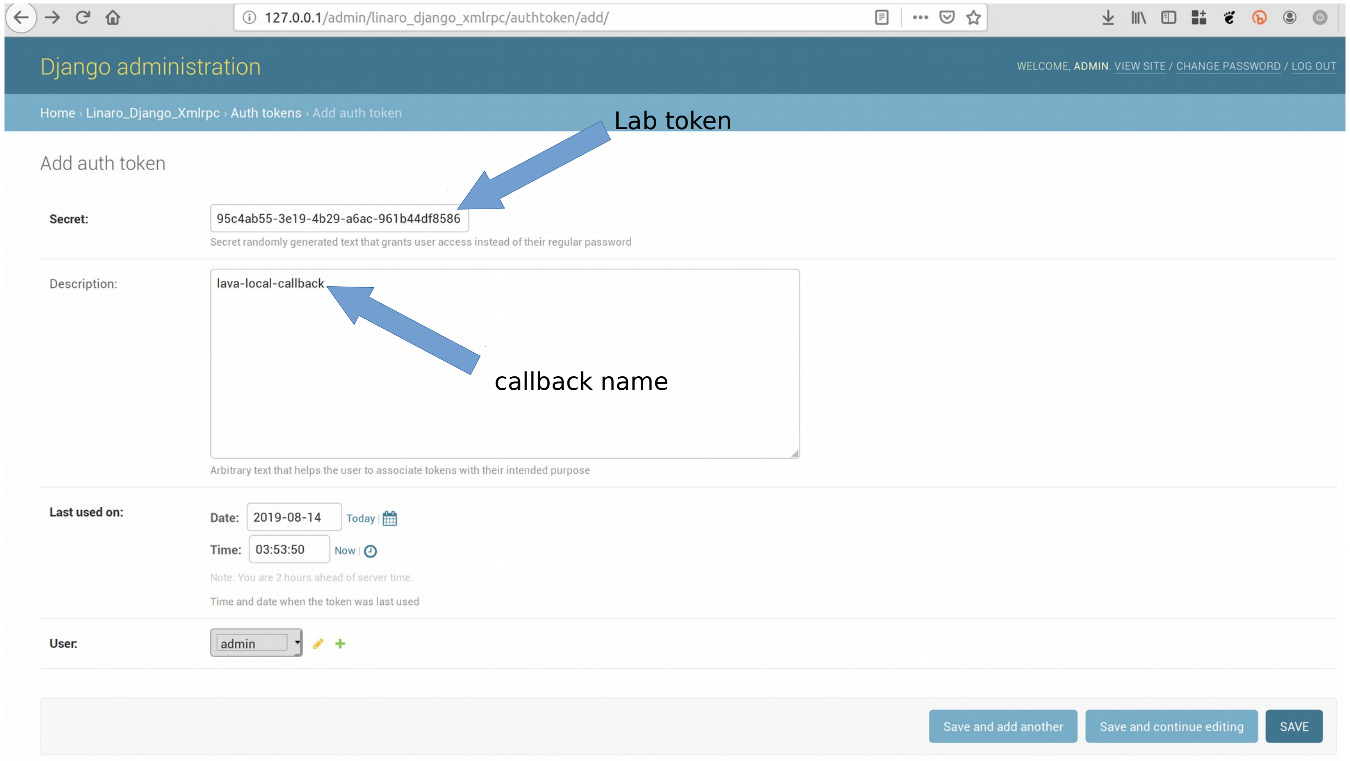
- Add a lab token

```
./kci add_lab --host kci-local --lab-name lava-local --first-name John --last-name Doe \
--email john.doe@collabora.com

  _id
    $oid                      5d5383555150d500408ee9de
  token                       95c4ab55-3e19-4b29-a6ac-961b44df8586
  name                        lava-local
```

lab token

Configure callback



127.0.0.1/admin/linaro_django_xmlrpc/authtoken/add/

Django administration

WELCOME, ADMIN. VIEW SITE / CHANGE PASSWORD / LOG OUT

Home › Linaro_Django_Xmlrpc › Auth tokens › Add auth token

Add auth token

Secret: 95c4ab55-3e19-4b29-a6ac-961b44df8586

Secret randomly generated text that grants user access instead of their regular password

Description: lava-local-callback

Arbitrary text that helps the user to associate tokens with their intended purpose

Last used on:

Date: 2019-08-14 Today |

Time: 03:53:50 Now |

Note: You are 2 hours ahead of server time.

