RALLY – PERFORMANCE BENCHMARKING OF OPENSTACK

Tips and tools for creating and presenting wide format slides
Openstack – cloud platform
Current Openstack - Ecosystem
Everything is written in python

End users can interact through a common web interface (Horizon) or directly to each service through their API

All services authenticate through a common source

Individual services interact with each other through their public APIs

Most daemons implemented WSGI middleware (Paste)
- Used extensively in OpenStack
- Configured through *-paste.ini files
Identity ("Keystone")

- Keystone provides a single point of integration for OpenStack policy, catalog, token and authentication.
- **keystone** handles API requests as well as providing configurable catalog, policy, token and identity services.
- Standard backends include LDAP or SQL, as well as Key Value Stores (KVS).
- Most people will use this as a point of customization for their current authentication services.
Dashboard ("Horizon")

- Django application that users can access in their web browser
- Communicates with each OpenStack service through their API (and sometimes their admin API)
Object Storage ("Swift")

- Stores and serves objects (files)
- Employs object level replication to safeguard data
- Accepts client requests via Objectstore API or HTTP from clients through `swift-proxy`
- Maintains distributed account and container databases
- Stores objects according the ring layout on filesystem with extended attributes (XFS, EXT4, etc.)
Image Service ("Glance")

- **glance-api** accepts Image API calls for image discovery, image retrieval and image storage.
- **glance-registry** stores, processes and retrieves metadata about images (size, type, etc.).
- Database to store the image metadata.
- A storage repository for the actual image files. In many deployments, this is OpenStack Swift.
Compute ("Nova")

- **nova-api** accepts and responds to end user compute API calls.

- Supports OpenStack Compute API, Amazon's EC2 API and a special Admin API (for privileged users to perform administrative actions).

- Initiates most of the orchestration activities (such as running an instance)

- Enforces some policy (mostly quota checks)

- Authentication is handled through middleware before getting to this daemon
Block Storage ("Cinder")

- **cinder-api** accepts API requests and routes them to cinder-volume for action.

- **cinder-volume** acts upon the requests by reading or writing to the Cinder database to maintain state, interacting with other processes (like cinder-scheduler) through a message queue and directly upon block storage providing hardware or software. It can interact with a variety of storage providers through a driver architecture. Currently, there are drivers for IBM, SolidFire, NetApp, Nexenta, Zadara, linux iSCSI and other storage providers.

- Much like nova-scheduler, the **cinder-scheduler** daemon picks the optimal block storage provider node.
Networking (“Quantum”)

- **quantum-server** accepts API requests and then routes them to the appropriate quantum plugin for action.
- Quantum ships with plugins and agents for:
  - Cisco virtual and physical switches
  - Nicira NVP product
  - NEC OpenFlow products
  - Open vSwitch
  - Linux bridging
  - Ryu Network Operating System
  - Midokua
- The common agents are L3 (layer 3), DHCP (dynamic host IP addressing) and the specific plug-in agent.
"How does OpenStack work at scale?"
Rally Components

- **Server Providers** - provide servers (virtual servers), with ssh access, in one L3 network.
- **Deploy Engines** - deploy OpenStack cloud on servers that are presented by Server Providers
- **Verification** - component that runs tempest (or another specific set of tests) against a deployed cloud, collects results & presents them in human readable form.
- **Benchmark engine** - allows to write parameterized benchmark scenarios & run them against the cloud.
Rally - Functionality

1. Deploy (or use existing) OpenStack cloud
2. Verify (run tempest)
3. Benchmark (generate real user load)
4. Generate report based on results of verification, benchmarks & profiling info

Major Rally actions:

- Cloud endpoints
- Verification results
- Profiling data from Ceilometer
- Benchmark results
- Get verification & benchmark results

Rally
OpenStack
Rally Database
Rally Use Cases

**Rally for Devs & QA:**

Rally → Deploy OpenStack → Simulate real user load → Process & aggregate results → Make OpenStack better

Not clear where is issue? Just run another benchmark or change load level

Deploy new OpenStack with:
1) another configuration
2) code that fix performance issue
3) different third party components (mysql or psycopg, rabbit or qpid)

**Rally for DevOps:**

Rally → Use existing cloud → Simulate real user load → Process & aggregate results → Ensure that OpenStack pass SLA

With admin access (create temp users)

without admin access:
use set of existing users

**Rally CI/CD:**

Rally → Deploy OpenStack on specific hardware and configuration with latest version of your tool and code → Run specific set of benchmarks → Store historical performance data

Improve OpenStack continuously

Track OpenStack Quality
Installing Rally on Ubuntu.

