Developing tests for the KVM autotest framework

Lucas Meneghel Rodrigues
lmr@redhat.com

KVM Forum 2010
August 9, 2010

Goals

- Describe how KVM autotest was created, what problems it tries to solve.
- Present the features provided by the test framework and API.
- Present how the tests are structured and how to run your first test sets.
- Develop a simple test, showing some of the high level utilities that KVM autotest provides to test writers.

Test automation

Test automation consists in using software to control the execution of tests in another software, that otherwise would have to be executed manually. With automation we have:

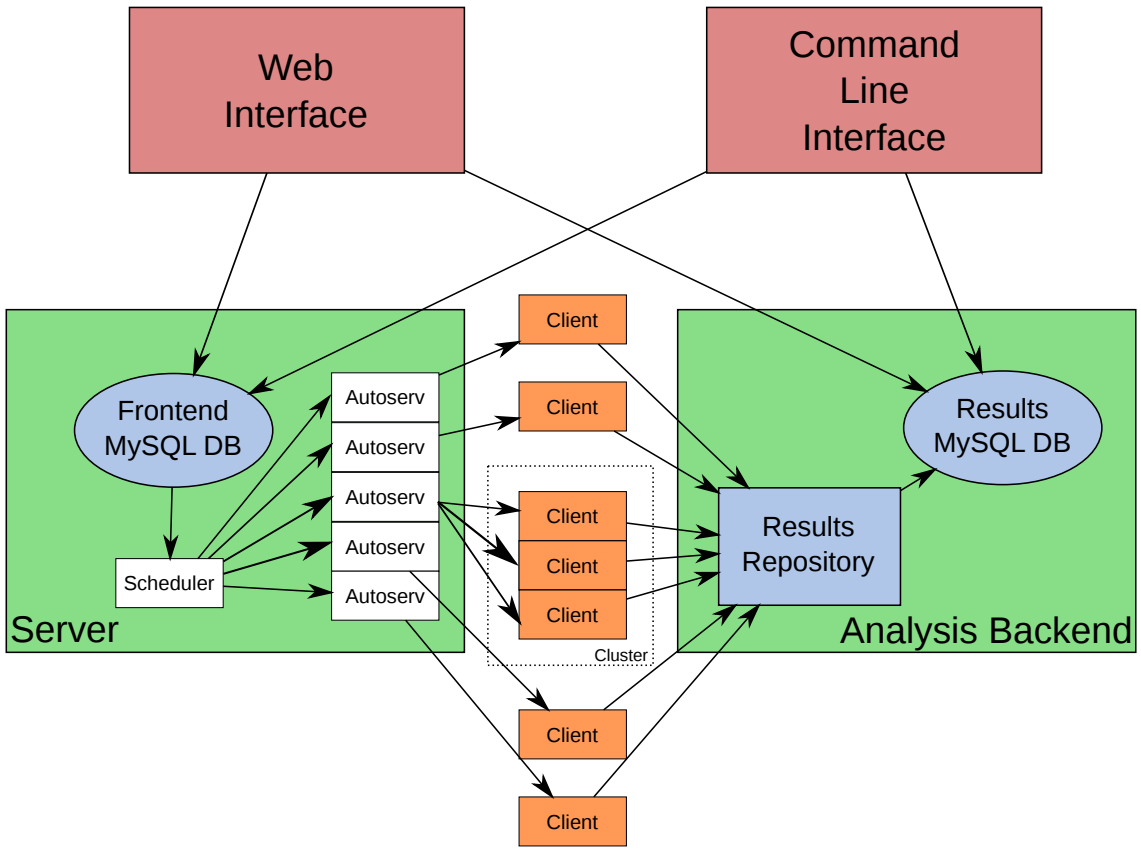
- Reduced execution time
- Reproducible, reliable tests
- Consistent test schedule

Autotest in a nutshell

Autotest (`http://autotest.kernel.org/`) is a set of libraries and programs used to automate regression and performance tests on the linux platform. Composed by:

- **Client:** Engine that executes tests in test machines
- **Server:** Copies client code to the test machines, triggers test execution, monitors machine/test status and brings back test results to the server machine
- **Scheduler:** Schedules test jobs according to user input, creating server processes for each job, and stores results on autotest's test database
- **Frontends:** Allows users to run jobs and visualize test results conveniently

Autotest in a nutshell



The wonders of virtualization testing

Virtualization presents a range of technical challenges to be resolved when it comes to effective automated testing:

- Large test matrices - Hypervisors usually take a lot of parameters
  - Image format type and disk controllers
  - Number of CPUs
  - Network cards
- Virtual machines can run a wide range of Operating systems, which need to be installed and controlled
- We need fine grained control for the userspace parts of the stack
- Test of different branches of the code base is required

KVM, meet autotest!

