avocado Documentation

Release 0

Lucas Meneghel Rodrigues

August 05, 2016

Contents

Contents:

CHAPTER 1

Introduction

Avocado-virt is a plugin for the Avocado Test Framework. It aims to provide libraries and extra functionality necessary to run virtualization tests on Linux. We started with KVM/QEMU, but we're certainly open to expand the coverage to things like Xen and libvirt.

CHAPTER 2

Getting Started

The first step towards using Avocado-Virt is, quite obviously, installing it.

2.1 Installing Avocado

Start by following the instructions on this link.

2.2 Installing Avocado-Virt

The official source for avocado-virt is the GIT repository host at **'GitHub <https://gitub.com/avocado-framework/avocado-virt'_**. You can clone it by running:

```
$ git clone https://gitub.com/avocado-framework/avocado-virt
```

Then install `avocado-virt` itself with:

```
$ cd avocado-virt
$ python setup.py install
```

You may want to use `python setup.py install --user` to install locally or even `python setup.py develop --user` to run from the source tree.

2.2.1 Bootstrapping Avocado-Virt

After the package, a bootstrap process must be run wit the *vt-bootstrap* command. Example:

```
$ avocado virt-bootstrap
```

The output should be similar to:

```
Probing your system for test requirements
7zip present
Verifying expected SHA1 sum from http://assets-avocadoproject.rhcloud.com/static/SHA1SUM_JEOS23
Expected SHA1 sum: 177468b8e5fcb7b9c5982a6bc21ff45df6d80b2f
Compressed JeOS image found in /home/<user>/avocado/data/images/jeos-23-64.qcow2.7z, with proper SHA1
Uncompressing the JeOS image to restore pristine state. Please wait...
Successfully uncompressed the image
Your system appears to be all set to execute tests
```

Another addition you'll notice is that the avocado subcommand `run` now has extra parameters that you can pass:

```
$ avocado run -h
...
virtualization testing arguments:
  --qemu-bin QEMU_BIN   Path to a custom qemu binary to be tested. Current
                        path: /bin/qemu-kvm
  --qemu-dst-bin QEMU_DST_BIN
                        Path to a destination qemu binary to be tested. Used
                        as incoming qemu in migration tests. Current path:
                        /bin/qemu-kvm
  --qemu-img-bin QEMU_IMG_BIN
                        Path to a custom qemu-img binary to be tested. Current
                        path: /bin/qemu-img
  --qemu-io-bin QEMU_IO_BIN
                        Path to a custom qemu-io binary to be tested. Current
                        path: /bin/qemu-io
  --guest-image-path GUEST_IMAGE_PATH
                        Path to a guest image to be used in tests. Current
                        path: /home/<user>/avocado/data/images/jeos-23-64.qcow2
  --guest-user GUEST_USER
                        User that avocado should use for remote logins.
                        Current: root
  --guest-password GUEST_PASSWORD
                        Password for the user avocado should use for remote
                        logins. You may omit this if SSH keys are setup in the
                        guest. Current: 123456
  --take-screendumps    Take regular QEMU screendumps (PPMs) from VMs under
                        test. Current: False
  --record-videos       Encode videos from VMs under test. Implies --take-
                        screendumps. Current: False
  --qemu-template [QEMU_TEMPLATE]
                        Create qemu command line from a template
```

That's right, the virt plugin gives you new options on the runner specific to the QEMU related tests. For example, you can provide `--qemu-bin` to tell your tests that you want a specific QEMU binary instead of whatever the runner could find looking in the system PATH or environment variables.

Now, after you bootstrapped your tests, you may want to look for some examples on how to build your tests. We have a repo with example virtualization tests in `https://github.com/avocado-framework/avocado-virt-tests.git`. Cloning this repo will allow you to run the example tests and study them:

```
$ git clone https://github.com/avocado-framework/avocado-virt-tests.git
Cloning into 'avocado-virt-tests'...
remote: Counting objects: 15, done.
remote: Total 15 (delta 0), reused 0 (delta 0)
Unpacking objects: 100% (15/15), done.
Checking connectivity... done.
$ cd avocado-virt-tests/
$ avocado run qemu/boot.py
JOB ID     : <id>
JOB LOG    : /home/<user>/avocado/job-results/job-<timestamp-shortid>/job.log
TESTS      : 1
(1/1) qemu/boot.py:BootTest.test_boot: PASS (23.13 s)
RESULTS    : PASS 1 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0
JOB HTML   : /home/<user>/avocado/job-results/job-<timestamp-shortid>/html/results.html
TIME       : 23.13 s
```

With this info, we are covering the basics. We'll cover setup details and the available test API in later sessions.

CHAPTER 3

Guest Configuration

By default, avocado-virt uses an x86_64 minimal guest image based on the latest stable version of Fedora available at a given time. The image file is a compressed qcow2 image located on my image repository that is downloaded, should you choose to run the sub command `virt-bootstrap`.

If you use avocado with default settings, the test runner is going to uncompress the pristine image of this so-called JeOS before each test. You might change this behavior in config `virt.restore.disable_for_{test|job}` (`/etc/avocado/conf.d/virt.conf`) or via multiplexer params `disable_restore_image_{test|job}` in `/plugins/virt/guest/` namespace if you want to completely skip the backup restore process.

Or, you may opt for using your own guest image in your tests.

CHAPTER 4

Guest Requirements

The JeOS is a fairly small guest, so your guest should be generally fine, as long as it does have open SSH running on port 22 after boot, for all the tests that require SSH connections (that is, tests that at some point call the VM method `.login_remote()`. That said, it is hard to keep requirements documented with precision, given that the tests and the plugin are going to evolve in scope and features. Please feel free to send us patches to this documentation file to correct any inaccuracies.

