Embedded Linux Conference

Even More Board Farm Goodness An Update on the REST API for Automated Testing

Harish Bansal

Technical Engineer
Timesys

Tim Bird

Principal Software Engineer
Sony Electronics

THE LINUX FOUNDATION

Abstract

This talk presents an update on work to create a standard API between automated tests and board farm hardware and software. Previously, we introduced the notion of a dual REST/command-line API that could be used for discovery, control and operation of hardware and network resources in a test lab. We would like to highlight additional progress of the project since then.. There are now APIs for control of audio hardware in the lab. Also, the API usage has been integrated into a few different testing frameworks. We will describe the new APIs we have added, and demonstrate new tests frameworks that work with the REST API system. This includes frameworks for performing UI testing of the device under test.

Although different equipment is utilized in different test labs (or board farms), by using the REST API the same test can be run in the different labs to obtain test results and provide quality assurance for products. It is hoped that this board farm API abstraction will pave the way for more sharing of automated tests and testing resources, to accelerate the use of automated testing for products based on embedded Linux.

Outline

- Review of REST API concepts
  - Review of Farm Resource model
- Updates since last year…
- Audio testing APIs (and demo!)
- Web application testing
- Integration into Jenkins CI Pipeline
- Future directions

Problem statement (review)

- **There are many tests but no standardized way of running tests on physical devices**
- **There are many different Test Frameworks**
- **There are a few Board Farm frameworks**
    - But no standardized way to use different Test Frameworks or run tests
- **Every farm implements test infrastructure differently**
    - Many labs use ad-hoc infrastructure
        - Cobble together available hardware, and write custom scripts for control and data collection
    - Tests written for one lab do not work in another lab
- **Nobody can share tests**

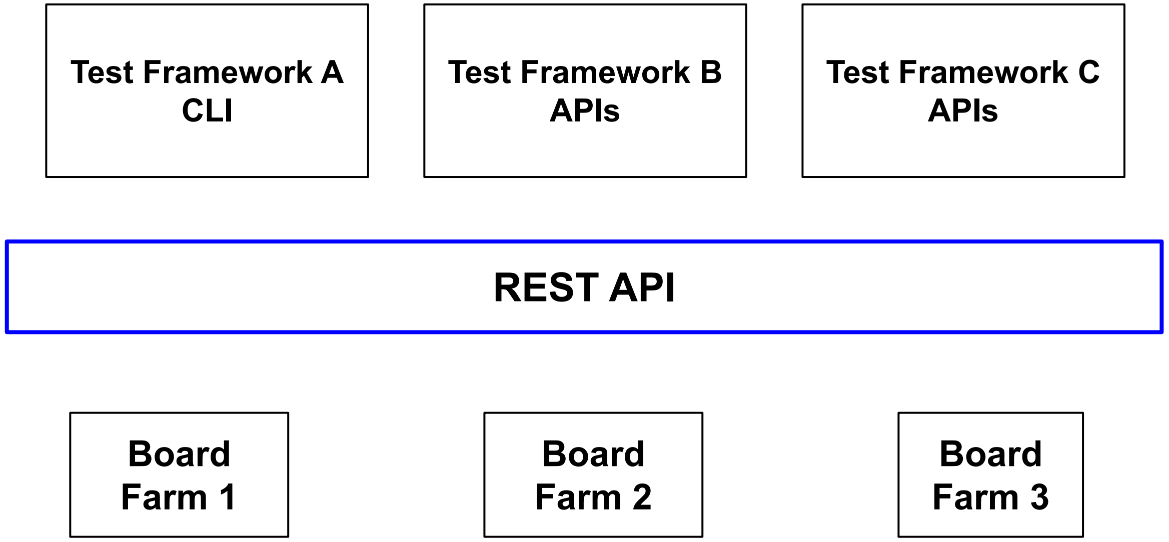
Solution:

- **Creating a standard method to access a Board Farm allows:**
    - Board Farm technologies can evolve separately from the interface to the farm
    - Tests can be written that work in more than one lab
    - Test Frameworks can work with more than one lab

