The functional test beast: tame it, bring it home and make it your pet

Cleber Rosa
Sr. Software Engineer
Oct. 26Th, 2018

AGENDA

- Functional Testing Challenges
- QEMU/KVM & libvirt testing background
- How Avocado fits into the picture
- QEMU Status Report
- What’s Next?

Functional Testing Challenges

Complexity

- Unit tests
  - You zoom into a small piece of functionality
  - Mostly disregard everything else
- Functional tests
  - Always consider the bigger picture

Interactions

- Unit tests
  - Machine based, usually using an API
  - Input is usually:
    - hard coded within the test
    - small accompanying data files
- Functional tests
  - Machines and humans alike
  - Humans will often act as “fuzzers”
  - Input is often too large to keep in-tree

Tools and Framework Requirements

- Unit tests
  - Treated as first class citizens
  - Often the same tools on your compiler tool/chain
- Functional tests
  - External tools
  - Dependencies on more external tools
  - Dependencies the environment
  - Most often than not, scripted in-house ad-hoc solutions

QEMU/KVM & libvirt Functional Testing Background

Avocado-VT Installation

- RPM package installation is your best bet
  - Additional repos
  - Large number of dependencies
- Bootstrap:
  - avocado vt-bootstrap --vt-type=[qemu | libvirt | ...]
  - Secondary dependencies check based on “--vt-type”
  - Configuration file generation
  - Test provider download
  - Images download

Avocado-VT – Writing a new test

- Official documentation contains 24 steps:
    - https://avocado-vt.readthedocs.io/en/latest/WritingTests/WritingSimpleTests.html
- Must understand the “Test Provider Layout”:
    - https://avocado-vt.readthedocs.io/en/latest/WritingTests/TestProviders.html
- No clear mapping of source code file to test
- Test is a function called **run()**, makes code reuse a bit more difficult
- Mandatory creation of configuration file pointing to a test
- Too many test parameters influence the test behavior
- No documentation of test parameters

How Avocado fits into the picture

Avocado – Installation & Use

```
$ pip install --user avocado-framework

$ avocado run /path/to/tests
```

Avocado – Writing Tests

- No fuzz, no previous knowledge:
  - chmod +x test
- Python-based tests give you more:
  - Parameter support
  - Advanced logging
  - Accompanying data files
  - A rich set of utility libraries

```
from avocado import Test

class My(Test):
   def test(self):
      do_something()
```

QEMU Status Report

Functional (AKA acceptance) tests

```
$ cd qemu
$ tree tests/acceptance/
tests/acceptance/
├── avocado_qemu
│   └── __init__.py
├── boot_linux_console.py
├── README.rst
├── version.py
└── vnc.py
```

Functional (AKA acceptance) tests

```
$ avocado run tests/acceptance
JOB ID     : 61e6a03699f576a6fd38564a5eb8e66162b1e644
JOB LOG    : /home/cleber/avocado/job-results/job-2018-10-11T00.02-61e6a03/job.log
 (1/6) tests/acceptance/boot_linux_console.py:BootLinuxConsole.test: PASS (2.00 s)
 (2/6) tests/acceptance/version.py:Version.test_qmp_human_info_version: PASS (0.06 s)
 (3/6) tests/acceptance/vnc.py:Vnc.test_no_vnc: PASS (0.05 s)
 (4/6) tests/acceptance/vnc.py:Vnc.test_no_vnc_change_password: PASS (0.05 s)
 (5/6) tests/acceptance/vnc.py:Vnc.test_vnc_change_password_requires_a_password: PASS (0.05 s)
 (6/6) tests/acceptance/vnc.py:Vnc.test_vnc_change_password: PASS (0.05 s)
RESULTS    : PASS 6 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 | CANCEL 0
JOB TIME   : 2.68 s
```