A bit of history

- Developers started to work on a set of automated tests for KVM, a project known as KVM autotest
- For over a year, it was maintained as an autotest 'fork'. During this period, different test architectures were tried until some agreement was reached on `kvm_runtest_2`
- Maintenance of forks is clearly not desirable, due to smaller mindshare. An upstream merge was necessary
- Merge happened and now the tests are maintained upstream, and several improvements and cleanups were made since them

KVM autotest today

- KVM autotest is the infrastructure used to develop functional and performance tests of KVM
- It is implemented as a client side test of autotest, (`kvm`)
- It is by far the most substantial and complex autotest test. A large number of libraries and infrastructure code was developed to solve the problems aforementioned
- Currently being used by:
  - KVM developers at IBM and Red Hat
  - Internal Red Hat test servers
  - KVM QA team at Red Hat, IBM QA teams

KVM autotest: APIs and features

How does KVM autotest solve some of the virtualization testing problems presented?

- **Define large test matrices**: A new config file format was developed, in order to easily define a large matrix by generating parameters based on a cartesian product of variants
- **How to reuse processes between tests**: An environment file is kept, with pickled instances of python objects, that allows processes to persist between tests
- **How to get fine grained control over userspace processes**: An expect-like library to control qemu processes, that also makes it possible for VM processes to persist between tests and even test jobs

KVM autotest: APIs and features

- Ability to build and install KVM from several methods (release tarballs, git, brew/koji rpms)
- Fully automated install of several breeds of Linux, and all supported versions of Windows (WinXP–Win7)
- Serial output collection and login, so it's easier to capture guest kernel panics and other abnormalities
- Infrastructure to capture and do some level of core dump analysis on qemu segmentation faults
- Mechanism to run the latest qemu-kvm unittests
- Ways to install virtio drivers and run WHQL Microsoft certification suite

Main files inside the KVM test folder

- `kvm.py`: KVM test main entry point. It is a simple loader of the subtests
- `kvm_config.py`: Parser of the configuration file format
- `kvm_preprocessing.py`: Functions to modify the environment
- `kvm_subprocess.py`: Expect like library
- `kvm_utils.py` and `kvm_test_utils.py`: Utility functions
- `kvm_vm.py`: The modeling of a KVM virtual machine. Implements its methods by spawning kvm subprocess instances of qemu

Anatomy of a KVM autotest subtest

A KVM autotest test implementation boils down to implementing a python function using the test API to accomplish what you need to do, which is usually something along the lines:

- Get a living VM from the test environment
- Stablish remote sessions to the VMs
- Send commands to the remote sessions on the VMs, verify their return codes, capture their outputs
- Send commands to the qemu monitor, verify their return codes, capture their outputs
- Determine whether the test has passed or failed based on this info

Developing a new test

Getting started with the framework:

- `git clone git://github.com/ehabkost/autotest.git`
- `/path/to/autotest/client/tests/kvm/get_started.py` – this script will give you some hints on getting a basic KVM autotest setup going. Follow the instructions

Developing a new test

Steps to create a new test:

- Create a python file with your test name inside the subfolder `tests`. Ex: `guest_info.py`
- Implement a function `run_test_name`. Ex: `run_guest_info`
- Add test parameters to `tests_base.cfg.sample`, on the test parameters section, creating a variant with an arbitrary name and your test name as the test type
- Modify one of the test sets under tests.cfg in order to include your test there
- Run your test, and keep developing until you're satisfied

Developing a new test

Steps to create a new test:

- Create a python file with your test name inside the subfolder `tests`. Ex: `guest_info.py`
- Implement a function `run_test_name`. Ex: `run_guest_info`
- Add test parameters to `tests_base.cfg.sample`, on the test parameters section, creating a variant with an arbitrary name and your test name as the test type
- Modify one of the test sets under tests.cfg in order to include your test there
- Run your test, and keep developing until you're satisfied
- **Hands on time, boys and girls!**

How to contribute

- `git clone git://github.com/ehabkost/autotest.git`
- `/path/to/autotest/client/tests/kvm/get_started.py`
- Hack :)
- Send patches to `autotest@test.kernel.org` (post allowed only to subscribers)

Contact

- lmr@redhat.com and mgoldish@redhat.com
- http://www.linux-kvm.org/page/KVM-Autotest
- KVM mailing list (kvm@vger.kernel.org), autotest mailing list (autotest@test.kernel.org)