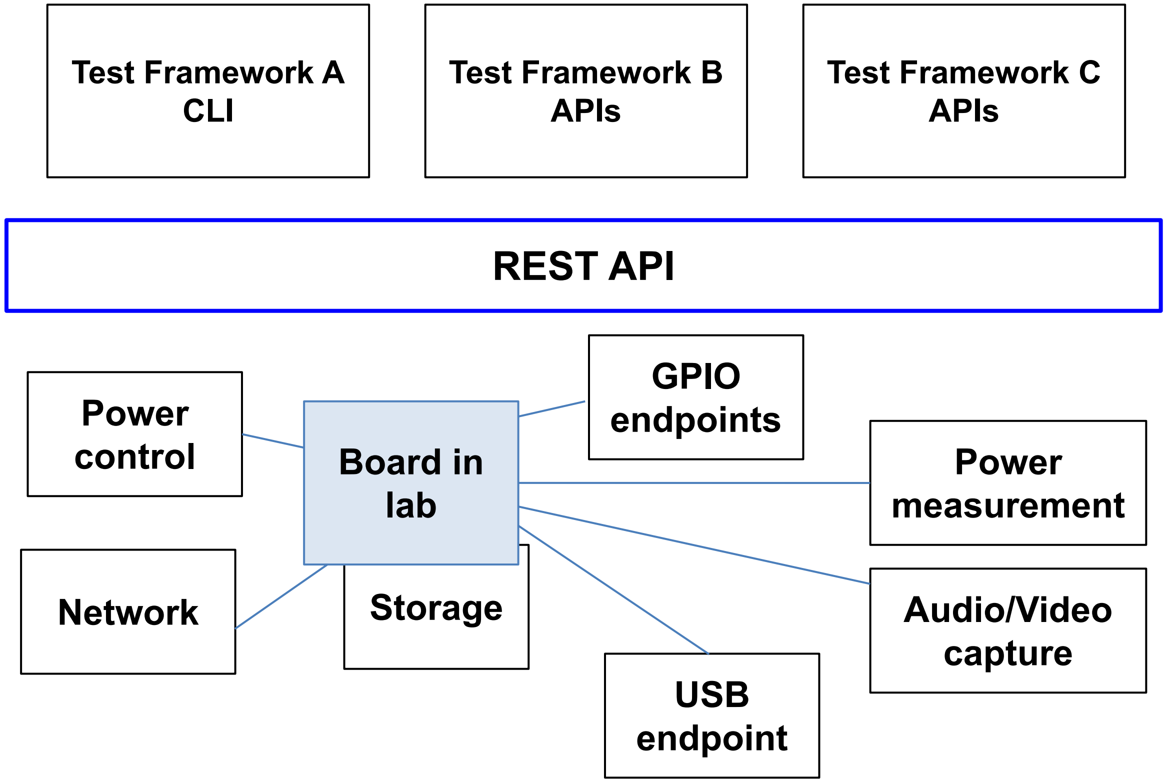
Examples of Hardware/Software integration tests

- **GPIO test, Serial Port test**
  - Need to control two endpoints
    - One on device under test (DUT) and one external endpoint
- **Audio, Video playback test**
  - Need to control two endpoints
    - One on device under test (DUT) and a capture device
- **Power measurement (via external power monitor)**
  - Need to control two endpoints:
    - Application or workload profile on DUT
    - Capture of power measurement data on external power monitor
- **USB connect/disconnect (robustness) testing**
  - Need to control two endpoints:
    - Application or monitor on DUT
    - USB hardware external to board (drop/re-connect vbus)

High level concept 1 – API between framework and lab



High level concept 2 – API between test and lab



Fuego/EBF REST-API elements

- API proposal
  - 2 parts
    - REST API
    - Command line interface
- REST API based on https and JSON
  - Extension to LAVA REST API
  - Can be implemented solely with 'curl' and 'jq'
- Command line tool
  - Same operations as REST API
  - Suitable for automated use, as well as human interactive use

Multiple implementations

- **LAVA-based implementation (TimeSys)**
  - LAVA server (based on Django)
  - 'ebf' client
    - Is a shell-based client using curl and jq
  - Is in production use now, as part of TimeSys Board Farm Cloud service
  - Git repo: *https://github.com/TimesysGit/board-farm-rest-api*
- **LabControl server and client implementation**
  - lcserver is a plain CGI script (no framework)
  - ‘lc’ client
    - Is a python client, using the python requests module
  - Source is available now (alpha-level quality)
  - Git Repo: *https://github.com/tbird20d/labcontrol.git*