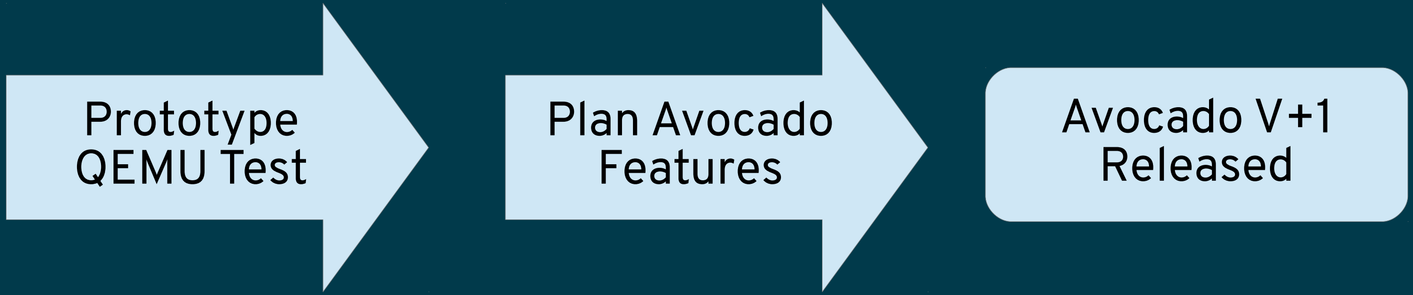
Avocado QEMU tests

- Have access to a predefined “VM”
  - **self.vm**
- The VM is a **QEMUMachine** instance (from **scripts/qemu.py**)
  - Add command line arguments with **add_args()**
  - Launch the VM with **launch()**
  - Send QMP commands with **command()**

Sample QEMU test (version.py)

```
from avocado_qemu import Test
class Version(Test):
    """
    :avocado: enable
    :avocado: tags=quick
    """
    def test_qmp_human_info_version(self):
        self.vm.launch()
        res = self.vm.command('human-monitor-command',
                              command_line='info version')
        self.assertRegexpMatches(res, r'^(\d+\.\d+\.\d)')
```

Avocado + QEMU Development model



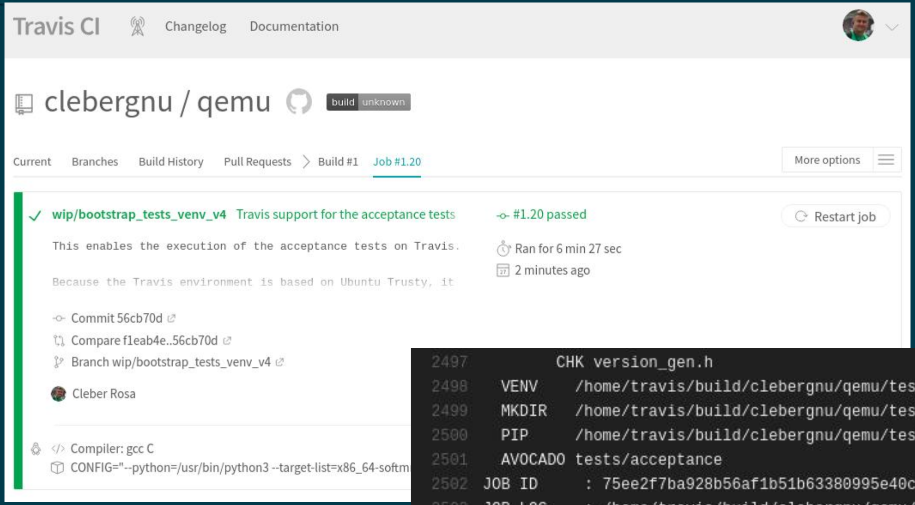
```
$ cd qemu
$ sed -i tests/env-requirements.txt -e ‘s/65.0/66.0/’
$ git add tests/acceptance/new_test.py tests/env-requirements.txt
```

QEMU Status Report
-
Under Development

One command bootstrap and test execution

```
$ make check-acceptance
  VENV    /tmp/qemu-build/tests/venv
  PIP     /home/cleber/src/qemu/tests/venv-requirements.txt
  MKDIR   /tmp/qemu-build/tests/results
  AVOCADO tests/acceptance

$ cat tests/results/latest/results.tap
1..6
ok 1 /home/cleber/src/qemu/tests/acceptance/boot_linux_console.py:BootLinuxConsole.test
ok 2 /home/cleber/src/qemu/tests/acceptance/version.py:Version.test_qmp_human_info_version
ok 3 /home/cleber/src/qemu/tests/acceptance/vnc.py:Vnc.test_no_vnc
ok 4 /home/cleber/src/qemu/tests/acceptance/vnc.py:Vnc.test_no_vnc_change_password
ok 5 /home/cleber/src/qemu/tests/acceptance/vnc.py:Vnc.test_vnc_change_password_requires_a_password
ok 6 /home/cleber/src/qemu/tests/acceptance/vnc.py:Vnc.test_vnc_change_password
```

