Critical to any engineering process is testing. Without proper testing, a client is bound to eat a half baked pie and the baker can be sure that the customer never visits again. Any testing process must check if the goals of the product are met and it works as expected. Testing can be done both manually or can be automated. Automated testing is usually done via a tool where the actual output is matched against the desired output. Usually, such sanity tests are performed just before the software is to be released but also during development to catch just-introduced issues. While it can't replace in-depth testing by a human, it speeds up development and saves time. One such tool is openQA.

openQA and rollout testing

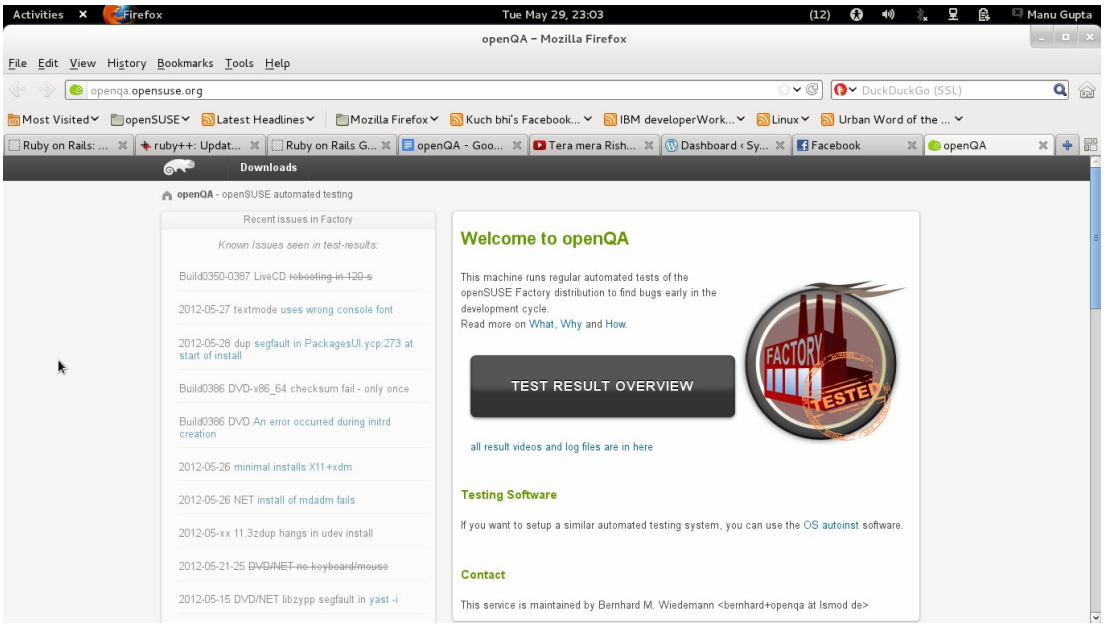
In-depth, frequent testing is difficult, time-consuming and boring. To ensure that software is tested regularly and predictably, automation is crucial. Many in-house tools which can test specific applications have been developed but there were few testing tools capable of testing a full operating system until openQA. openQA does its work entirely hands-off and offers a convenient web interface with an overview of the results. With openQA developers can catch and fix issues in the development process much faster, speeding up the development process. openQA can also help during the final stage when the software is ready to be rolled out. openQA is capable of checking time critical bugs fast without any manual work.

openQA - Introduction

openQA was developed by Bernhard Wiedemann at SUSE to provide a fully automated testing framework for testing the different operating systems he works on. The goal was to have the ability to run specific tests from checking proper functioning of individual applications like firefox to the whole boot-up procedure of an OS. One of the major aims of openQA is to provide frequent in-depth testing of operating systems and individual applications to avoid a lot of overhead and frustration in manual testing. With openQA developers can catch and fix issues in the development process much faster and thus ensure a more reliable product. Setting up openQA and analyzing test results can be as easy as pie a feat which only a few testing tools are able to do. Any operating system with a virtual machine installation (kvm or VirtualBox) along with perl will run openQA. The test results provided by openQA are provided as logs, videos and images. This in turn helps to get detailed as well as an overview of the results.

openQA testing Distributions

openQA is used by openSUSE, one of the most popular Linux distributions. Along with that, openQA also tests Debian Stable, Fedora and Arch Linux. openQA is also capable of testing BSD and MS Windows.



We will discuss how openQA is used before rolling out openSUSE as our test case. Testing is generally done on the latest development release, with additional testing sometimes done using updates from Factory to verify bug fixes. Everything in Factory is passed through our automated test framework openQA. openQA is a great test suite and is capable of producing videos of the whole process and also screenshots. This greatly reduces the overhead for the testers. An overview of the test results can be found at http://openqa.opensuse.org/ takes us to the web interface. The left column shows us the recent bugs in the latest factory build ready to be rolled out. The bugs are linked to the bugzilla and a strike over them shows them as resolved. The test overview links to the current tests running and information about the status of the tests. A detailed overview of the test results and video of the test result can be found correspondingly.