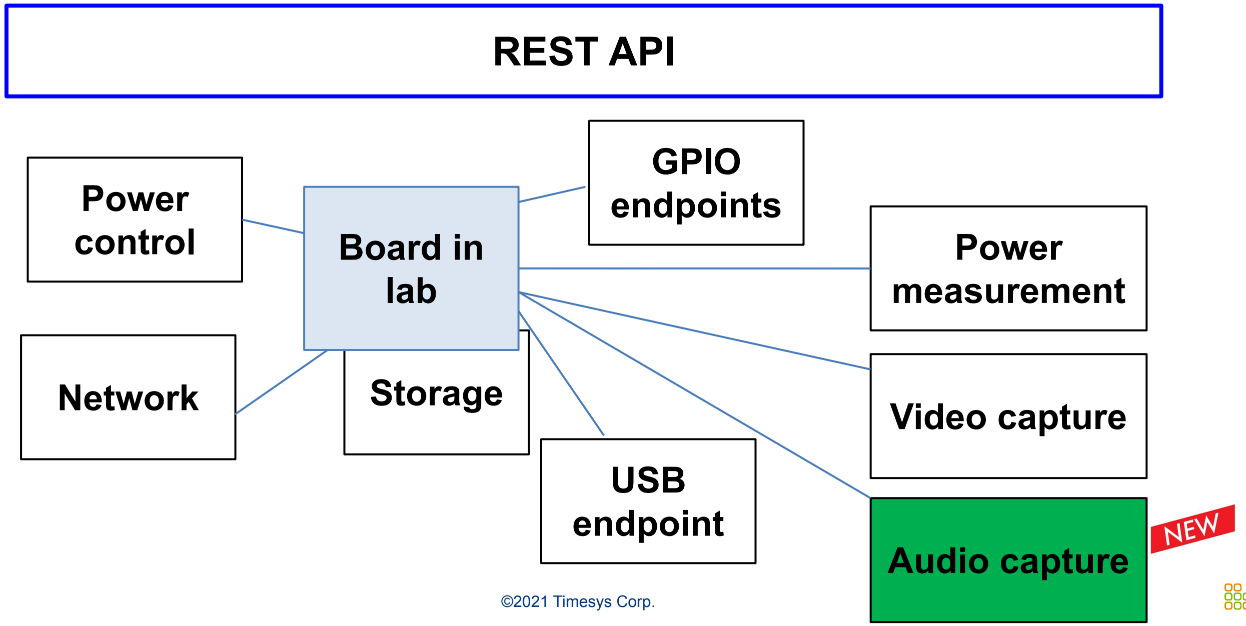
Stuff Added in 2021

- More implementations
- Added the resource model
  - new API: get-resource
- Added the generic capture API model
  - start-capture, stop-capture, get-data, delete
- APIs for new resource types:
  - power measurement
  - image and video capture
  - serial port receive and transmit
- Direct support for API in Fuego
- Created test example in Robot Test Framework

What has happened in the last year?

New Stuff in 2022

- **Added Audio testing APIs**
- **Added integration with Jenkins pipeline (External CI Framework)**
- **Labcontrol added WebTerminal feature for DUT access over web**
    - TimeSys EBF already had that feature



Added audio APIs using the resource model

- Added 'audio' resource type (to get-resource API)
  - api/v0.2/devices/{board}/get-resource/audio/[{feature}]
  - resource=$(ebf rpi4 get-resource audio hdmi2)
- Added optional “feature” argument to get-resource API
  - Indicates the particular input or output item on the board, for which the test is being run
  - Is needed to distinguish which hardware connection on the board is being tested (e.g. hdmi2 audio channel, rear-microphone-jack, etc.)

Supported Resource types

- **Previously supported resource types are:**
  - power-measurement
  - camera
  - serial
- **New Resource type**
  - audio

Re-used the generic capture API model verbs

- **Audio 'capture' API consist of 4 verbs:**
  - start-capture
    - Begin capturing data
    - Returns a token for capture data manipulation
  - stop-capture
    - Stop capturing data
  - ~~get-data~~
    - ~~Retrieve the data from the server~~
  - get-ref
    - Get a URL path to captured data on the server
  - delete
    - Remove the data from the server

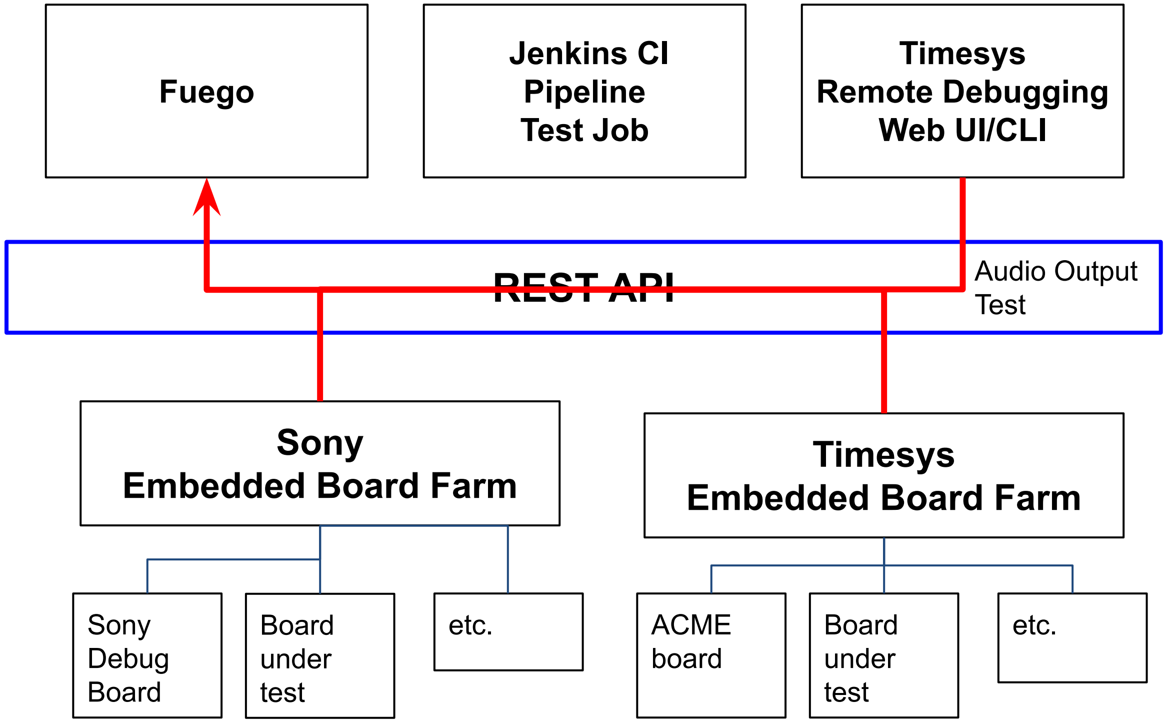
Use case:
Lab-independent Audio Quality Test