**Note**: We would recommend using a separate machine for Rally. If the machine that you use to run rally has some Openstack components running, we suggest creating a virtual environment for running Rally because it may have conflicts with version of the client python libraries.

**Prerequisite**

- `sudo apt-get update`
- `sudo apt-get install libpq-dev git-core python-dev libevent-dev libssl-dev python-pip libffi-dev`
- `sudo pip install pbr`
Installing Rally on Ubuntu.

**Installing Rally**

**Clone**
- git clone https://github.com/stackforge/rally.git & cd rally
- python setup.py install

**Configure**
- sudo mkdir /etc/rally
- sudo cp etc/rally/rally.conf.sample /etc/rally/rally.conf
- sudo vim /etc/rally/rally.conf
- # Change the "connection" parameter.
- # e.g. to connection="sqlite:///home/<your_username>/venv/rally-db/$sqlite_db" (or any other place)

**Create Database**
- rally-manage db recreate
Rally : Deploy Engines.

The task of a deploy engine is to control the process of deploying some OpenStack distribution like DevStack or FUEL before any benchmarking procedures take place.

Every deploy engine should implement the following fairly simple interface:

- constructor, which takes a deployment entity as its only parameter;
- deploy(), which should deploy the appropriate OpenStack distribution given the cloud config from the deployment object the engine was initialized with (possibly using one of available server providers);
- cleanup(), which should clean up the OpenStack deployment (again, possibly using one of available server providers).
Rally. Deploy Engine . Example

If you already have a existing Openstack Deployment that you want to benchmark:

Use a json file that looks like one below. (with value specific to your system):

```json
{
    "name": "DummyEngine",
    "endpoint": {
        "auth_url": "http://192.168.122.22:5000/v2.0/",
        "username": "admin",
        "password": "password",
        "tenant_name": "admin"
    }
}
```

For Devstack based deployment:

```json
{
    "name": "DevstackEngine",
    "localrc": {
        "ADMIN_PASSWORD": "secret",
        "NOVA_REPO": "git://example.com/nova/",
        ...
    }
}
```
Server providers in Rally are typically used by deploy engines to manage virtual machines necessary for OpenStack deployment and its following benchmarking.

The key feature of server providers is that they provide a unified interface for interacting with different virtualization technologies (LXS, Virsh etc.) and cloud suppliers (like Amazon).

Every server provider should implement the following basic interface:

- **constructor**, which should take the deployment entity the provider should bind to and a config dictionary as its parameters;
- **create_servers(image_uuid, type_id, amount)**, which should create the requested number of virtual machines of the given type using a specific image. The method should also return the list of created servers wrapped in special Server entities.
- **destroy_servers()**, which should destroy all virtual machines previously created by the same server provider.
Rally – Server Providers – Ex

DummyProvider – If you already have an Openstack Deployment.

This provider does nothing, but returns endpoints from configuration. This may be useful if you have specific software/hardware configuration ready to deploy OpenStack.

```json
{
    "name": "ExampleEngine",
    "provider": {
        "name": "DummyProvider",
        "credentials": [
            {"user": "root", "host": "host1.net"},
            {"user": "root", "host": "host2.net"}]
    }
}
```
Rally – How_to_run

Simple 😊

1. Initialize your Deployment
2. Create a json for Supported Benchmarking scenario
3. Run Benchmarking against a deployment above
Rally – Initialize Deployment

1. Create a Deployment configuration (json) file.
   If you are running Rally against a existing Openstack Deployment your should look like

   ```json
   {
     "name": "DummyEngine",
     "endpoint": {
       "auth_url": "<KEYSTONE_AUTH_URL>",
       "username": "<ADMIN_USER_NAME>",
       "password": "<ADMIN_PASSWORD>",
       "tenant_name": "<ADMIN_TENANT>
     }
   }
   ```

2. Create a deployment using `deployment create` command

   ```
   $ rally deployment create --filename=dummy_deployment.json --name=dummy
   ```

3. If you want to list deployments

   ```
   $ rally deployment list
   ```

4. Switch to a different Deployment

   ```
   $ rally use deployment --deploy-id=<Another deployment UUID>
   ```
Rally – Set Benchmark scenario.

Some sample configurations are provided at "rally/doc/samples/tasks/".

Let's pick up a scenario boot-and-delete.json from Nova. It looks like

```json
{
    "NovaServers.boot_and_delete_server": [
        {
            "args": {
                "flavor_id": 1,
                "image_id": "73257560-c59b-4275-a1ec-ab140e5b9979"},
            "execution": "continuous",
            "config": {
                "times": 10,
                "active_users": 2,
                "tenants": 3,
                "users_per_tenant": 2
            }
        }
    ]
}
```

Modify this to design your test-case. Similarly for all other cases other available json can be modified or you can even write a new one for a custom case.

