Devcloud Continuous Tests

Apache CloudStack's DevCloud appliance is used today to deploy a basic zone CloudStack environment. Within this DevCloud appliance we run basic integration tests to ensure that essential functionality isn't broken by the rapid checkin process. This page lists the details on how these workers are setup with our continuous integration system.

DevCloud (XenServer - basic)

The XCP-based DevCloud is capable of deploying a basic zone CloudStack environment. We've patched this appliance with a clone of the incubator-CloudStack ASF repo and our test utilities (marvin, cloudmonkey) and created a 'testcloud' appliance. The following workflow explains the way tests are run within the appliance:

Workflow

The testcloud appliance contains an init script at /etc/init.d/DevCloud that will start up a testrunner process when the appliance boots. Following this the testrunner will perform the necessary setup to prepare a basic zone CloudStack management server.

1. The test runner fetches the latest HEAD (currently set to master) from the git repository and builds CloudStack. In other words - mvn install
2. This is followed by an installation of marvin so we have the latest API's available via the test framework
3. Further on the client inside a jetty:run will start up a management server process and have the API server listening for requests
4. We then configure this management server with the basic zone DevCloud configuration with one XCP host and local storage. i.e., the config at tools/DevCloud/DevCloud.cfg
5. Once we have the basic zone running we will do a basic health check that verifies the following
    a. All system VM's have come up and are successfully 'Running'
    b. All the featured templates (built-in) are in Ready (downloaded) state
6. If the health check succeeds the environment is ready to run tests
7. nose's testrunner with the marvin plugin will collect all the tests marked with the attribute of 'DevCloud' @attr(tags='DevCloud'), put them in a suite and run them
    a. The nose xUnit plugin gives us the XML output for consumption into jenkins
8. Management Server log, logs from the test appliance's init script and the test results are posted to the machine hosting the 'testcloud' appliance and held as artifacts on the jenkins job for troubleshooting and held as artifacts on the jenkins job for troubleshooting
9. A jenkins slave agent within the host will send through the results and logs to the jenkins master at jenkins.c.o

Test Workers

The host machine is a Ubuntu server with VirtualBox 4.2 installed with guest additions. VirtualBox performs DHCP for the 'testcloud' workers that it spins up. The testcloud image itself is cloned into multiple test workers (currently 5). A simple scheduler will pick up an idle worker VM on trigger from jenkins master that is polling for commits on the git:repo. Once picked up each worker will perform all the tests and post the results back to the gateway/DHCP which in our case is the host machine running VirtualBox. A jenkins slave runs in headless mode on the host machine and posts results back to jenkins master.

After the testworker has done its job and posted its results back to jenkins we wait for a timeout (1800s) to release it back to the pool of workers. To clean up the image of any remnants from the previous test run the worker is restored to a base snapshot keeping the environment clean for the subsequent run.

Code
When we change mvn instructions for building CloudStack and/or the tests that need to run the testrunner process will fetch the latest version of itself from a github repo running DevCloud-ci. This is useful so we don't change the image of the testcloud appliance when our build process changes.

All the code is hosted on github. Feel free to fork it, play around with it and contribute to making it an effective tool for automating our tests. Comments and Suggestions welcome on the CloudStack-dev@i.a.o lists

Jenkins Job

The jenkins job is a vanilla job enabled with JUnit reports. E-mail will be enabled to CloudStack-commits@i.a.o

Setup

The setup is a single machine with 4 core CPU, 8GB RAM and 500GB of hard disk space. Installed with Ubuntu 12.04 LTS - Server Edition.

Following additional packages are installed: VirtualBox 4.2.4, VirtualBox 4.2.4 - Extension Pack, VirtualBox Guest additions

1. create a jenkins user on this machine that will run a jenkins-slave service which will interact with jenkins master to trigger the test runs and send test reports. you'll need access to add nodes and create jobs on the jenkins CI. once the node is setup and the slave is connected you are ready to schedule jobs.

2. modify the default network in VirtualBox to perform DHCP as follows and enable the DHCP server

```
vboxmanage dhcpserver modify --netname HostInterfaceNetworking-vboxnet0 --ip 192.168.56.1 --netmask
255.255.255.0 --lowerip 192.168.56.101 --upperip 192.168.56.254
```

3. wget the testcloud appliance to your server. The appliance is pre-seeded with the init script that will launch the test runner and configure CloudStack. import the appliance into VirtualBox to change the default credentials in the script to suit those of your jenkins user.

```
vboxmanage import testcloud.ova --vsys 0 --eula accept --vsys 0 --vmname testcloud
```

4. start the VM and login to DevCloud

```
$ vboxmanage startvm testcloud --type headless
```

and change the password in the /etc/init.d/DevCloud init script

```
28       python $SCRIPTDIR/setUpTestWorker.py --host 192.168.56.1 --user jenkins --pass yourpassword --out /var
/lib/jenkins/reports
```

and then reinitialize the system-V service

```
$ update-rc.d DevCloud defaults
```

5. stop the VM and export it as an appliance out of which we will create further test workers.

```
$ vboxmanage controlvm testcloud poweroff
$ vboxmanage export testcloud --output testcloudworker.ova
```

6. import the appliance in to virtual box again but creating as many workers as you need.

```
$for i in {1..5}; do vboxmanage import testcloudworker.ova --vsys 0 --eula accept --vsys 0 --vmname
testworker$i; done
```

7. create a base snapshot that will be restored to after the testworker VM has finished running a test. This is to restore the appliance to its clean state for the next test run

```
$for i in {1..5}; do vboxmanage snapshot testworker$i take testworkerbase$i; done
```

8. your workers are now ready to be used and connected to the jenkins job. The jenkins scheduler script is available on the github repo as ci/scripts /schedulerun.sh

```
jenkins@apache-81-1:~$ vboxmanage list vms
"testworker1" {e6b5924b-efac-40cf-9621-3bb0b4683ab9}
"testworker2" {070da3d1-4e09-4171-9ec0-17c10d8ca425}
"testworker3" {dff3a63c-32ac-4631-aae1-66837a049fb0}
"testworker4" {931baa82-1b37-460e-a500-7764a3ee38dd}
"testworker5" {0bdd381d-bfc0-4b8e-8bbc-908fbe7b4710}
```

TODO's and FIXME's

1. Speed up the mvn clean install step with a local nexus proxy/maven cache. Currently this eats up around 5m

2. The logs should continuously write to a remote syslog appender on the virtual box gateway. We shouldn't depend on test worker process to complete each time.

3. Draft a proposal for moving this to builds.a.o

DevCloud (KVM - advanced)

WIP: We intend to do the same for a KVM based DevCloud environment to so as to test any functionality that is essential for advanced zone CloudStack deployments.