Audio REST APIs mapping

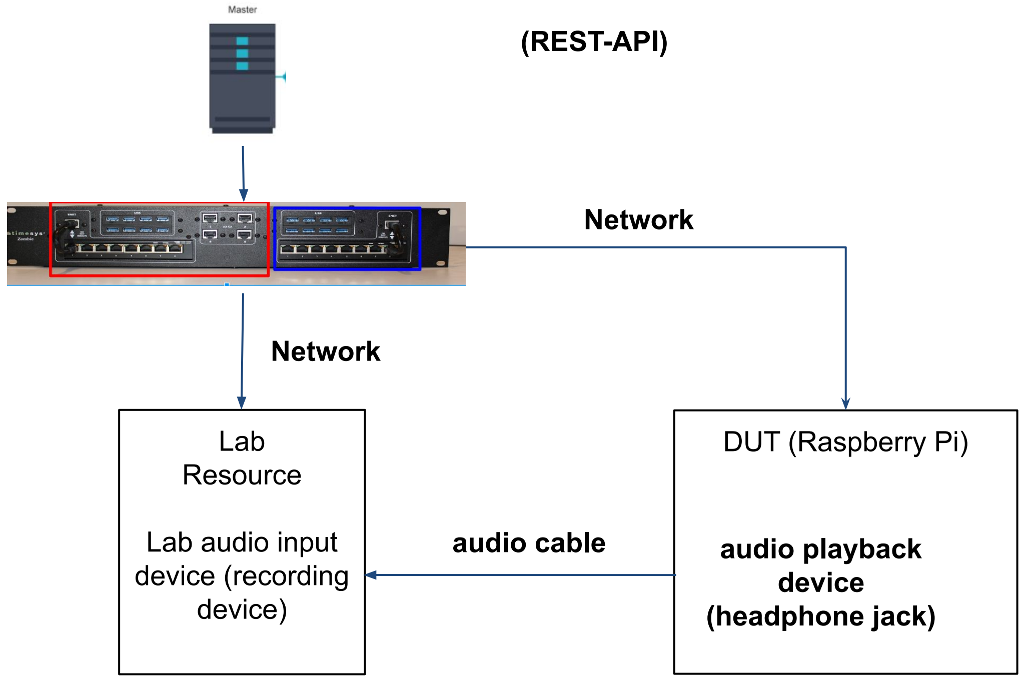
Test operation	CLI command	REST API
01 Resource Detection	ebf <Device Name> get-resource audio <Audio ID>	https://api/devices/<DeviceName>/get_resource/audio/<AudioID>
02 Audio Capture	ebf <Resource Name> audio start ebf <Resource Name> audio stop <Token> ebf <Resource Name> audio get-ref <Token> [-o <Audio file>]	https://api/resources/<ResName>/audio/start https://api/resources/<ResName>/audio/stop/<Token> https://api/resources/<ResName>/audio/get-ref/<Token> https://<Data Ref>
03 Audio Playback	ebf <Device Name> ssh upload <Audio file> <DUT file path> ebf <Device Name> ssh run "<playback command>"	https://api/devices/<DeviceName>/ssh/upload/ https://api/devices/<DeviceName>/ssh/run/
04 Cleanup	ebf <Resource Name> audio delete <Token>	https://api/resources/<ResName>/audio/delete/<Token>