Rally by default uses the last created deployment. Use the switch deployment commands to run the tests with a different deployment.
Lets dig deeper

- **Test name**: "NovaServers.boot_and_delete_server"
- **Execution**: either continuous / periodic
- **Times**: Number of times the test needs to be run
- **Active_users**: Number of parallel threads (concurrent users)
- **Tenants**: Total number of tenants to be created
- **Users_per_tenant**: Number of users per single tenant

Other parameters to be used only with supported tests

- **Network**: Name of network to be used
- **Script**: If a script is passed as input to the test
- **Actions**: *soft_reboot* / *stop_start*
Rally – Run Benchmark.

Run your benchmark scenario by pointing at the json you created in the previous step
$ rally --verbose task start --task=my-task.json

You can check the state of the task
$ rally task list

To check a complete task analysis
$ rally task detailed <Task UUID>
Rally Result – Example.

`$ rally task detailed <Task UUID>`

<table>
<thead>
<tr>
<th>action</th>
<th>max (sec)</th>
<th>avg (sec)</th>
<th>min (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>nova.boot_server</td>
<td>9.22798299789</td>
<td>8.90022659302</td>
<td>8.57247018814</td>
</tr>
<tr>
<td>nova.delete_server</td>
<td>4.24928498268</td>
<td>3.26377093792</td>
<td>2.27825689316</td>
</tr>
</tbody>
</table>

---

Task <Task UUID> is finished. Failed: False

---

test scenario NovaServers.boot_and_delete_server
args position 0
args values:
{u'args': {u'flavor_id': <Flavor UUID>,
            u'image_id': u'<Image UUID>'},
   u'config': {u'active_users': 1, u'times': 2}}
Detailed performance benchmarking
Docker in OpenStack

• Havana
  – Nova virt driver which integrates with docker REST API on backend
  – Glance translator to integrate docker images with Glance
• Icehouse
  – Heat plugin for docker
• Both options are still under development

**nova-docker virt driver**

**docker heat plugin**
About This Benchmark

• Use case perspective
  – As an OpenStack Cloud user I want a Ubuntu based VM with MySQL... Why would I choose docker LXC vs a traditional hypervisor?

• OpenStack “Cloudy” perspective
  – LXC vs. traditional VM from a Cloudy (OpenStack) perspective
  – VM operational times (boot, start, stop, snapshot)
  – Compute node resource usage (per VM penalty); density factor

• Guest runtime perspective
  – CPU, memory, file I/O, MySQL OLTP

• Why KVM?
  – Exceptional performance
Benchmark Environment Topology @ SoftLayer

