Apertis integration testing with LAVA

Contents

LAVA[1] is a testing system allowing the deployment of operating systems to physical and virtual devices, sharing access to devices between developers. As a rule tests are started in non-interactive unattended mode and LAVA provides logs and results in a human-readable form for analysis.

As a common part of the development cycle we need to do some integration testing of the application and validate it's behavior on different hardware and software platforms. LAVA provides the ability for Apertis to share a pool of test devices, ensuring good utilization of these resources in addition to providing automated testing.

Integration testing example

Let's take the `systemd` service and `systemctl` CLI tool as an example to illustrate how to test an application with a D-Bus interface.

The goal could be defined as follows:

> As a developer of the `systemctl` CLI tool, I want to ensure that `systemctl` is able to provide correct information about the system state.

Local testing

To simplify the guide we are testing only the status of `systemd` with the command below:

```
$ systemctl is-system-running
running
```

It doesn't matter if `systemctl` is reporting some other status, `degraded` for instance. The goal is to validate if `systemctl` is able to provide a proper status, rather than to check the `systemd` status itself.

To ensure that the `systemctl` tool is providing the correct information we may check the system state additionally via the `systemd` D-Bus interface:

```
$ gdbus call --system --dest=org.freedesktop.systemd1 --object-path "/org/freedesktop/systemd1" --method org.freedesktop.DBus.Properties.Get org.freedesktop.systemd1.Manager SystemState
(<'running'>,)
```

[1] https://www.lavasoftware.org/

So, for local testing during development we are able to create a simple script validating that `systemctl` works well in our development environment:

```
#!/bin/sh

status=$(systemctl is-system-running)

gdbus call --system --dest=org.freedesktop.systemd1 \
  --object-path "/org/freedesktop/systemd1" \
 --method org.freedesktop.DBus.Properties.Get org.freedesktop.systemd1.Manager SystemState | \
  grep "${status}"

if [ $? -eq 0 ]; then
  echo "systemctl is working"
else
  echo "systemctl is not working"
fi
```

Testing in LAVA

As soon as we are done with development, we push all changes to GitLab and CI will prepare a new version of the package and OS images. But we do not know if the updated version of `systemctl` is working well for all supported devices and OS variants, so we want to have the integration test to be run by LAVA.

Since the LAVA is a part of CI and works in non-interactive unattended mode we can't use the test script above as is.

To start the test with LAVA automation we need to:

1. Adopt the script for LAVA
2. Integrate the testing script into Apertis LAVA CI

Changes in testing script

The script above is not suitable for unattended testing in LAVA due some issues:

- LAVA relies on exit code to determine if test a passed or not. The example above always return the `success` code, only a human-readable string printed by the script provides an indication of the status of the test
- if `systemctl is-system-running` call fails for some other reason (with a segfault for instance), the script will proceed further without that error being detected and LAVA will set the test as passed, so we will have a false positive result
- LAVA is able to report separately for any part of the test suite – just need to use LAVA-friendly output pattern

So, more sophisticated script suitable both for local and unattended testing in LAVA could be the following:

```
#!/bin/sh

# Test if systemctl is not crashed
testname="test-systemctl-crash"
status=$(systemctl is-system-running)
if [ $? -le 4 ]; then
  echo "${testname}: pass"
else
  echo "${testname}: fail"
  exit 1
fi

# Test if systemctl return non-empty string
testname="test-systemctl-value"
if [ -n "$status" ]; then
  echo "${testname}: pass"
else
  echo "${testname}: fail"
  exit 1
fi

# Test if systemctl is reporting the same status as
# systemd exposing via D-Bus
testname="test-systemctl-dbus-status"
gdbus call --system --dest=org.freedesktop.systemd1 \
  --object-path "/org/freedesktop/systemd1" \
  --method org.freedesktop.DBus.Properties.Get \
  org.freedesktop.systemd1.Manager SystemState | \
  grep "${status}"
if [ $? -eq 0 ]; then
  echo "${testname}: pass"
else
  echo "${testname}: fail"
  exit 1
fi
```

Now the script is ready for adding into LAVA testing. Pay attention to output format which will be used by LAVA to detect separate tests from our single script. The exit code from the testing script must be non-zero to indicate the test suite failure.

The script above is available in the Apertis GitLab example repository[2].