CHAPTER 5

Using your own image

You can use your own image by specifying the following options:

- `--guest-image-path` - You can provide this option with an arbitrary path to a QEMU disk image file with your guest. You can use any of the file formats specified, such as qcow2, qed or even raw image formats.
- `--guest-user` - If your image has a specific user set up previously that you want avocado to use when logging into the remote guest, please provide this option. Avocado will inform the default values used in the `avocado run --help` output.
- `--guest-password` - If your image has a specific password for the user set up previously that you want avocado to use when logging into the remote guest, please provide this option. Avocado will inform the default values used in the `avocado run --help` output. Note that a previous setup of ssh keys on that guest can let you ignore that option entirely.

Next, we'll learn how to write a simple test, using the avocado basic APIs.

CHAPTER 6

Writing Avocado Virt Tests

6.1 Basic example: Boot test

Avocado virt tests are similar to non-virt ones, they only differ on that they use some specialized libraries, that let you use special virt features.

Here's an example of a basic virt testing, a test that starts QEMU with a guest image, then it'll try to establish an ssh connection to this guest:

```
from avocado.virt import test


class BootTest(test.VirtTest):

    def test_boot(self):
        self.vm.power_on()
        self.vm.login_remote()

    def tearDown(self):
        if self.vm:
            self.vm.power_off()
```

The base class for the test is `avocado.virt.test.VirtTest` instead of the base `avocado.test`. The reason for this is that the `VirtTest` class can make the params from the test runner available for tests, and provide other convenience methods for your tests.

If you chose to not override or extend the default virt test `setUp()` method, you'll have at your disposal a basic vm object in `self.vm`. The VM is not started (powered on) yet, and you need to start it yourself. Calling `self.vm.power_on` starts the QEMU process, then from that point forwards we are just waiting for the VM to be active. The proof that the VM started and the guest OS is healthy is that we can establish a remote session (SSH on linux guests) to it, by using the `login_remote` method. That method is going to wait for a default 60 seconds until the SSH connection is established, and fail in case the connection can't be established.

If we have an SSH connection, all is good, the test passed, and we're going to clean things up as a good practice. The `cleanup` method is going to run a `shutdown` command in the remote connection, and then we proceed to shutting down the VM (end the QEMU process), through the `power_off` method.

If that goes fine as well, the test passed and everybody is happy. We ended our test with PASS. If any of the operations described above FAIL, avocado is going to proceed accordingly and FAIL the test.

6.2 Basic example: Migrate test

Now, what if I want to migrate the state of a QEMU VM to another QEMU process on that very same machine? Here's what a live migration test looks like:

```
from avocado.virt import test


class MigrationTest(test.VirtTest):

    def test_migrate(self):
        self.vm.power_on()
        migration_mode = self.params.get('migration_mode', 'tcp')
        for _ in xrange(self.params.get('migration_iterations', 4)):
            self.vm.migrate(migration_mode)
            self.vm.login_remote()

    def cleanup(self):
        if self.vm:
            self.vm.power_off()
```

Fortunately, most of the migration logic is wrapped up in the method `vm.migrate`. Here we modeled things after the concept of live migration, so you have a single vm object, that when migrated keeps working just as it did work before, with no service interruption (it doesn't care that the VM state was passed on to another QEMU process). The method will clone the command line of the current VM, add the appropriate snippets for incoming migration, start the new process, and call the appropriate `migrate` command in the QMP monitor of the source VM. After it detects the migration is over, we might repeat the process again `migration_iteration` times (here it has the default value of 4).

6.3 More to come

This is a basic guide, as the plugin is in heavy developmemt. Soon we'll have more APIs and cover more cases.

CHAPTER 7

Reference Guide

This guide presents information on the Avocado-virt basic design and its internals.

7.1 Basic avocado-virt params

Avocado-virt uses test params to affect the environment independently on the test code. This can be used to reproduce the same steps on different setup, for example various disk drivers. You can set these via multiplex `YAML` file.

Table of supported params:

Params path	Params key	Description
/plugins/virt/guest/*	disable_restore_image_test	Don't restore the image after each test
/plugins/virt/guest/*	image_path	Path to the guest image
/plugins/virt/guest/*	password	Guest remote login password
/plugins/virt/guest/*	shell_prompt	Regexp of the guest remote command line
/plugins/virt/guest/*	user	Guest remote login name
/plugins/virt/qemu/migrate/*	timeout	Migration timeout
/plugins/virt/qemu/paths/*	qemu_bin	Path to the QEMU executable file
/plugins/virt/qemu/paths/*	qemu_img_bin	Path to the qemu-img executable file
/plugins/virt/qemu/paths/*	qemu_io_bin	Path to the qemu-io executable file
/plugins/virt/qemu/template/*	contents	Template of the QEMU command to be run instead of autogenerated one
/plugins/virt/screendumps/*	enable	Enable screendump service
/plugins/virt/screendumps/*	interval	Interval between screendumps
/plugins/virt/videos/*	enable	Encode screendumps into video after the test
/plugins/virt/videos/*	jpeg_quality	Quality of the screendump image postprocessing

Note: Some of these values can be modified in config files and/or overridden on the command line. To view the setting on your system run `avocado multiplex -s -c` with avocado-virt enabled.

Note: Not all params are used in every run, some of them depends on each other or on features touched in the test

(for example when your test doesn't use qemu-io executing the test with various values makes no sense. Changing the qemu_bin on the other hand makes the test executed on different QEMU versions.)

CHAPTER 8

Indices and tables

- genindex
- modindex
- search