controller

compute node

controller

compute node

9/12/2014
## Benchmarks Specs

<table>
<thead>
<tr>
<th>Spec</th>
<th>Controller Node (4CPU x 8G RAM)</th>
<th>Compute Node (16CPU x 96G RAM)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Environment</strong></td>
<td>Bare Metal @ SoftLayer</td>
<td>Bare Metal @ SoftLayer</td>
</tr>
<tr>
<td><strong>Mother Board</strong></td>
<td>SuperMicro X8SIE-F Intel Xeon QuadCore SingleProc SATA [1Proc]</td>
<td>SuperMicro X8DTU-F_R2 Intel Xeon HexCore DualProc [2Proc]</td>
</tr>
<tr>
<td><strong>CPU</strong></td>
<td>Intel Xeon-Lynnfield 3470-Quadcore [2.93GHz]</td>
<td>(Intel Xeon-Westmere 5620-Quadcore [2.4GHz]) x 2</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td>(Kingston 4GB DDR3 2Rx8 4GB DDR3 2Rx8 [4GB]) x2</td>
<td>(Kingston 16GB DDR3 2Rx4 16GB DDR3 2Rx4 [16GB]) x 6</td>
</tr>
<tr>
<td><strong>HDD (LOCAL)</strong></td>
<td>Digital WD Caviar RE3 WD5002ABYS [500GB]; SATAII</td>
<td>Western Digital WD Caviar RE4 WD5003ABYX [500GB]; SATAII</td>
</tr>
<tr>
<td><strong>NIC</strong></td>
<td>eth0/eth1 @ 100 Mbps</td>
<td>eth0/eth1 @ 100 Mbps</td>
</tr>
<tr>
<td><strong>Operating System</strong></td>
<td>Ubuntu 12.04 LTS 64bit</td>
<td>Ubuntu 12.04 LTS 64bit</td>
</tr>
<tr>
<td><strong>Kernel</strong></td>
<td>3.5.0-48-generic</td>
<td>3.8.0-38-generic</td>
</tr>
<tr>
<td><strong>IO Scheduler</strong></td>
<td>deadline</td>
<td>deadline</td>
</tr>
<tr>
<td><strong>Hypervisor tested</strong></td>
<td>NA</td>
<td>- KVM 1.0 + virtio + KSM (memory deduplication)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- docker 0.10.0 + go1.2.1.1 + commit dc9c2bf + AUFS</td>
</tr>
<tr>
<td><strong>OpenStack</strong></td>
<td>Trunk master via devstack</td>
<td>Trunk master via devstack. Libvirt KVM nova driver / nova-docker virt driver</td>
</tr>
<tr>
<td><strong>OpenStack Benchmark Client</strong></td>
<td><a href="https://github.com/openstack/project_rally">OpenStack project rally</a></td>
<td>NA</td>
</tr>
<tr>
<td><strong>Metrics Collection</strong></td>
<td>NA</td>
<td>dstat</td>
</tr>
<tr>
<td><strong>Guest Benchmark Driver</strong></td>
<td>NA</td>
<td>- [Sysbench](<a href="https://github.com/">https://github.com/</a> CLICKL/systest) 0.4.12</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- mbw 1.1.1-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- [iibench](<a href="https://github.com/">https://github.com/</a> CLICKL/systest) (py)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- netperf 2.5.0-1</td>
</tr>
<tr>
<td><strong>VM Image</strong></td>
<td>NA</td>
<td>- Scenario 1 (KVM): <a href="https://releases.ubuntu.com/12.04/">official ubuntu 12.04 image</a> + mysql snapshotted and exported to qcow2 – 1080 MB</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Scenario 2 (docker): [guillermo/mysql](<a href="https://github.com/">https://github.com/</a> guillermo/docker) -- 1080 MB</td>
</tr>
</tbody>
</table>
## Test Descriptions: Cloudy Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Benchmark Driver</th>
<th>Description</th>
</tr>
</thead>
</table>
| Serial VM boot (15 VMs)    | OpenStack Rally  | - Boot VM from image  
- Wait for ACTIVE state  
- Repeat the above a total of 15 times  
- Delete VM |
| VM reboot (5 VMs rebooted 5 times each) | OpenStack Rally | - Boot VM from image  
- Wait for ACTIVE state  
- Soft reboot VM 5 times  
- Delete VM  
- Repeat the above a total of 5 times |
| VM snapshot (1 VM, 1 snapshot) | OpenStack Rally | - Boot VM from image  
- Wait for ACTIVE state  
- Snapshot VM to glance image  
- Delete VM |
## Test Descriptions: Guest Benchmarks

<table>
<thead>
<tr>
<th>Benchmark</th>
<th>Benchmark Driver</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Guest Runtime Benchmarks</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CPU performance</td>
<td>Sysbench from within the guest</td>
<td>- Run sysbench cpu test</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Repeat a total of 10 times</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Average results over the 10 times</td>
</tr>
<tr>
<td>OLTP (MySQL) performance</td>
<td>Sysbench from within the guest</td>
<td>- Run sysbench OLTP test</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Repeat a total of 10 times</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Average results over the 10 times</td>
</tr>
<tr>
<td>MySQL</td>
<td>Indexed insertion benchmark</td>
<td>- Run iibench for a total of 1M inserts printing stats at 100K intervals</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Collect data over 5 runs &amp; average</td>
</tr>
<tr>
<td>File I/O performance</td>
<td>Sysbench from within the guest</td>
<td>- Run sysbench OLTP test</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Repeat a total of 10 times</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Average results over the 10 times</td>
</tr>
<tr>
<td>Memory performance</td>
<td>Mbw from within the guest</td>
<td>- Run mbw with array size of 1000 MiB and each test 10 times</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Collect average over 10 runs per test</td>
</tr>
<tr>
<td>Network performance</td>
<td>Netperf</td>
<td>- Run netperf server on controller</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- From guest run netperf client in IPv4 mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Repeat text 5x</td>
</tr>
<tr>
<td></td>
<td></td>
<td>- Average results</td>
</tr>
</tbody>
</table>
SERIALLY BOOT 15 VMS
Cloudy Performance: Serial VM Boot

• Benchmark scenario overview
  – Boot VM via OpenStack nova
  – Wait for VM to become active
  – Repeat the above steps for a total of 15 VMs
  – Delete all VMs

Benchmark Visualization
Cloudy Performance: Serial VM Boot

Docker 1.5x faster

Average Server Boot Time

<table>
<thead>
<tr>
<th></th>
<th>docker</th>
<th>KVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3.900927941</td>
<td>5.884197426</td>
</tr>
</tbody>
</table>

