



Test case dependencies on immutable roots

1 **Contents**

2 **Possible solutions** **2**

3 **Overview of applicable approach** **2**

4 Rework tests to ship their dependencies in ‘/var/lib/tests’ 2

5 Pros: 2

6 Cons: 3

7 OSTree branch or static deltas usage 3

8 Pros: 3

9 Cons: 3

10 OSTree overlay 4

11 Pros: 4

12 Cons: 4

13 **Overall proposal** **4**

14 Create separate git repository for each test 5

15 Reduce dependencies 5

16 Make test relocatable 5

17 Immutable root filesystems have several security and maintainability advan-
 18 tages, and avoiding changes to them increases the value of testing as the system
 19 under test would closely match the production setup.

20 This is fundamental for setups that don’t have the ability to install packages at
 21 runtime, like OSTree-based deployments, but it’s largely beneficial for package
 22 based setups as well.

23 To achieve that, tests should then ship their own dependencies in a self-contained
 24 way and not rely on package installation at runtime.

25 **Possible solutions**

26 For adding binaries into OSTree-based system, the following approaches are
 27 possible:

- 28 • Build the tests separately on Jenkins and have them run from
 29 /var/lib/tests;
- 30 • Create a Jenkins job to extract tests from their .deb packages shipped on
 31 OBS and to publish the results, so they can be run from /var/lib/tests;
- 32 • Use layered filesystem for binaries install on top of testing image;
- 33 • Publish a separate OSTree branch for tests created at build time from the
 34 same OS pack as image to test;
- 35 • Produce OSTree static deltas at build time from the same OS pack as
 36 image to test with additional packages/binaries installed;

- 37 • Create mechanism for `dpkg` similar to RPM-OSTree project* to allow in-
38 stallation of additional packages in the same manner as we have for now.
39 – Creation of `dpkg-ostree` project will use a lot of time and human
40 resources due to changes in `dpkg` and `apt` system utilities.

41 Overview of applicable approach

42 Rework tests to ship their dependencies in `‘/var/lib/tests‘`

43 Build the tests separately and have them run from `/var/lib/tests` or create a
44 Jenkins job to extract tests from their `.deb` packages to `/var/lib/tests`

45 Pros:

- 46 • ‘clean’ testing environment – the image is not polluted by additions, so
47 tests and dependencies have no influence on SW installed on image
48 • possibility to install additional packages/binaries in runtime

49 Cons:

- 50 • some binaries/scripts expect to find the dependencies in standard places
51 – additional changes are needed to create the directory with relocated test
52 tools installed
- 53 • we need to be sure if SW from packages works well from relocated directory
- 54 • additional efforts are needed to maintain 2 versions of some packages
55 and/or packaging for some binaries/libraries might be tricky
- 56 • can’t install additional packages without some preparations in a build time
57 (save `dpkg/apt`-related infrastructure or create a tarball from pre-installed
58 SW)
- 59 • possible versions mismatch between SW installed into testing image and
60 SW from tests directory
- 61 • problems in dependencies installation are detected only in runtime

62 OSTree branch or static deltas usage

63 Both approaches are based on native OSTree upgrade/rollback mechanism – only
64 transport differs.

65 Pros:

- 66 • test of OSTree upgrade mechanism is integrated
- 67 • easy to create and maintain branches for different groups of tests – so only
68 SW needed for the group is installed during the tests
- 69 • developer can obtain the same environment as used in LAVA in a few
70 `ostree` commands

- 71 • problems with installation of dependencies for the test are detected in a
- 72 buildtime
- 73 • the original image do not need to have `wget`, `curl` or any other tool for
- 74 download – `ostree` tool have own mechanism for download needed commit
- 75 from test branches
- 76 • with OSTree static deltas we are able to test ‘offline’ upgrades without
- 77 network access
- 78 • saves a lot of disk space for infrastructure due OSTree repository usage

79 **Cons:**

- 80 • ‘dirty’ testing environment – the list of packages is not the same as we
- 81 have in testing image; e.g. system places for binaries and libraries are used
- 82 by additional packages installed, as well as changes in system configura-
- 83 tion might occur (the same behavior we have in current test system with
- 84 installation of additional packages via `apt`)
- 85 • not possible to install additional packages at runtime
- 86 • additional branch(es) should be created at build time
- 87 • reboot is needed to apply the test environment
- 88 • in case of OSTree static deltas – creation of delta is an expensive operation
- 89 in terms of time and resources usage

90 **OSTree overlay**

91 Overlay is a native option provided by `ostree` project, re-mounting “/usr” direc-
 92 tory in R/W mode on top of ‘overlayfs’. This allows to add any software into
 93 “/usr” but changes will disappear just after reboot.

94 **Pros:**

- 95 • limited possibility to install additional packages at runtime (with saved
- 96 state of `dpkg` and `apt`) – merged “/usr” is desirable
- 97 • possibility to copy/unpack prepared binaries directly to “/usr” directory
- 98 • able to use OSTree pull/checkout mechanism to apply overlay

99 **Cons:**

- 100 • dirty testing environment – the list of packages is not the same as we have
- 101 in testing image
- 102 • OSTree branch should contain only “/usr” if used. In other case need to
- 103 use foreign for OSTree methods to store binaries and/or filesystem tree
- 104 • can’t apply additional software without some preparations in a build time
- 105 (save `dpkg/apt`-related infrastructure, create a tarball from pre-installed
- 106 SW or create an ostree branch)
- 107 • possible versions mismatch between SW installed into testing image and
- 108 SW from tests directory
- 109 • problems in dependencies installation are detected only in runtime

110 Overall proposal

111 The proposal consist of a transition from a full apt based test mechanism to a
112 more independant test mechanism.

113 Each tests will be pulled of `apertis-tests` and moved to its own git repository.
114 During the move, the test will be made relocatable, and its dependencies will
115 be reduced.

116 Dependencies that could not be removed would be added to the test itself.

117 At any time, it would still be possible to run the old tests on the non OSTree
118 platform. The new test that have already be transitionned could run on both
119 OSTree and apt platforms.

120 The following steps are envisioned.

121 Create separate git repository for each test

122 In order to run the tests on LAVA, the use of git is recommended. LAVA
123 is already able to pull test definitions from git, but it can pull only one git
124 repository for each test.

125 To satisfy this constraint, each test definition, scripts, and dependencies must
126 be grouped in a single git repository.

127 In order to run the tests manually, GitLab is able to dynamically build a tarball
128 with the content of a git repository at any time. The tarball can be retrieved
129 at a specific URL. By specifying a branch other than master, a release-specific
130 test can be generated. A tool such as `wget` or `curl` can be used, or it might be
131 necessary to download the test tarball from a host, and copy it to the device
132 under test using `scp`.

133 Reduce dependencies

134 To minimize impact of the tests dependencies on the target environment,
135 some dependencies need to be dropped. For example, Python requires several
136 megabytes of binaries and dependencies itself, so all the Python scripts will
137 need to be rewritten using Posix shell scripts or compiled binaries.

138 For tests using data files, the data should be integrated in the git repository.

139 Make test relocatable

140 Most of the tests rely on static path to find binaries. It is straightforward to
141 modify a test to use a custom `PATH` instead of static one. This custom `PATH` would
142 point to a subdirectory in the test repository itself.

143 This applies to dependencies which could be relocated, such as statically linked
144 binaries, scripts, and media files.

¹⁴⁵ For the test components that might not be ported easily, such as For example
¹⁴⁶ AppArmor profiles that are designed to work on binaries at fixed locations, a
¹⁴⁷ case-by-case approach needs to be taken.