EBF CLI is implemented using the REST API

High Level Concept illustration



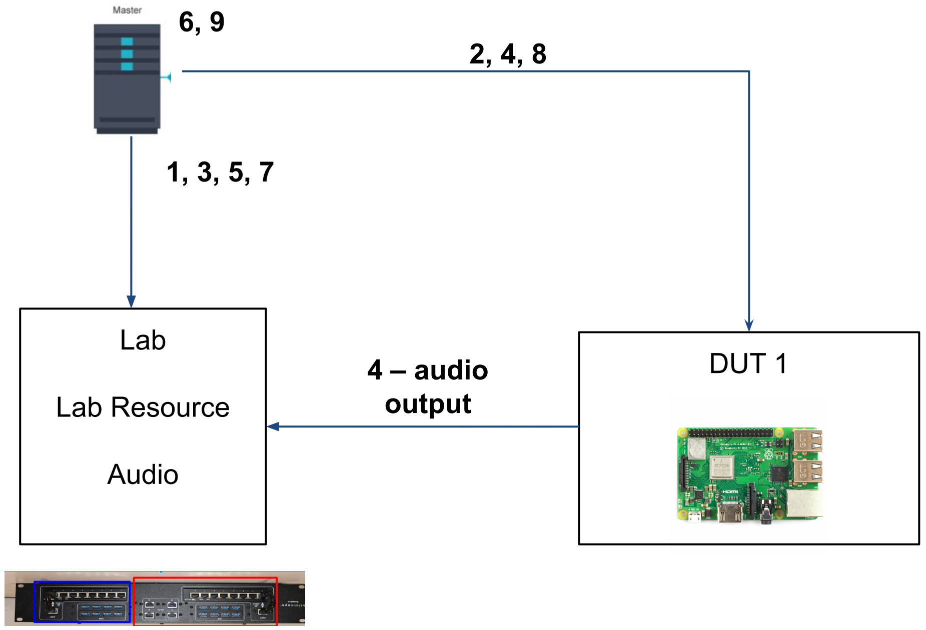
Audio Test – Hardware configuration


Master
(REST-API)
Network
Network
Lab Resource
Lab audio input device (recording device)
audio cable
DUT (Raspberry Pi)
audio playback device (headphone jack)

Hardware Setup:
Lab “knows” the cable connection of DUT and audio recording device

- Lab resource (audio receiver or mic) is connected to the raspberry PI output
  - To a specific audio output
- Test discovers connection at runtime by querying the lab

Audio Testing Sequence



Test of Audio hardware

1. Discover lab resource connected to DUT audio output (lab microphone)
2. Put audio test file on DUT
3. Initiate capture
4. Initiate playback
5. End capture, collect captured audio data
6. Compare played vs captured data

7,8,9. Remove data in lab, on DUT, and on test host

What the API looks like in practice

Excerpt from audio-playback-test.sh:

Setup

Capture & Playback

Data Analysis

Cleanup

```
RESOURCE=$($CLIENT $BOARD get-resource audio $DEVICE_ID)
$CLIENT $BOARD ssh upload $AUDIO_FILE $DUT_TMP_AUDIO_FILE
…
TOKEN="$($CLIENT $RESOURCE audio start)"
$CLIENT $BOARD ssh run "aplay -D $DEVICE $DUT_TMP_AUDIO_FILE"
$CLIENT $RESOURCE audio stop $TOKEN
$CLIENT $RESOURCE audio get-ref $TOKEN -o $CAPTURED_AUDIO_FILE
…
$ALSABAT --readcapture=/tmp/temp2.wav | tee /tmp/audio-test-results.txt
TESTCASE="Alsabat tone check with $base_filename"
if grep PASS /tmp/audio-test-results.txt ; then succeed "$TESTCASE" else fail "$TESTCASE" fi
….
$CLIENT $BOARD ssh run "rm $DUT_TMP_AUDIO_FILE"
$CLIENT $RESOURCE audio delete $TOKEN
rm $CAPTURED_AUDIO_FILE /tmp/temp2.wav /tmp/audio-test-results.txt …
```

```
hbansal@harish-GL552VW:~/SampleTests/audio-test-script $ 
```