Docker is 1.5x faster than KVM.
Cloudy Performance: Serial VM Boot

Docker: Compute Node CPU

KVM: Compute Node CPU

9/12/2014
Cloudy Performance: Serial VM Boot

Docker: Compute Node Used Memory

KVM: Compute Node Used Memory
### Cloudy Performance: Serial VM Boot

#### Docker: Compute Node 1m Load Average

- **Average:** 0.09%

#### KVM: Compute Node 1m Load Average

- **Average:** 9.94%
SERIAL VM SOFT REBOOT
Cloudy Performance: Serial VM Reboot

- Benchmark scenario overview
  - Boot a VM
    - Wait for it to become active
  - Soft reboot the VM
    - Wait for it to become active
    - Repeat soft reboot a total of 5 times
  - Delete VM
  - Repeat the above for a total of 5 VMs

Benchmark Visualization

<table>
<thead>
<tr>
<th>Time</th>
<th>Active VMs</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>2</td>
</tr>
<tr>
<td>23</td>
<td>2</td>
</tr>
<tr>
<td>25</td>
<td>2</td>
</tr>
<tr>
<td>27</td>
<td>2</td>
</tr>
<tr>
<td>29</td>
<td>2</td>
</tr>
<tr>
<td>31</td>
<td>2</td>
</tr>
<tr>
<td>33</td>
<td>2</td>
</tr>
<tr>
<td>35</td>
<td>2</td>
</tr>
<tr>
<td>37</td>
<td>2</td>
</tr>
<tr>
<td>39</td>
<td>2</td>
</tr>
<tr>
<td>41</td>
<td>2</td>
</tr>
<tr>
<td>43</td>
<td>2</td>
</tr>
<tr>
<td>45</td>
<td>2</td>
</tr>
<tr>
<td>47</td>
<td>2</td>
</tr>
<tr>
<td>49</td>
<td>2</td>
</tr>
<tr>
<td>51</td>
<td>2</td>
</tr>
<tr>
<td>53</td>
<td>2</td>
</tr>
<tr>
<td>55</td>
<td>2</td>
</tr>
</tbody>
</table>
Cloudy Performance: Serial VM Reboot

Average Server Reboot Time

Docker 18.9x faster

124.4525079

6.591313448

9/12/2014
Cloudy Performance: Serial VM Reboot

Average Server Delete Time

<table>
<thead>
<tr>
<th>Time In Seconds</th>
<th>docker</th>
<th>KVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>7.857602167</td>
<td>4.451871514</td>
<td></td>
</tr>
</tbody>
</table>

KVM 1.75x faster
Cloudy Performance: Serial VM Reboot

• After investigating docker delete times
  – Docker sends SIGTERM to container process to stop
  – Bash is immune to SIGTERM
  – Docker waits for X seconds before stop
  – Default container image (init) command was using bash
  – See: https://github.com/dotcloud/docker/issues/3766

• Rebuild the docker mysql image
  – Don’t use bash for container image command
  – No change to docker (hypervisor); image config only change
Cloudy Performance: Serial VM Reboot

Average Server Reboot Time (Round 2)

Docker 48.99x faster

Time In Seconds

Docker: 2.541945305
KVM: 124.4525079

9/12/2014
Cloudy Performance: Serial VM Reboot

Average Server Delete Time (Round 2)

Docker 1.09x faster

<table>
<thead>
<tr>
<th>Time In Seconds</th>
<th>docker</th>
<th>KVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4.093254519</td>
<td>4.451871514</td>
</tr>
</tbody>
</table>

9/12/2014
Cloudy Performance: Serial VM Reboot

Docker: Compute Node CPU

KVM: Compute Node CPU

9/12/2014
Cloudy Performance: Serial VM Reboot

Docker: Compute Node Used Memory

KVM: Compute Node Used Memory

Delta
57 MB

Delta
467 MB

9/12/2014
Cloudy Performance: Serial VM Reboot

Docker: Compute Node 1m Load Average

KVM: Compute Node 1m Load Average
OpenStack Cloudy Benchmark

SNAPSHOT VM TO IMAGE
Cloudy Performance: Snapshot VM To Image

• Benchmark scenario overview
  – Boot a VM
  – Wait for it to become active
  – Snapshot the VM
  – Wait for image to become active
  – Delete VM
Cloudy Performance: Snapshot VM To Image

