redhat.

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?

redhat.

Functional Testing Challenges

redhat.

Complexity

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

redhat.

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

redhat.

Tools and Framework Requirements

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

redhat.

QEMU/KVM & libvirt Functional Testing Background

redhat.

Installation is a challenges on “Legacy” Avocado-VT

- 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

redhat.

Writing a test is also a challenge on “Legacy” Avocado-VT

- 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

redhat.

How **Avocado** (but not Avocado-VT) fits into the picture

redhat.

Avocado – Installation & Use

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

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


redhat.

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
  - The chance for finer grained assertions and error messages

```
from avocado import Test

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


redhat.

Avocado QEMU Status Report

redhat.

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
```


redhat.

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
```


redhat.

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 **cmd()** or **command()**

redhat.

Adding a functional test to QEMU

redhat.

Preventing further regressions

```
$ git log --no-merges --oneline --grep=fix
726a2a951b net: cadence_gem: Announce availability of priority queues
ac55d00709 target/arm: Reorg NEON VLD/VST all elements
ea358872a6 hw/arm/boot: Increase compliance with kernel arm64 boot protocol
612aea2017 vnc: Clean up error reporting in vnc_init_func()
a22528b918 numa: Fix QMP command set-numa-node error handling
e1eb292ace vfio: Use warn_report() & friends to report warnings
4b5766488f error: Fix use of error_prepend() with &error_fatal, &error_abort
4c9ab1e693 scripts: Remove check-qerror.sh
...
```


redhat.

```
commit a22528b918c7d29795129b5a64c4cb44bb57a44d
Author: Markus Armbruster <armbru@redhat.com>
Date:   Wed Oct 17 10:26:39 2018 +0200

    numa: Fix QMP command set-numa-node error handling

    Calling error_report() in a function that takes an Error ** argument
    is suspicious.  parse_numa_node() does that, and then exit()s.  It
    also passes &error_fatal to machine_set_cpu_numa_node().  Both wrong.
    Attempting to configure numa when the machine doesn't support it kills
    the VM:

        $ qemu-system-x86_64 -nodefaults -S -display none -M none -preconfig -qmp stdio
        {"QMP": {"version": {"qemu": {"micro": 50, "minor": 0, "major": 3},
                             "package": "v3.0.0-837-gc5e4e49258"}, "capabilities": []}}
        {"execute": "qmp_capabilities"}
        {"return": {}}
        {"execute": "set-numa-node", "arguments": {"type": "node"}}
        NUMA is not supported by this machine-type
        $ echo $?
        1
```


redhat.

```python
from avocado_qemu import Test


class QmpSetNumaNode(Test):
    """
    :avocado: enable
    :avocado: tags=quick,qmp,numa,tests:a22528b918,fixes:7c88e65d9e9
    """
    def test_numa_not_supported(self):
        self.vm.add_args('-nodefaults', '-S', '-preconfig')
        self.vm.set_machine('none')
        self.vm.launch()
        res = self.vm.qmp('set-numa-node', type='node')
        self.assertEqual(res['error']['class'], 'GenericError')
        self.assertEqual(res['error']['desc'], 'NUMA is not supported by this machine-type')
        self.assertTrue(self.vm.is_running())
        self.vm.qmp('x-exit-preconfig')
        self.vm.shutdown()
        self.assertEqual(self.vm.exitcode(), 0)
```


redhat.

QEMU Status Report
-
Under Development

redhat.

Avocado + QEMU Development model


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


redhat.

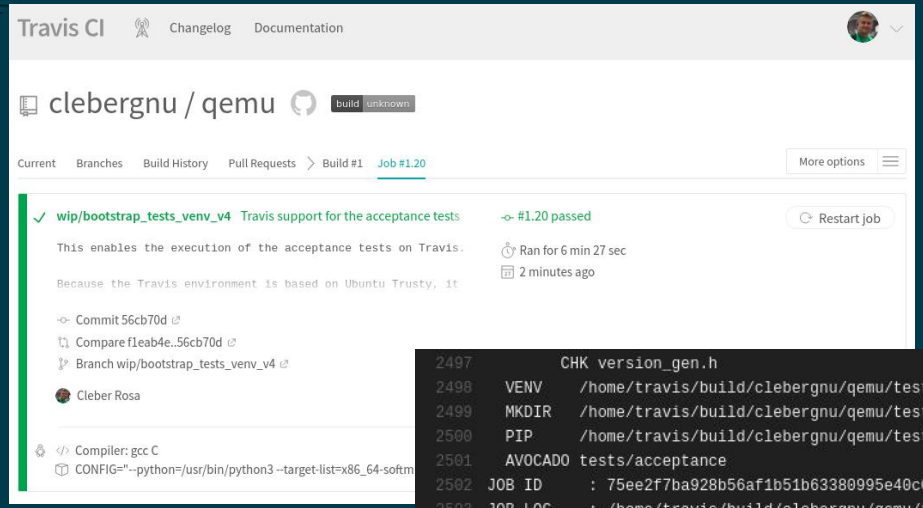
One command bootstrap and test execution

```
$ 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
```


redhat.

Travis CI Integration