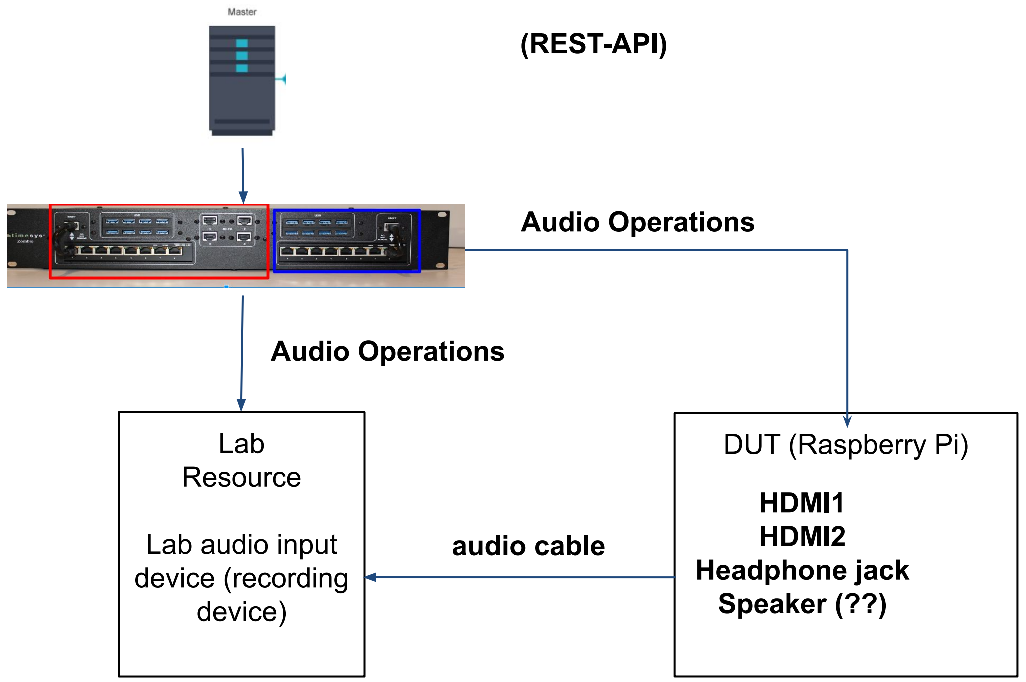
Audio Analysis using Alsabat

- **'alsabat' is part of alsa-utils**
- **Can be used to do automated analysis of playback and capture**
- **Normal usage:**
  - Play a tone and capture in a loopback mode
  - Analyze captured data - by doing tonal analysis
- **Added new mode of operation: (still need to send patch upstream)**
  - Analyze data from a pre-captured file
- **Didn't want to run alsabat on DUT**
  - That’s not where captured data is
  - Compiling for target would be a pain
  - Only need capability to play data on target
    - Used 'aplay' (also from alsa-utils) in test, but could be anything available on the DUT

Audio Analysis Using Sox

- **Sox**
  - Use ‘stat’ filter to get info
- **Audio quality dependent on audio input gain on farm receiver**
- **For low-quality sound reproduction, sampling creates artifacts at weird harmonics, that skew the dominant frequency in the captured audio**
  - **I had to adjust a frequency threshold for one board, to compensate for this**
  - **I wasn't sure whether I should call this a failure or not**

Multiplexing problem



This setup requires "rigging" for each audio connection on the device.

The device might have multiple audio outputs:
- HDMI1, HDMI2
- headphone jack
- onboard speaker

Multiplexing problem

- **Hardware connections with a device present a "multiplexing problem"**
- **Devices have multiple connections that need to be tested**
  - USB, audio, video, gpio, power
- **In order to test them physically, you need to attach a device to each one**
  - Requires some kind of extreme rigging - custom per lab
  - Unless you go through a bus which has some kind of multiplexing capability
    - Does the test have to know about that, to arrange the multiplexing, or can the lab handle it transparently?
      - e.g. Establish the physical connection on a get-resource call
- **Probably need reservation and release operations for lab resources**

Multiplexing problem (cont.)

- What to do about built-in speakers?
- In this case, the "air" is a shared connection between speakers and a single lab microphone
- Could use the same microphone for all speaker outputs
- Has to be quiet in the lab during the test
  - Even if no one is talking or typing, there is fan noise

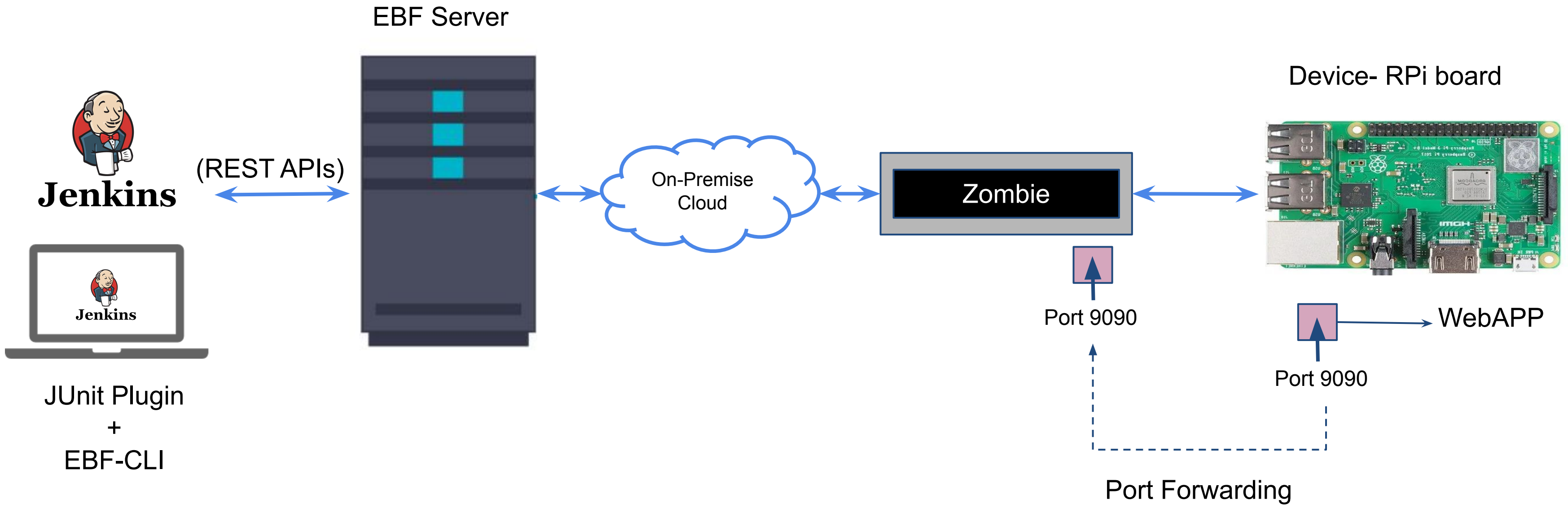
- Need to reserve the "open soundspace" in the lab, and only conduct one audio test using that resource at a time