Setting up openQA

As said, setting up openQA is as easy as baking a pie. Minimum requirements to run openQA include perl and virtualization support using qemu, kvm or virtualbox. Once you have these successfully setup, clone the latest git repo from *https://git.gitorious.org/os-autoinst/os-autoinst*.git. Once you have your openQA, check the *distri/* directory to see if your distribution is supported. To keep the default configuration copy env.sh.sample to evn.sh. Now all you need is an image of the distribution and you are ready to test your distro over KVM. To run a default test, run the isotovideo tool .

os-autoinst/tools/isotovideo PATHTO/openSUSE-KDE-LiveCD-i686-Build0625-Media.iso

To configure openQA for virtual box, you need to change the backend to vbox from the default qemu in start.pl and make a virtual machine named osautoinst with a minimum of 8GB of hard disk. To customise the testing process, edit env.sh with the variables provided. env.sh is very well documented and changes are reflected from the next test onwards.

Analysing the results

Next up is analysing the test results. Test results can be divided into 3 parts: the video, the images and a more detailed log. The images can be found at os-autoinst/testresults, the videos at os-autoinst/videos and the log of the last run test at currentautoinst-log.txt. The video and the

images generated at the end of the process are usually helpful to get an indication of the bug. The text log covers a lot of details about the tests that were covered and left unchecked and also tells you if a specific application passed a test or not. Overall, the videos, images and the log provide comprehensive and detailed results which are enough to pinpoint to a bug which can be either filed or debugged.

One of the easiest way to analyse the results is to have a look at the video itself. By having a look at the video, we can easily identify the point out the points of bugs in the system. The videos make it trivial to identify a problem. However, most of the times videos are not enough. To simplify that, Bernhard devised a clever method. With a set of reference images, he coupled the installation results to check if a particular test is working as expected. To dig deeper, the best way is to look at currentautoinst-log.txt which contains the info on the last test run. This file gives you information about every single qemu / vbox key injected along with them md5 checksums. The end of the file gives a short log info on the state of the iso along with the number of tests failed and the ones that passed successfully and the ones that failed.

Newer Additions

Bernhard has recently updated his result checks to include OCR, Audio and reference images. To add a reference image, you need to add the reference image in the testimgs folder. The name of the reference image should be of the format <testname>-<screenshotnumber>-<filename>.ppm

Writing test cases

To really take full advantage of openQA you want to tweak preexisting tests or creating your own. The test modules are spread across the os-autoinst directory which can be tweaked as suited to the tester's need. Every test module has two parts, one which contains the general flow of sendkey events to test an application or feature, the second one being a set of md5 hash sums to determine the validity of test results. os-autoinst/bmqemu.pm can act as a reference for the functions that can be used in our test modules. The commands in there can be used to write the desired test module. To verify if the test results are valid or not, a set of md5 hash sums of screenshots of the desired results is checked. To calculate these hashes you can use tools/inststagedetect2.pl. These tests are written in perl. While knowing perl is not a necessity, a basic know how is recommended.

We will see how the following ssh test module works in os-autoinst.

```
1. use base "basetest";
2. use bmwqemu;
3. # check if sshd works
4. sub run()
5. {
6.         my $self=shift;
7.         script_sudo('/sbin/insserv -r SuSEfirewall2_setup'); # disable firewall to
   make better cloud images
8.         script_sudo('/sbin/insserv -r SuSEfirewall2_init');
```

```
9.          script_sudo('/sbin/insserv sshd');
10.         script_sudo('/etc/init.d/sshd restart'); # will do nothing if it is already
    running
11.         $self->take_screenshot;
12.         sendkey("ctrl-l");
13.         script_run('echo $?');
14.         script_sudo('/etc/init.d/sshd status');
15.}
16.
17.sub checklist()
18.{
19.         # return hashref:
20.         return {qw(
21.                 369dfa49bdaeb2c74be111ddae4c75b1 OK
22.                 f358adccd925bc86974213b4c3482b26 OK
23.                 77a8bc8416a0644b9bb6c31c20dbc23d OK
24.         )}
25.}
26.1;
```

The first 2 imports depict the modular nature of the module, where Line 1 takes in the base test module and Line 2 imports the utility function. The next thing we notice are the two functions, run() and checklist(). The run() function runs the test and the checklist() functions contains the screenshot-section md5sums occur with good ("OK") or bad ("fail") results. These md5sums can be taken from logfile or computed with tools/inststagedetect2.pl ssh.ppm The run() scripts runs a set of scripts predefined by perl functions. One of the interesting pieces of code is $self->take_screenshot; which takes in the screenshot and saves it as an image. If you are interested in writing your own test cases, you must read http://os-autoinst.org/testmodules.html

Limitations

While openQA is able to do most of the test cases, there is one major limitation. openQA is unable to check for hardware issues. Quoting Bernhard, “the point of openQA is that 90% of all important bugs are not specific to any hardware so by finding these, the other 10% have a higher chance to stick out elsewhere. “ However, Bernhard is not short of ideas. He plans to implement HDMI-capture and a USB-keyboard-emulator to test physical hardware or emulate hardware testing using VNC.

Similar Tools

Along with openQA, there is one more interesting tool that is coming up and is fast under development is Hydra which is used by the NixOS project. It continuously checks out sources of software projects from version management systems to build, test and release them. This may interest some users too.

The future

openQA looks promising. A few Debian, Ubuntu and Arch developers have expressed interest to use openQA as the base framework for their automated testing. Its a young project and is actively seeking developers to develop on it and packagers to port on it. So if you want to work on openQA, you can contact Bernhard or look up his TODO list at http://os-autoinst.org/todo.txt.