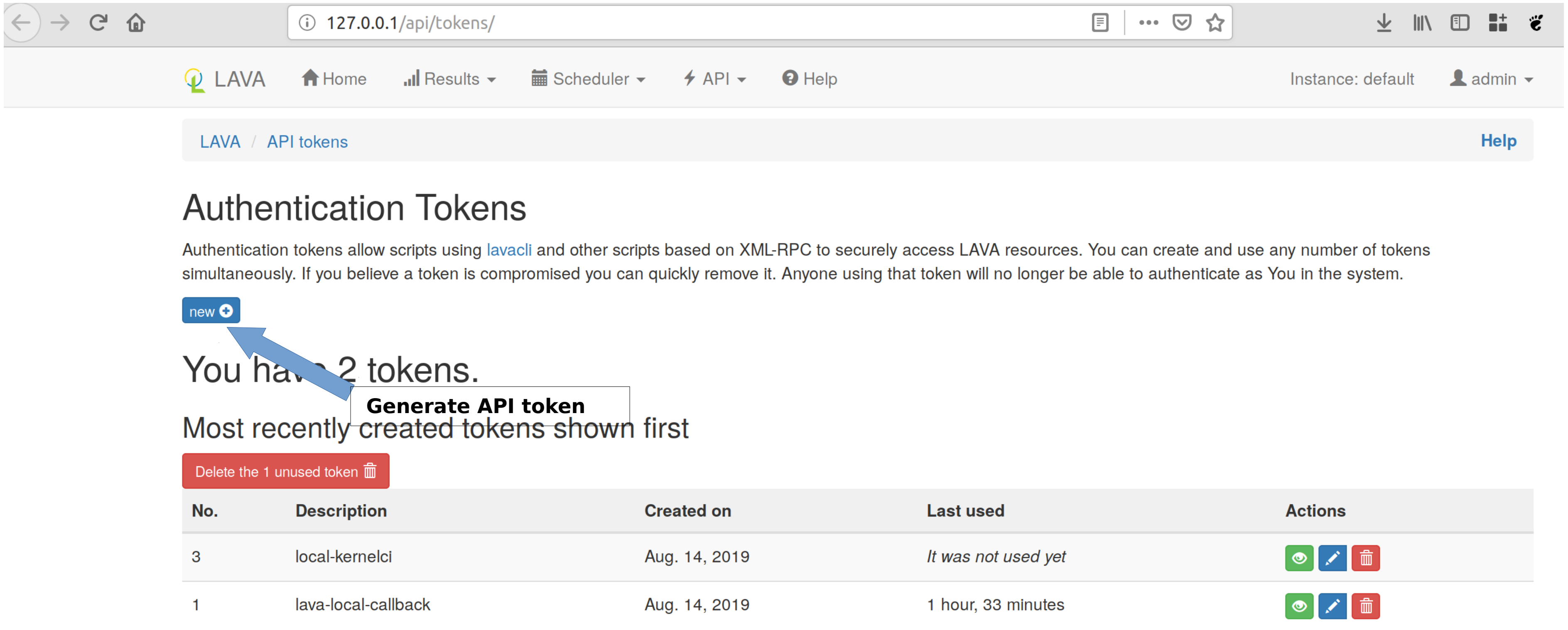
Time and date when the token was last used

User: admin

COLLABORA

Open First

Generate LAVA API token



Authentication Tokens

Authentication tokens allow scripts using lavacli and other scripts based on XML-RPC to securely access LAVA resources. You can create and use any number of tokens simultaneously. If you believe a token is compromised you can quickly remove it. Anyone using that token will no longer be able to authenticate as You in the system.

You have 2 tokens.

Most recently created tokens shown first

No.	Description	Created on	Last used	Actions
3	local-kernelci	Aug. 14, 2019	*It was not used yet*	
1	lava-local-callback	Aug. 14, 2019	1 hour, 33 minutes	

COLLABORA

Open First

Build kernel

- kci_build
  - https://github.com/kernelci/kernelci-doc/wiki/KernelCI-command-line
  - A tool that facilitates kernel building and publishing
  - Provides clear and generic way to call the build phases
  - Makes it possible to use orchestrators different than Jenkins
  - ...or simply use the CLI

Prepare the repository

```
$ git clone \
  --mirror git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git \
  --reference=~/src/linux linux-mirror.git

$ ./kci_build update_repo --config=next --kdir=linux --mirror=linux-mirror.git
```

config name from
build-configs.yaml

Build kernel

```
$ ./kci_build generate_fragments --config=next –kdir=linux

$ ./kci_build build_kernel –defconfig=defconfig \
 --arch=arm64 --build-env=gcc-8 --kdir=linux

$ ./kci_build install_kernel --config=next --kdir=linux
```

COLLABORA

Open First

Upload kernel

- Push kernel binaries to the storage

```
$ ./kci_build push_kernel –kdir=linux \
--api=http://127.0.0.1:8081 \
--token 9d413d7b-f9de-4d4e-a801-29f15a8ff9f0
```

- Publish kernel metadata

```
$ ./kci_build publish_kernel –kdir=linux \
--api=http://127.0.0.1:8081 \
--token 9d413d7b-f9de-4d4e-a801-29f15a8ff9f0
```

Run tests

- There are helper scripts in kernelci-core repository
  - lava-v2-jobs-from-api.py
    - Generates LAVA test jobs based on available builds, devices and test plans
  - lava-v2-submit-jobs.py
    - Submits previously generated test jobs to LAVA

KernelCI developer’s tips

- Using dockerized kernelCI and LAVA may be a good choice unless you’re setting up a real lab.
- Use *kci* tool from kernelci-admin to manage labs and tokens instead of the raw REST API
- Use *kci_build* if you want to test a build in your local dev environment.

What’s next?

- Develop *kci_test*
  - A tool similar to kci_build, that’ll facilitate running tests and gathering results
- Continue LAVA and KernelCI docker configuration automation
  - Create a fully working test environment with reasonable default configuration to make developer’s life easier

Thank you!