Cloudstack - Continuous Integration

Infrastructure, Simulator, Automation , new CI Changes for CloudStack:

Overview: With new modifications in CI, Improvements in Infrastructure Automation, Simulator Enhancements, Marvin Framework enhancements, Test Categorization, we can now automatically provision a Hypervisor Ex: XenServer, KVM etc, auto Install management server, automatically install Marvin, run parallel tests for a given build across multiple hypervisors, automatically log bugs, analyze logs collected, generate reports and email them with detailed information etc, are all possible. With this, CI will automatically kick start every few hours, picking up latest code for a configured git branch and repo, does all the things mentioned above.

So, this is more of an end to end automation capability, and get CI run frequently with latest code and minimal effort.

With new self service changes, any body can request an automation run and get the report emailed to them easily. This will help them in validating their feature mergers and other easy validations.(Phase 2)

High level Objectives of these changes:

- Automate the semi-manual testing procedure that is being followed now.
- Automate log gathering and bug filing.
- Optimize the test runs so that they can be used as self service and can be run on simulator without hardware, for continuous integration and validation of business logic.
- Fix the simulator so that it is in sync with the new features added during last few releases
- Add capability to the simulator for fault injection tests,timeouts etc.
- Self service regression tests that community can use to verify their changes.
- Provide Profiling and coverage Information.

Simulator now up to date with current features.

- Test cases are now categorized as simulator based(selfservice) and hardware dependent(provisioning).
- Simulator based tests can be used for regression check by developers in their developers setup itself with out real hardware.
- Reduced test cycle times by enabling parallel test execution in multiple zones. For example simulator and provisioning test cases can run in different zones parallel.
- Improved resource management, tests can be parallel executed and queued.

About Simulator :

- Run tests without using actual hardware, eliminates setup and reduces test execution time.
- With bug fixes and enhancements, additional tests can now run against simulator and these can be automated for better
- With the enhancements additional tests can now run against simulator resulting in improved code coverage

Simulator: Comparison (old vs new) :

Old Simulator	New Simulator Improvements
Mocks only success scenarios	Supports simulating faults-injections, timeouts
No way to change/cleanup mock	Allows defining test case specific mocks which can defined and so applicable for entire test be cleaned up as appropriate suite
Test only basic scenarios	Ability to test complex scenarios like VM. deployment retry logic, HA, VM migration etc.
No way to change/cleanup mock once. No way to check if the mocked behavior is executed. Limited code coverage	Ability to check if mock executed. This makes tests more deterministic . With the ability to tests failure/negative scenarios. Tests additional tests can be quickly developed thereby improving overall coverage. Enables devs to do better upfront testing rather then relying on QA team, should result in less bugs late in the cycle

Validating check-ins for your local changes, using Simulator

• From developers' point of view, whenever they do a change on their local environment, they can now easily validate the changes by running **“selfservice"** tests before check-in the change.

• Using "simulator", they can run these tests locally on their development environment, without requiring any hardware . This will help them to make sure that CS code is not broken with their changes, by running basic integration tests.

Simulator: Next steps

• Think of simulator as a first class HV, feature specification should clearly call it out in the list of supported HVs. Feature devs needs to keep the simulator up to date with respective changes.

• Devs/QA to write new tests for both existing and new features (target should be to improve test coverage of the overall product code)

Automation Infrastructure Enhancements:

• Ability to install, run simulator and other hyper visors in different zones simultaneously. Run tests in parallel against multiple hypervisors, EX: Xenserver and KVM regression can now run in parallel, automatic provisioning of kvm, xen and other hypervisors now available.

• Automatic collection of logs. Addition of analysis engine for log analysis post the run .

• Automatic bug filing system.

• Better utilization of test hardware: Simulator is run in a VM, real hardware is used only for provisioning tests.

• Queuing of jobs if capacity of resources is reached.

• Use of oss and easy to use tools like cobbler and puppet with extensible jenkins system eases addition of new

New CI Run Design and WorkFlow: <will be added soon>

Email Notification and Reporting: With CI runs continuously for a given version of CS, reports will be automatically emailed with below information.