Average Snapshot Server Time

Docker 1.62x faster

Time (Seconds)

docker: 26.39477992
KVM: 42.92771101
Cloudy Performance: Snapshot VM To Image

Docker: Compute Node CPU

CPU Usage In Percent

Time

KVM: Compute Node CPU

CPU Usage In Percent

Time

9/12/2014
Cloudy Performance: Snapshot VM To Image

**Docker: Compute Node Used Memory**

- Delta: 48 MB

**KVM: Compute Node Used Memory**

- Delta: 114 MB

9/12/2014
Cloudy Performance: Snapshot VM To Image

KVM: Compute Node 1m Load Average

Average
0.37 %

Docker: Compute Node 1m Load Average

Average
0.1 %
Guest VM Benchmark

GUEST PERFORMANCE BENCHMARKS
Guest Performance: CPU

• Linux sysbench 0.4.12 cpu test
• Calculate prime numbers up to 20000
• 2 threads
• Instance size
  – 4G RAM
  – 2 CPU cores
  – 20G disk
Guest Performance: CPU

Calculate Primes

<table>
<thead>
<tr>
<th></th>
<th>Docker</th>
<th>KVM</th>
<th>Bare Metal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Seconds</td>
<td>15.11</td>
<td>15.08</td>
<td>15.03</td>
</tr>
</tbody>
</table>

9/12/2014
Guest Performance: Memory

• Linux mbw 1.1.1-2
• Instance size
  – 2 CPU
  – 4G memory
• Execution options
  – 10 runs; average
  – 1000 MiB
Guest Performance: Memory

Memory Benchmark Performance

Memory Tests

MEMCPY
DUMB
MCBLOCK

MiB/s

BareMetal
docker
KVM
Guest Performance: Network

• Netperf 2.5.0-1
  – Netserver running on controller
  – Netperf on guest
  – Run netperf 5 times & average results

• Instance size
  – 2 CPU
  – 4G memory

• Execution options
Guest Performance: Network

Network Throughput

Throughput in 10^6 bits/second

- docker
- KVM

9/12/2014
Guest Performance: File I/O

- Linux sysbench 0.4.12 fileio test
  - Synchronous IO
  - Random read / write
  - Total file size of 150G
  - 16K block size

- Thread variations: 1, 8, 16, 32

- Instance size
  - 4G RAM
  - 2 CPU cores
  - 200G disk

- KVM specs
  - Disk cache set to none
  - Virtio
  - Deadline scheduler (host & guest)

- Docker specs
  - AUFS storage driver
  - Deadline scheduler
Guest Performance: File I/O

File I/O: Read

![Graph showing read performance for Docker and KVM across different thread counts.]

File I/O: Write

![Graph showing write performance for Docker and KVM across different thread counts.]

9/12/2014
Guest Performance: File I/O – Read / Write

File I/O: Transfer Rate

Kb/sec

Threads

Docker
KVM

9/12/2014
Guest Performance: MySQL OLTP

• Linux sysbench 0.4.12 oltp test
  – Table size of 2,000,000
  – MySQL 5.5 (installed on Ubuntu 12.04 LTS with apt-get)
• Variations
  – Number of threads
  – Read only & read / write
• Instance size
  – 4G RAM
  – 2 CPU cores
Guest Performance: MySQL OLTP (Read)

MySQL OLTP Read Transactions

- Docker
- KVM

MySQL OLTP Read Requests

- Docker
- KVM

9/12/2014
Guest Performance: MySQL OLTP (Read / Write)

MySQL OLTP Read/Write Transactions

MySQL OLTP Read/Write Requests
Guest Performance: MySQL Indexed Insertion

• Indexed insertion benchmark (iiibench python script)
  – A total of 1,000,000 insertions
  – Print stats at 100K intervals
  – Collect stats over 5 runs
  – Average

• Instance size
  – 4G RAM
  – 2 CPU cores
Guest Performance: MySQL Indexed Insertion

MySQL Indexed Insertion @ 100K Intervals

![Graph showing the performance of MySQL Indexed Insertion at 100K intervals. The x-axis represents the table size in 100K increments, while the y-axis shows the seconds per 100K insertion batch. The graph compares the performance of docker (blue line) and kvm (red line).]
BENCHMARK OBSERVATIONS
Docker LXC CPU Growth 26x Lower Than VM