```
2497          CHK version_gen.h
2498    VENV    /home/travis/build/clebergnu/qemu/tests/venv
2499    MKDIR   /home/travis/build/clebergnu/qemu/tests/results
2500    PIP     /home/travis/build/clebergnu/qemu/tests/venv-requirements.txt
2501    AVOCADO tests/acceptance
2502  JOB ID     : 75ee2f7ba928b56af1b51b63380995e40c03fbe5
2503  JOB LOG    : /home/travis/build/clebergnu/qemu/tests/results/job-2018-10-11T14.56-75ee2f7/job.log
2504   (1/6) /home/travis/build/clebergnu/qemu/tests/acceptance/boot_linux_console.py:BootLinuxConsole.test:  PASS (5.06 s)
2505   (2/6) /home/travis/build/clebergnu/qemu/tests/acceptance/version.py:Version.test_qmp_human_info_version:  PASS (0.04 s)
2506   (3/6) /home/travis/build/clebergnu/qemu/tests/acceptance/vnc.py:Vnc.test_no_vnc:  PASS (0.05 s)
2507   (4/6) /home/travis/build/clebergnu/qemu/tests/acceptance/vnc.py:Vnc.test_no_vnc_change_password:  PASS (0.05 s)
2508   (5/6) /home/travis/build/clebergnu/qemu/tests/acceptance/vnc.py:Vnc.test_vnc_change_password_requires_a_password:  PASS (0.05 s)
2509   (6/6) /home/travis/build/clebergnu/qemu/tests/acceptance/vnc.py:Vnc.test_vnc_change_password:  PASS (0.05 s)
2510  RESULTS    : PASS 6 | ERROR 0 | FAIL 0 | SKIP 0 | WARN 0 | INTERRUPT 0 | CANCEL 0
2511  JOB TIME   : 5.45 s
2512
2513
2514  The command "make ${MAKEFLAGS} && ${TEST_CMD}" exited with 0.
2515  store build cache                                                          cache.2
2521
2522  Done. Your build exited with 0.
                                                                                    Top
```


redhat.

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

redhat.

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)
```


redhat.

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.log.info("Launching VM")
self.vm.launch()
cloudinit.wait_for_phone_home(('0.0.0.0', phone_home_port), self.name)
self.log.info("Linux Guest booted successfully")
```


redhat.

Multi Arch Support

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

redhat.

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

redhat.

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()
```


redhat.

Guest interaction (aka linux_hw_check.py)

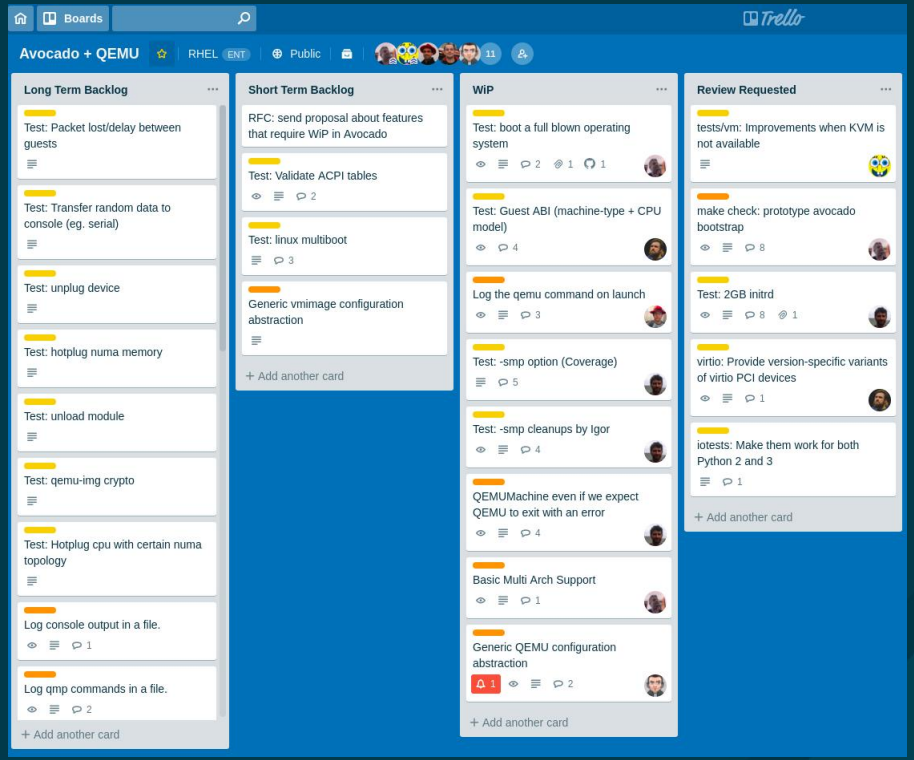
```
priv_key = os.path.join(self.vm_hw['key_path'], 'id_rsa')
with ssh.Session(('127.0.0.1', ssh_port),
                 ('root', priv_key)) as session:
    # cpu
    proc_count_cmd = 'egrep -c "^processor\s\:" /proc/cpuinfo'
    self.assertEqual(int(self.vm_hw['smp']),
                     int(session.cmd(proc_count_cmd).stdout_text.strip()))

    # 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)
```


redhat.

What else is happening now?

- Guest ABI (machine-type + CPU model) - Eduardo Habkost
- SMP Coverage and corner cases - Wainer Moschetta
- BIOS/OVMF tests – Philippe Mathieu-Daudé
- Generic (simpler) QEMUMachine configuration – Caio Carrara
- And more!



redhat.

What's next?

- Weekly meetings
  - How to join them will be posted on qemu-devel
- Migration support
- Test sets:
  - subsystem/maintainer specific
  - Combinatorial Independent Test based
- Regression tests for known fixed issues
- libvirt?
- **Whatever the community says**

redhat.

Resources

- Avocado GitHub project:
  - https://github.com/avocado-framework
- Avocado Trello Planning Board:
  - https://trello.com/b/WbqPNI2S/avocado
- Avocado QEMU Trello Planning Board:
  - https://trello.com/b/6Qi1pxVn/avocado-qemu

redhat.

THANK YOU