Web APP CD & CT Pipelines

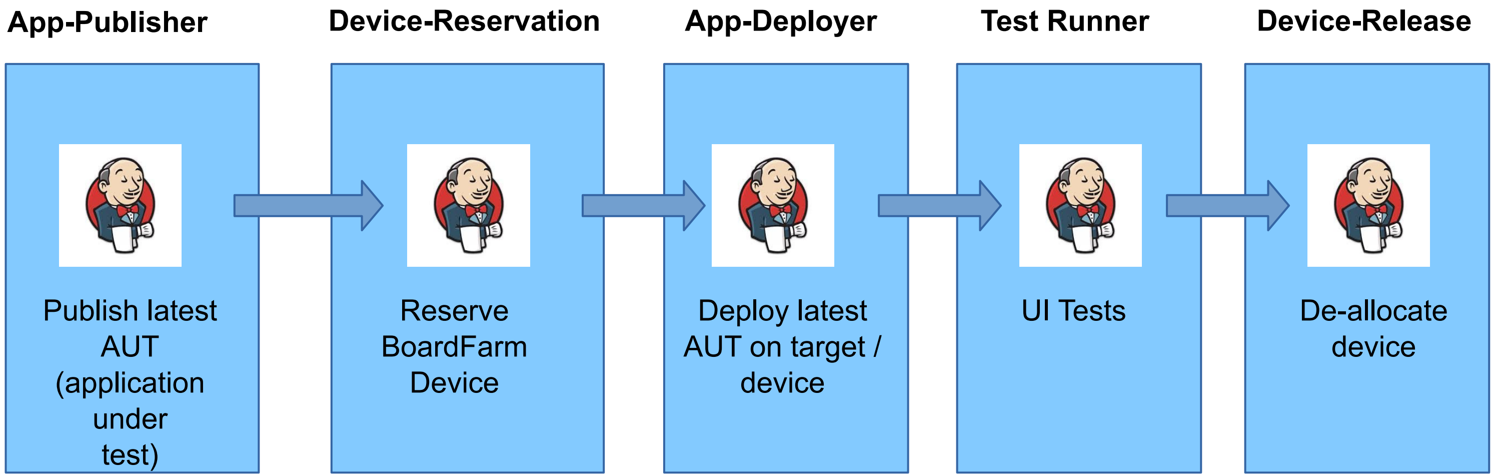
Extension of REST API to build infrastructure independent CD/CT pipelines