User CPU Growth Trend

CPU Usage In Percent

Slope

- 0.0091
- 0.237

y = 0.237x + 2.2993
y = 0.0091x + 0.7349

9/12/2014
Docker LXC Memory Growth 3x Lower Than VM

Memory Usage Growth Trend

9/12/2014
Observations

• Cloudy operations with Docker LXC outperform VM
  – 48x server reboot, 1.5x server boot, 1.62x server snapshot, etc.
• Docker LXC density potential compared to VMs
  – 3x memory savings
  – 26x CPU savings
  – 3.22x smaller images in this test (note – image sizes can vary based on required packages)
• Docker LXC containers run on bare metal
  – Greater / equivalent “in the VM” performance at the micro benchmark level
• Micro benchmark results do not always reflect macro performance
  – Always benchmark your “real” workload
• Docker image (init) command can impact performance
  – Bash ignores SIGTERM
• Nova-docker virt driver and docker-registry components still under dev
  – Nice work, but room for improvement (python anyone?)
• Real performance of Docker LXC ops capped my Cloud manager
  – Can start the SQL image from docker CLI in 0.191s
REFERENCES
References & Related Links

- http://www.slideshare.net/BodenRussell/realizing-linux-containerslxc
- https://www.docker.io/
- http://sysbench.sourceforge.net/
- http://dag.wiee.rs/home-made/dstat/
- http://www.openstack.org/
- https://wiki.openstack.org/wiki/Rally
- http://devstack.org/
- http://www.linux-kvm.org/page/Main_Page
- https://github.com/stackforge/nova-docker
- https://github.com/dotcloud/docker-registry
- http://www.netperf.org/netperf/
REFERENCE
Rally Boot 15 VM Configuration

- Rally config:

```json
"NovaServers.boot_server": [
  {
    "args": {
      "flavor_id": 2,
      "image_id": "IMAGE_ID"
    },
    "runner": {
      "type": "constant",
      "times": 15,
      "active_users": 1
    },
    "context": {
      "users": {
        "tenants": 1,
        "users_per_tenant": 1
      }
    }
  }
]
```

- Flavor

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Memory_MB</th>
<th>Disk</th>
<th>Ephemeral</th>
<th>Swap</th>
<th>VCPUs</th>
<th>RXTX_Factor</th>
<th>Is_Public</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>m1.tiny</td>
<td>512</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1.0</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>m1.small</td>
<td>2048</td>
<td>20</td>
<td>0</td>
<td>1</td>
<td>1.0</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>m1.medium</td>
<td>4096</td>
<td>40</td>
<td>0</td>
<td>2</td>
<td>1.0</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>m1.large</td>
<td>8192</td>
<td>80</td>
<td>0</td>
<td>4</td>
<td>1.0</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>m1.nano</td>
<td>64</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.0</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>451</td>
<td>m1.heat</td>
<td>1824</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>1.0</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>m1.xlarge</td>
<td>16384</td>
<td>160</td>
<td>0</td>
<td>8</td>
<td>1.0</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>m1.perf</td>
<td>4896</td>
<td>200</td>
<td>0</td>
<td>2</td>
<td>1.0</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>m1.sql</td>
<td>512</td>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1.0</td>
<td>True</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>m1.bench</td>
<td>512</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1.0</td>
<td>True</td>
<td></td>
</tr>
</tbody>
</table>
Cloudy Benchmark: Serially Boot 15 VMs

- **Docker**

<table>
<thead>
<tr>
<th>action</th>
<th>count</th>
<th>max (sec)</th>
<th>avg (sec)</th>
<th>min (sec)</th>
<th>90 percentile</th>
<th>95 percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>nova.boot_server</td>
<td>15</td>
<td>4.8055100441</td>
<td>3.900927941</td>
<td>3.64957404137</td>
<td>4.56917948723</td>
<td>4.80114896297</td>
</tr>
</tbody>
</table>

- **KVM**

<table>
<thead>
<tr>
<th>action</th>
<th>count</th>
<th>max (sec)</th>
<th>avg (sec)</th>
<th>min (sec)</th>
<th>90 percentile</th>
<th>95 percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>nova.boot_server</td>
<td>15</td>
<td>7.00468301773</td>
<td>5.88419742584</td>
<td>4.84723997116</td>
<td>6.13021831512</td>
<td>6.48467411995</td>
</tr>
</tbody>
</table>

9/12/2014
Rally Reboot 5x5 Configuration

• Rally config:

```json
{
    "NovaServers.boot_and_bounce_server": [ 
        {
            "args": {
                "flavor_id": 2,
                "image_id": "ID",
                "actions": [
                    {"soft_reboot": 5}
                ],
            },
            "runner": {
                "type": "constant",
                "times": 5,
                "active_users": 1
            },
            "context": {
                "users": {
                    "tenants": 1,
                    "users_per_tenant": 1
                }
            }
        }
    ]
}
```
Cloudy Performance: Serial VM Reboot