Travis CI Integration



Travis CI Changelog Documentation

clebergnu / qemu build unknown

Current Branches Build History Pull Requests Build #1 Job #1.20 More options

wip/bootstrap_tests_venv_v4 Travis support for the acceptance tests

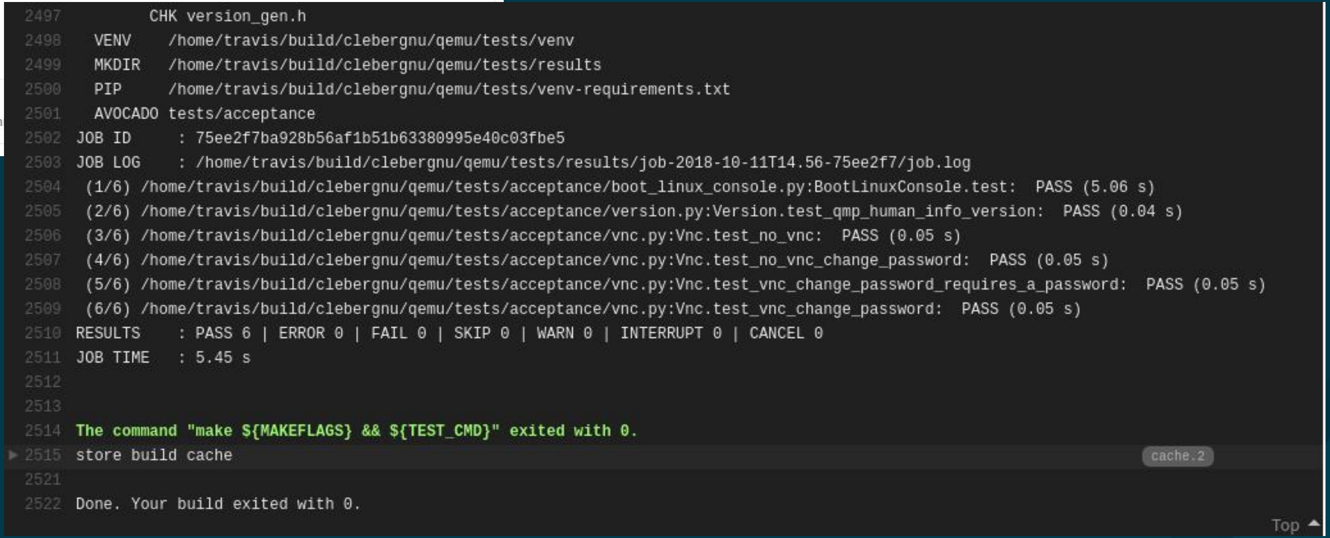
```
This enables the execution of the acceptance tests on Travis.

Because the Travis environment is based on Ubuntu Trusty, it
```

#1.20 passed
Ran for 6 min 27 sec
2 minutes ago

Restart job

Commit 56cb70d
Compare f1eab4e..56cb70d
Branch wip/bootstrap_tests_venv_v4
Cleber Rosa

Compiler: gcc C
CONFIG="--python=/usr/bin/python3 --target-list=x86_64-softm



```
        CHK version_gen.h
  VENV    /home/travis/build/clebergnu/qemu/tests/venv
  MKDIR   /home/travis/build/clebergnu/qemu/tests/results
  PIP     /home/travis/build/clebergnu/qemu/tests/venv-requirements.txt
  AVOCADO tests/acceptance
JOB ID     : 75ee2f7ba928b56af1b51b63380995e40c03fbe5
JOB LOG    : /home/travis/build/clebergnu/qemu/tests/results/job-2018-10-11T14.56-75ee2f7/job.log
 (1/6) /home/travis/build/clebergnu/qemu/tests/acceptance/boot_linux_console.py:BootLinuxConsole.test:  PASS (5.06 s)
 (2/6) /home/travis/build/clebergnu/qemu/tests/acceptance/version.py:Version.test_qmp_human_info_version:  PASS (0.04 s)
 (3/6) /home/travis/build/clebergnu/qemu/tests/acceptance/vnc.py:Vnc.test_no_vnc:  PASS (0.05 s)
 (4/6) /home/travis/build/clebergnu/qemu/tests/acceptance/vnc.py:Vnc.test_no_vnc_change_password:  PASS (0.05 s)
 (5/6) /home/travis/build/clebergnu/qemu/tests/acceptance/vnc.py:Vnc.test_vnc_change_password_requires_a_password:  PASS (0.05 s)
 (6/6) /home/travis/build/clebergnu/qemu/tests/acceptance/vnc.py:Vnc.test_vnc_change_password:  PASS (0.05 s)
RESULTS    : PASS 6 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 | CANCEL 0
JOB TIME   : 5.45 s



The command "make ${MAKEFLAGS} && ${TEST_CMD}" exited with 0.
store build cache                                                                          cache.2

Done. Your build exited with 0.
```

