



Apertis integration testing with LAVA

1 Contents

2	Integration testing example	2
3	Local testing	2
4	Testing in LAVA	3
5	Changes in testing script	3
6	Create GIT repository for the test suite	4
7	Add the test into Apertis LAVA CI	5

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

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

17 Integration testing example

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

20 The goal could be defined as follows:

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

24 Local testing

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

```
27 $ systemctl is-system-running
28 running
```

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

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

```
34 $ gdbus call --system --dest=org.freedesktop.systemd1 --object-path "/org/freedesktop/systemd1" --
35 -method org.freedesktop.DBus.Properties.Get org.freedesktop.systemd1.Manager SystemState
36 (<'running'>,)
```

¹<https://www.lavasoftware.org/>

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

```
39 #!/bin/sh
40
41 status=$(systemctl is-system-running)
42
43 gdbus call --system --dest=org.freedesktop.systemd1 \
44   --object-path "/org/freedesktop/systemd1" \
45   --method org.freedesktop.DBus.Properties.Get org.freedesktop.systemd1.Manager SystemState | \
46   grep "${status}"
47
48 if [ $? -eq 0 ]; then
49   echo "systemctl is working"
50 else
51   echo "systemctl is not working"
52 fi
```

53 Testing in LAVA

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

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

60 To start the test with LAVA automation we need to:

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

63 Changes in testing script

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

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

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

```

76 #!/bin/sh
77
78 # Test if systemctl is not crashed
79 testname="test-systemctl-crash"
80 status=$(systemctl is-system-running)
81 if [ $? -le 4 ]; then
82     echo "${testname}: pass"
83 else
84     echo "${testname}: fail"
85     exit 1
86 fi
87
88 # Test if systemctl return non-empty string
89 testname="test-systemctl-value"
90 if [ -n "$status" ]; then
91     echo "${testname}: pass"
92 else
93     echo "${testname}: fail"
94     exit 1
95 fi
96
97 # Test if systemctl is reporting the same status as
98 # systemd exposing via D-Bus
99 testname="test-systemctl-dbus-status"
100 gdbus call --system --dest=org.freedesktop.systemd1 \
101     --object-path "/org/freedesktop/systemd1" \
102     --method org.freedesktop.DBus.Properties.Get \
103     org.freedesktop.systemd1.Manager SystemState | \
104     grep "${status}"
105 if [ $? -eq 0 ]; then
106     echo "${testname}: pass"
107 else
108     echo "${testname}: fail"
109     exit 1
110 fi

```

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

115 The script above is available in the Apertis GitLab [example repository](https://gitlab.apertis.org/sample-applications/test-systemctl)².

²<https://gitlab.apertis.org/sample-applications/test-systemctl>

116 **Create GIT repository for the test suite**

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

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

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

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

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

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

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

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

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

138 **Add the test into Apertis LAVA CI**

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

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

145 1. Create test case description

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

- 148 • where to get the test

³https://jwd.pages.apertis.org/apertis-website/guides/version_control/

⁴<https://gitlab.apertis.org/>

⁵https://jwd.pages.apertis.org/apertis-website/guides/development_process/

⁶<https://gitlab.apertis.org/tests/apertis-test-cases>

149 • how to run the test

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

```
1  metadata:
2    name: test-systemctl
3    format: "Apertis Test Definition 1.0"
4    image-types:
5      minimal: [ armhf, arm64, amd64 ]
6    image-deployment:
7      - OSTree
8    type: functional
9    exec-type: automated
10   priority: medium
11   maintainer: "Apertis Project"
12   description: "Test the systemctl."
13
14   expected:
15     - "The output should show pass."
16
17   install:
18     git-repos:
19       - url: https://gitlab.apertis.org/d4s/test-systemctl.git
20         branch: apertis/v2022dev1
21
22   run:
23     steps:
24       - "# Enter test directory:"
25       - cd test-systemctl
26       - "# Execute the following command:"
27       - lava-test-case test-systemctl --shell ./test-systemctl.sh
28
29   parse:
30     pattern: "(?P<test_case_id>.*):\s+(?P<result>(pass|fail))"
31
```

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

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

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

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

160 Action “parse” provides its own detection for the status of test results
161 printed by script.

162 The test case is available in the [examples repository](#)⁷.

163 2. Push the test case to the GIT repository.

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

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

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

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

```
175 job_name: systemctl test on {{release_version}} {{pretty}} {{image_date}}  
176 {% if device_type == 'qemu' %}  
177 {% include 'common-qemu-boot-tpl.yaml' %}  
178 {% else %}  
179 {% include 'common-boot-tpl.yaml' %}  
180 {% endif %}  
  
181  
182 - test:  
183     timeout:  
184         minutes: 15  
185     namespace: system  
186     name: common-tests  
187     definitions:  
188         - repository: https://gitlab.apertis.org/tests/apertis-test-  
189 cases.git  
190         revision: 'wip/example'  
191         from: git  
192         path: test-cases/test-systemctl.yaml  
193         name: test-systemctl
```

194 Hopefully you don’t need to deal with the HW-related part, boot and
195 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>

196 and Apertis OS images. See [common boot template](#)⁸ for instance.

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

199 Instead of creating a new template, you may want to extend the appropri-
200 ate existing template with additional test definition. In this case the next
201 step could be omitted.

202 The template above could be found in [repository with examples](#)⁹.

203 4. Add the template into a profile.

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

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

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

213 5. Configure and test the `lqa` tool

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

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

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

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

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

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

226 To test the setup just run command below, if the configuration is correct
227 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://jwd.pages.apertis.org/apertis-website/qa/lqa/>


```
228 $ lqa whoami
229 d4s
```

230 6. Check the profile and template locally first.

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

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

```
236 release=v2022dev1
237 version=v2022dev1.0rc2
238 variant=minimal
239 arch=armhf
240 board=uboot
241 baseurl="https://images.apertis.org"
242 imgpath="release/${release}"
243 profile_name=apertis_ostree-${variant}-${arch}-${board}
244 image_name=apertis_ostree_${release}-${variant}-${arch}-${board}_${version}
```

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

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

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

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

256 7. Submit your first job to LAVA.

257 Just repeat the `lqa` call above without the `-n` option. After the job sub-
258 mission you will see the job ID:

```
259 $ lqa submit -g lava/profiles.yaml -p "${profile_name}" -t visibility:"{'group': ['Apertis']}" -
260   t priority:"high" -t imgpath:${imgpath} -t release:${release} -
261   t image_date:${version} -t image_name:${image_name}
262 Submitted job test-systemctl-tpl.yaml with id 3463731
```

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

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

267 8. Push your template and profile changes.

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

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

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