Test case dependencies on immutable rootfs

Contents

Test case dependencies on immutable rootfs

Overview

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

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

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

Possible solutions

For adding binaries into OStree-based system, the following approaches are possible:

- Build the tests separately on Jenkins and have them run from `/var/lib/tests`;
- Create a Jenkins job to extract tests from their .deb packages shipped on OBS and to publish the results, so they can be run from `/var/lib/tests`;
- Use layered filesystem for binaries install on top of testing image;
- Publish a separate OStree branch for tests created at build time from the same OS pack as image to test;
- Produce OStree static deltas at build time from the same OS pack as image to test with additional packages/binaries installed;
- Create mechanism for `dpkg` similar to RPM-OStree project* to allow installation of additional packages in the same manner as we have for now.

– Creation of `dpkg-ostree` project will use a lot of time and human resources due to changes in `dpkg` and `apt` system utilities.

Overview of applicable approach

Rework tests to ship their dependencies in '/var/lib/tests'

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

Pros:

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

Cons:

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

OStree branch or static deltas usage

Both approaches are based on native OStree upgrade/rollback mechanism – only transport differs.

Pros:

- test of OStree upgrade mechanism is integrated
- easy to create and maintain branches for different groups of tests – so only SW needed for the group is installed during the tests
- developer can obtain the same environment as used in LAVA in a few `ostree` commands
- problems with installation of dependencies for the test are detected in a buildtime
- the original image do not need to have `wget`, `curl` or any other tool for download – `ostree` tool have own mechanism for download needed commit from test branches

- with OStree static deltas we are able to test 'offline' upgrades without network access
- saves a lot of disk space for infrastructure due OStree repository usage

Cons:

- 'dirty' testing environment – the list of packages is not the same as we have in testing image; e.g. system places for binaries and libraries are used by additional packages installed, as well as changes in system configuration might occur (the same behavior we have in current test system with installation of additional packages via `apt`)
- not possible to install additional packages at runtime
- additional branch(es) should be created at build time
- reboot is needed to apply the test environment
- in case of OStree static deltas – creation of delta is an expensive operation in terms of time and resources usage

OStree overlay

Overlay is a native option provided by `ostree` project, re-mounting "/usr" directory in R/W mode on top of 'overlayfs'. This allows to add any software into "/usr" but changes will disappear just after reboot.

Pros:

- limited possibility to install additional packages at runtime (with saved state of `dpkg` and `apt`) – merged "/usr" is desirable
- possibility to copy/unpack prepared binaries directly to "/usr" directory
- able to use OStree pull/checkout mechanism to apply overlay

Cons:

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

Overall proposal

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

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

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

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

The following steps are envisioned.

Create separate git repository for each test

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

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

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

Reduce dependencies

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

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

Make test relocatable

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

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

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