Top

Multi Arch Support

- Many tests:
  - use devices as infrastructure (console, networking, etc)
  - can be reused across different target archs
- Current proposal brings support for:
  - aarch64
  - ppc
  - ppc64
  - s390x
  - x86_64
- https://lists.gnu.org/archive/html/qemu-devel/2018-10/msg01821.html

Linux Guest Boot Test (aka boot_linux.py)

- Based on avocado.utils.vmimage, and supports:
    - Fedora
    - CentOS
    - Debian
    - Ubuntu
    - SUSE
- Automatically downloads and caches the guest image
- Creates a “cloudinit.iso” file
- Waits for successful boot notification from the guest
- https://lists.gnu.org/archive/html/qemu-devel/2018-09/msg02530.html

Linux Guest Boot Test (aka boot_linux.py)

```
class BootLinux(Test):

    def test(self):
        self.vm.set_machine(self.params.get('machine', default='pc'))
        self.vm.add_args('-accel', self.params.get('accel', default='kvm'))
        self.vm.add_args('-smp', self.params.get('smp', default='2'))
        self.vm.add_args('-m', self.params.get('memory', default='4096'))

        arch = self.params.get('arch', default=os.uname()[4])
        distro = self.params.get('distro', default='fedora')
        version = self.params.get('version', default='28')
        boot = vmimage.get(distro, arch=arch, version=version,
                           cache_dir=self.cache_dirs[0],
                           snapshot_dir=self.workdir)
        self.vm.add_args('-drive', 'file=%s' % boot.path)
```

Linux Guest Boot Test (aka boot_linux.py)

```
cloudinit_iso = os.path.join(self.workdir, 'cloudinit.iso')
phone_home_port = network.find_free_port()
cloudinit.iso(cloudinit_iso, self.name,
              # QEMU's hard coded usermode router address
              phone_home_host='10.0.2.2',
              phone_home_port=phone_home_port)
self.vm.add_args('-drive', 'file=%s' % cloudinit_iso)

self.vm.launch()
cloudinit.wait_for_phone_home(('0.0.0.0', phone_home_port), self.name)
```

Guest interaction (aka linux_hw_check.py)

- Prepares a guest for key based SSH authentication
  - reuses qemu/tests/keys/ by default
- Boots a guest
  - similar to previously shown boot_linux.py
  - same Linux distros supported (Fedora, CentOS, Debian, Ubuntu, OpenSUSE)
- Establish SSH session
- Interacts via QMP possible (not done here)
- Verify state/actions on the guest side

Guest interaction (aka linux_hw_check.py)

```
class LinuxHWCheck(Test):
    """
    Boots a Linux system, checking for a successful initialization

    :avocado: enable
    """

    timeout = 600

    def test_hw_resources(self):
        self.set_vm_image()
        self.set_vm_cloudinit()
        ssh_port = network.find_free_port(start_port=self.vm_hw['phone_home_port']+1)
        self.vm.add_session_network(ssh_port)
        self.vm.launch()
        self.wait_for_vm_boot()
```

Guest interaction (aka linux_hw_check.py)

```
priv_key = os.path.join(self.vm_hw['key_path'], 'id_rsa')
with ssh.Session(('127.0.0.1', ssh_port),
                 ('root', priv_key)) as session:
    # cpu
    proc_count_cmd = 'egrep -c "^processor\s\:" /proc/cpuinfo'
    self.assertEqual(int(self.vm_hw['smp']),
                     int(session.cmd(proc_count_cmd).stdout_text.strip()))

    # memory
    match = re.match(r"^MemTotal:\s+(\d+)\skB",
                     session.cmd('cat /proc/meminfo').stdout_text.strip())
    self.assertIsNotNone(match)
    exact_mem_kb = int(self.vm_hw['memory']) * 1024
    guest_mem_kb = int(match.group(1))
    self.assertGreaterEqual(guest_mem_kb, exact_mem_kb * 0.9)
    self.assertLessEqual(guest_mem_kb, exact_mem_kb)
```

What else is hapenning now?

- Guest ABI (machine-type + CPU model) - Eduardo Habkost
- SMP Coverage and corner cases - Wainer Moschetta
- BIOS/OVMF tests – Philippe Mathieu-Daudé

What’s next?

- Migration support
- Test sets:
    - subsystem/maintainer specific
    - Combinatorial Independent Test based
- Regression tests for known fixed issues
- libvirt?
- Whatever the community says

Resources

- Avocado GitHub project:
  - https://github.com/avocado-framework
- Avocado Trello Planning Board:
  - https://trello.com/b/WbqPNl2S/avocado
- Avocado QEMU Trello Planning Board:
  - https://trello.com/b/6Qi1pxVn/avocado-qemu

redhat.

THANK YOU