[2] https://gitlab.apertis.org/sample-applications/test-systemctl

Create GIT repository for the test suite

We assume the developer is already familiar with GIT version control system[3] and has an account for the Apertis GitLab[4] as described in the Development Process guide[5]

The test script must be accessible by LAVA for downloading. LAVA has support for several methods for downloading but for Apertis the GIT fetch is preferable since we are using separate versions of test scripts for each release.

It is strongly recommended to create a separate repository with test scripts and tools for each single test suite.

As a first step we need a fresh and empty GIT repository somewhere (for example in your personal space of the GitLab instance) which needs to be cloned locally:

```
git clone git@gitlab.apertis.org:d4s/test-systemctl.git
cd test-systemctl
```

By default the branch name is set to `main` but Apertis automation require to use the branch name aimed at a selected release (for instance `apertis/v2022dev1`), so need to create it:

```
git checkout HEAD -b apertis/v2022dev1
```

Copy your script into GIT repository, commit and push it into GitLab:

```
chmod a+x test-systemctl.sh
git add test-systemctl.sh
git commit -s -m "Add test script" test-systemctl.sh
git push -u origin apertis/v2022dev1
```

Add the test into Apertis LAVA CI

Apertis test automation could be found in the GIT repository for Apertis test cases[6], so we need to fetch a local copy and create a work branch `wip/example` for our changes:

```
git clone git@gitlab.apertis.org:tests/apertis-test-cases.git
cd apertis-test-cases
git checkout HEAD -b wip/example
```

1. Create test case description

   First of all we need to create the instruction for LAVA with following information:

   - where to get the test

[3] https://martyn.pages.apertis.org/apertis-website/guides/version_control/
[4] https://gitlab.apertis.org/
[5] https://martyn.pages.apertis.org/apertis-website/guides/development_process/
[6] https://gitlab.apertis.org/tests/apertis-test-cases

- how to run the test

Create the test case file `test-cases/test-systemctl.yaml` with your favorite editor:

```
metadata:
  name: test-systemctl
  format: "Apertis Test Definition 1.0"
  image-types:
    minimal:  [ armhf, arm64, amd64 ]
  image-deployment:
    - OSTree
  type: functional
  exec-type: automated
  priority: medium
  maintainer: "Apertis Project"
  description: "Test the systemctl."

  expected:
    - "The output should show pass."

install:
  git-repos:
    - url: https://gitlab.apertis.org/d4s/test-systemctl.git
      branch: apertis/v2022dev1

run:
  steps:
    - "# Enter test directory:"
    - cd test-systemctl
    - "# Execute the following command:"
    - lava-test-case test-systemctl --shell ./test-systemctl.sh

parse:
  pattern: "(?P<test_case_id>.*):\\s+(?P<result>(pass|fail))"

```

This test is aimed to be run for an ostree-based minimal Apertis image for all supported architectures. However the metadata is mostly needed for documentation purposes.

Action “install” points to the GIT repository as a source for the test, so LAVA will fetch and deploy this repository for us.

Action “run” provides the step-by-step instructions on how to execute the

test. Please note that it is recommended to use wrapper for the test for integration with LAVA.

Action "parse" provides its own detection for the status of test results printed by script.

The test case is available in the examples repository[7].

2. Push the test case to the GIT repository.

   This step is mandatory since the test case would be checked out by LAVA internally during the test preparation.

```
git add test-cases/test-systemctl.yaml
git commit -s -m "add test case for systemctl" test-cases/test-systemctl.yaml
git push --set-upstream origin wip/example
```

3. Add a job template to be run in lava. Job template contains all needed information for LAVA how to boot the target device and deploy the OS image onto it.

   Create the simple template `lava/test-systemctl-tpl.yaml` with your lovely editor:

```
job_name: systemctl test on {{release_version}} {{pretty}} {{image_date}}
{% if device_type == 'qemu' %}
{% include 'common-qemu-boot-tpl.yaml' %}
{% else %}
{% include 'common-boot-tpl.yaml' %}
{% endif %}

  - test:
      timeout:
        minutes: 15
      namespace: system
      name: common-tests
      definitions:
          - repository: https://gitlab.apertis.org/tests/apertis-test-cases.git
            revision: 'wip/example'
            from: git
            path: test-cases/test-systemctl.yaml
            name: test-systemctl
```

   Hopefully you don't need to deal with the HW-related part, boot and deploy since we already have those instructions for all supported boards

[7] https://gitlab.apertis.org/sample-applications/test-systemctl/-/blob/main/test-cases/test-systemctl.yaml

and Apertis OS images. See common boot template[8] for instance.