1. Test Run Information: The report contains test run and other meta data information.

EX:

Build URL	http://jenkins.abcd.com/job/report_generator_47_Sandbox-simulator_simulator.xml/5/
Project:	report_generator_47_Sandbox-simulator_simulator.xml
Build Start Date:	Wed May 14 16:55:57 EST 2014
Build End Date:	Wed May 14 16:55:57 EST 2014
Git Repo Url:	https://git-wip-us.apache.org/repos/asf/cloudstack.git
Git Commit Id:	260e06d64c07c6e5f3c133d8bdc2779fad62c672
Git Branch Info:	4.4
Zone Name:	xs
Hyper Visor Info:	Simulator
Build No:	47
Build Detailed Report:	http://jenkins.abcd.com/job/${ENV,var="JOB_NAME"}/5/testReport/
Wiki Links:	Http://cwiki.apache.org
Build Cause:	Started by user
Build Description:	
Built on:	cobbler-SC

2. Test case Success,Error,Skipped count.

3. Individual test case pass\fail description. If there are bugs logged already for a test case, then it will be mentioned as a known issue.

Log Uploader Job: This job post the CI run uploads all logs to nfs share. All logs post the run are available under designated nfs share at configured location in CI, EX: **//example-nfs.ex.com/var/www/<4.4 version>** etc. The logs uploaded here, will have test case run log, product logs etc available here for analysis for failures\run information. Once logs are uploaded, people can see logs in web UI and browse them from UI available at by clicking on and selecting individual versions and hypervisors.EX: ***Http://<Ip-address>/LogAnalyzer***

BugLoggerJob: A job with bug logger service is created in jenkins, which will automatically log bugs to jira system, configured. This job currently can be run by QA post the run, by configuring the build, path of logs etc, and it will analyze logs uploaded as mentioned earlier and log bugs to jira system .

QA\Team Responsibilities in CI Run: Once QA\Team identifies failures in the mail report sent above, and logs bugs for the failures, then they can mark individual test case with the logged bugid information. **EX:** If a test case by name **"test_deploy_vm_start_failure" fails** under test suite **"test_deploy_vm .py"** , then this test case need to be marked with bugid information as mentioned below. Push these changes to repo. So, next time, when CI runs, this test case will be disabled automatically from run till it gets fixed.

@attr(tags = ['selfservice'],BugId="234")

def test_deploy_vm_start_failure(self):

Developers Responsibilities in CI Run: Developers, once they have a bug raised for failures in CI, have to fix the bug, and then remove the bugId information from the respective test case. Once disabled, the test case will again be part of CI run.

CI Run: In order to exclude failed test cases(for which there are bugs logged already) from re running again and again, then while we are running CI, we have to use below additional tags to nosetests as mentioned below.

nosetests --with-xunit --xunit-file=test_deploy_vm.xml --with-marvin --marvin-config=/home/santhosh/softwares/cs_4_4/cloudstack/setup/dev/advanced.cfg /home/santhosh/softwares/cs_4_4/cloudstack/test/integration/smoke/test_deploy_vm.py ***-a tags=advanced,'!BugId'*** *--zone="Sandbox-simulator" --collect-only*

Code Changes Location:

1. Infrastructure Code: All the Infrastructure code related to the new CI changes are maintained at the private repo. Once decided and cleaned, we will upload those changes to a particular location either inside of CS or otherwise.

2. Marvin\TestCode Changes: All the marvin\test code changes are available as part of CS git Repository and relevant branch itself.

https://git-wip-us.apache.org/repos/asf/cloudstack.git

Links\ References:

https://cwiki.apache.org/confluence/display/CLOUDSTACK/Validating+checkins+for+your+local+changes%2C+using+Simulator

https://cwiki.apache.org/confluence/display/CLOUDSTACK/Simulator+enhancements

https://cwiki.apache.org/confluence/display/CLOUDSTACK/Writing+tests+leveraging+the+simulator+enhancements