2022 work

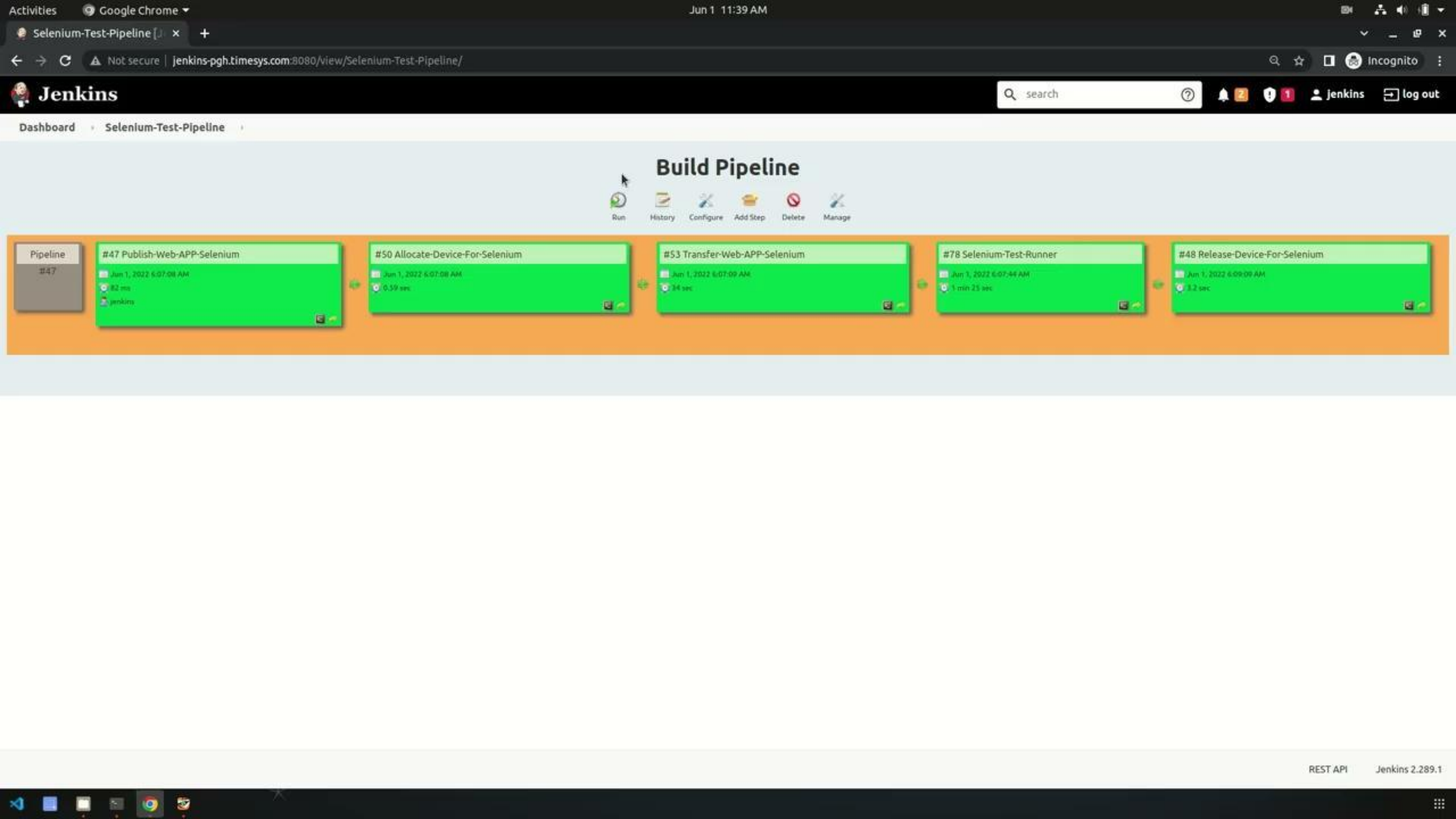
Environment Setup - WebAPP



Web UI CT Pipeline



Video of Web UI Pipeline



Build Pipeline

Pipeline #47

#47 Publish-Web-APP-Selenium
Jun 1, 2022 6:07:08 AM
82 ms
jenkins

#50 Allocate-Device-For-Selenium
Jun 1, 2022 6:07:08 AM
0.59 sec

#53 Transfer-Web-APP-Selenium
Jun 1, 2022 6:07:09 AM
34 sec

#78 Selenium-Test-Runner
Jun 1, 2022 6:07:44 AM
1 min 25 sec

#48 Release-Device-For-Selenium
Jun 1, 2022 6:09:09 AM
3.2 sec

Test Runner

- **Web UI Automation Tool - Selenium**
  - Web Browser - Google Chrome (101.0.4951.54)
  - Webdriver - Chrome
- **Test Framework - pytest**
  - Test result format - JUnit
- **CI Tool - Jenkins**
- **Jenkins**
  - Test results visualization plugin - JUnit
  - Pipeline view plugin - build-pipeline

Pipeline - REST APIs mapping

	Job	CLI command	REST API
01	Device Reservation	ebf <Device Name> info ebf <Device Name> allocate	https://api/devices/<DeviceName>/ https://api/devices/<DeviceName>/assign/
02	App Deployer	ebf <Device Name> power reboot ebf <Device Name> ssh run ebf <Device Name> ssh upload	https://api/devices/<DeviceName>/power/reboot/ https://api/devices/<DeviceName>/ssh/run/ https://api/devices/<DeviceName>/upload/ssh/
03	Test Runner	ebf <Device Name> ssh run *ebf <Device Name> portfw add	https://api/devices/<DeviceName>/ssh/run/ https://api/devices/<DeviceName>/portfw/nat/
04	Device Release	*ebf <Device Name> portfw delete ebf <Device Name> ssh run ebf <Device Name> power off ebf <Device Name> release	https://api/devices/<DeviceName>/portfw/nat/ https://api/devices/<DeviceName>/ssh/run/ https://api/devices/<DeviceName>/power/off/ https://api/devices/<DeviceName>/release/

EBF CLI is implemented using the REST API

** Required in Timesys EBF since DUTs are connected in Zombie's private network*

CD /CT Pipeline

- **Demonstrated use of Jenkins pipeline for device provisioning and Test execution**
- **Currently this pipeline works on Timesys Boardfarm Infrastructure Only (Which had additional Port forwarding steps)**
    - Integration with Fuego- Jenkins is in progress

What's next?

What's next

- **Promote the use of the API and implementations**
- **Add APIs**
  - Keyboard/Mouse API (like teensy simulating USB keyboard, or like VNC)
  - CAN bus, USB, and other buses
  - Provisioning APIs (installation and board bring-up)
- **Windows Client for CLI tools**
- **Integrate with more existing test suites**
- **LTP and kselftest**

What's next (cont.)

- **Integration with additional test frameworks**
- **CD /CT Pipelines**
  - Run the same pipeline on different Boardfarm infra. like Fuego, labgrid, gitlab
  - Create pipelines for other types of testing
    - e.g. System Testing, desktop app testing, desktop UI testing
  - Support & Verify different boot and deployment mechanisms in the pipelines
    - e.g. u-boot / fastboot /pxe, nfs/tftp/sdcard/usb
- **Use for more production testing**
  - More real-world testing, especially for new APIs, to help refine them

Questions or comments?

Embedded Linux Conference