Please pay attention to `revision` – it must point to your development branch while you are working on your test.

Instead of creating a new template, you may want to extend the appropriate existing template with additional test definition. In this case the next step could be omitted.

The template above could be found in repository with examples[9].

4. Add the template into a profile.

   Profile file is mapping test jobs to devices under the test. So you need to add your job template into the proper list. For example we may extend the templates list named `templates-minimal-ostree` in file `lava/profiles.yaml`:

   ```
   .templates:
     - &templates-minimal-ostree
       - test-systemctl-tpl.yaml
   ```

   It is highly recommended to temporarily remove or comment out the rest of templates from the list to avoid unnecessary workload on LAVA while you're developing the test.

5. Configure and test the `lqa` tool

   For interaction with LAVA you need to have the `lqa` tool installed and configured as described in LQA[10] tutorial.

   The tool is pretty easy to install in the Apertis SDK:

   ```
   $ sudo apt-get update
   $ sudo apt-get install -y lqa
   ```

   To configure the tool you need to create file `~/.config/lqa.yaml` with the following authentication information:

   ```
   user: '<REPLACE_THIS_WITH_YOUR_LAVA_USERNAME>'
   auth-token: '<REPLACE_THIS_WITH_YOUR_AUTH_TOKEN>'
   server: 'https://lava.collabora.co.uk/'
   ```

   where `user` is your login name for LAVA and `auth-token` must be obtained from LAVA API: https://lava.collabora.co.uk/api/tokens/

   To test the setup just run command below, if the configuration is correct you should see your LAVA login name:

[8] https://gitlab.apertis.org/tests/apertis-test-cases/-/blob/apertis/v2022dev1/lava/common-boot-tpl.yaml

[9] https://gitlab.apertis.org/sample-applications/test-systemctl/-/blob/main/lava/hello-world-tpl.yaml

[10] https://martyn.pages.apertis.org/apertis-website/qa/lqa/

```
$ lqa whoami
d4s
```

6. Check the profile and template locally first.

   As a first step you have to define the proper profile name to use for the test in LAVA.

   Since LAVA is a part of Apertis OS CI, it requires some variables to be provided for using Apertis profiles and templates. Let's define the board we will use for testing, as well as the image release and variant:

```
release=v2022dev1
version=v2022dev1.0rc2
variant=minimal
arch=armhf
board=uboot
baseurl="https://images.apertis.org"
imgpath="release/$release"
profile_name=apertis_ostree-${variant}-${arch}-${board}
image_name=apertis_ostree_${release}-${variant}-${arch}-${board}_${version}
```

   And now we are able to submit the test in a dry-run mode:

```
lqa submit -g lava/profiles.yaml -p ${profile_name} \
  -t visibility:"{'group': ['Apertis']}" -t priority:"high" \
  -t imgpath:${imgpath} -t release:${release} -t image_date:${version} \
  -t image_name:${image_name} -n
```

   There should not be any error or warning from `lqa`. You may want to add `-v` argument to see the generated LAVA job.

   It is recommended to set `visibility` variable to "Apertis" group during development to avoid any credentials/passwords leak by occasion. Set the additional variable `priority` to `high` allows you to bypass the jobs common queue if you do not want to wait for your job results for ages.

7. Submit your first job to LAVA.

   Just repeat the `lqa` call above without the `-n` option. After the job submission you will see the job ID:

```
$ lqa submit -g lava/profiles.yaml -p "${profile_name}" -t visibility:"{'group': ['Apertis']}" -
t  priority:"high"  -t  imgpath:${imgpath}  -t  release:${release}  -
t image_date:${version} -t image_name:${image_name}
Submitted job test-systemctl-tpl.yaml with id 3463731
```

   It is possible to check the job status by URL with the ID returned by the above command: https://lava.collabora.co.uk/scheduler/job/3463731

   The `lqa` tool generates the test job from local files, so you don't need to push your changes to GIT until your test job is working as designed.

8. Push your template and profile changes.

   Once your test case works as expected you should restore all commented templates for profile, change the `revision` key in file `lava/test-systemctl-tpl.yaml` to a suitable target branch and submit your changes:

   ```
   git add lava/test-systemctl-tpl.yaml lava/profiles.yaml
   git commit -a -m "hello world template added"
   git push
   ```

As a last step you need to create a merge request in GitLab. As soon as it gets accepted your test becomes part of Apertis testing CI.