- **Docker**

<table>
<thead>
<tr>
<th>action</th>
<th>count</th>
<th>max (sec)</th>
<th>avg (sec)</th>
<th>min (sec)</th>
<th>90 percentile</th>
<th>95 percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>nova.boot_server</td>
<td>5</td>
<td>4.96266412735</td>
<td>4.13350987434</td>
<td>3.82891011238</td>
<td>4.61819009781</td>
<td>4.79042711258</td>
</tr>
<tr>
<td>nova.delete_server</td>
<td>5</td>
<td>8.74263191223</td>
<td>7.85760216713</td>
<td>6.61050701141</td>
<td>8.72944793701</td>
<td>8.73603992462</td>
</tr>
</tbody>
</table>

- **KVM**

<table>
<thead>
<tr>
<th>action</th>
<th>count</th>
<th>max (sec)</th>
<th>avg (sec)</th>
<th>min (sec)</th>
<th>90 percentile</th>
<th>95 percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>nova.reboot_server</td>
<td>10</td>
<td>124.951740026</td>
<td>124.452507925</td>
<td>123.876129866</td>
<td>124.950992012</td>
<td>124.951366019</td>
</tr>
<tr>
<td>nova.boot_server</td>
<td>2</td>
<td>6.1228749752</td>
<td>5.58395600319</td>
<td>5.04503703117</td>
<td>6.0150911808</td>
<td>6.068983078</td>
</tr>
<tr>
<td>nova.delete_server</td>
<td>2</td>
<td>4.45799517632</td>
<td>4.45187151432</td>
<td>4.44574785233</td>
<td>4.45677044392</td>
<td>4.45738281012</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>max (sec)</th>
<th>avg (sec)</th>
<th>min (sec)</th>
<th>90 percentile</th>
<th>95 percentile</th>
<th>success/total</th>
<th>total times</th>
</tr>
</thead>
<tbody>
<tr>
<td>632.718183994</td>
<td>632.315432072</td>
<td>631.912680149</td>
<td>632.63763361</td>
<td>632.677908802</td>
<td>0.4</td>
<td>5</td>
</tr>
</tbody>
</table>
Rally Snapshot Configuration

- Rally config:
  
  ```json
  {
    "NovaServers.boot_and_bounce_server": [
      {
        "args": {
          "flavor_id": 1,
          "image_id": "IMAGE",
          "actions": [
            {"soft_reboot": 5}
          ]
        },
        "runner": {
          "type": "constant",
          "times": 5,
          "active_users": 1
        },
        "context": {
          "users": {
            "tenants": 1,
            "users_per_tenant": 1
          }
        }
      }
    ]
  }
  ```
# Cloudy Performance: Snapshot VM To Image

- **Docker (note -- defect deleting image)**

<table>
<thead>
<tr>
<th>action</th>
<th>count</th>
<th>max (sec)</th>
<th>avg (sec)</th>
<th>min (sec)</th>
<th>90 percentile</th>
<th>95 percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>nova.boot_server</td>
<td>2</td>
<td>4.120429039</td>
<td>3.85578501225</td>
<td>3.5911498549</td>
<td>4.06750023365</td>
<td>4.0939646363</td>
</tr>
<tr>
<td>nova.delete_server</td>
<td>2</td>
<td>8.61395692825</td>
<td>7.64416801929</td>
<td>6.67437911034</td>
<td>8.41999914646</td>
<td>8.51697803736</td>
</tr>
</tbody>
</table>

- **KVM**

<table>
<thead>
<tr>
<th>action</th>
<th>count</th>
<th>max (sec)</th>
<th>avg (sec)</th>
<th>min (sec)</th>
<th>90 percentile</th>
<th>95 percentile</th>
</tr>
</thead>
<tbody>
<tr>
<td>nova.delete_image</td>
<td>1</td>
<td>0.700344085693</td>
<td>0.700344085693</td>
<td>0.700344085693</td>
<td>0.700344085693</td>
<td>0.700344085693</td>
</tr>
<tr>
<td>nova.create_image</td>
<td>1</td>
<td>42.92771101</td>
<td>42.92771101</td>
<td>42.92771101</td>
<td>42.92771101</td>
<td>42.92771101</td>
</tr>
<tr>
<td>nova.boot_server</td>
<td>2</td>
<td>40.9650099277</td>
<td>22.9950100183</td>
<td>5.02501010895</td>
<td>37.3710099459</td>
<td>39.160099368</td>
</tr>
<tr>
<td>nova.delete_server</td>
<td>2</td>
<td>4.47270512581</td>
<td>4.46178817749</td>
<td>4.45087122917</td>
<td>4.47052173615</td>
<td>4.47161343089</td>
</tr>
</tbody>
</table>

## Performance Summary

- **nova.delete_image**

<table>
<thead>
<tr>
<th>max (sec)</th>
<th>avg (sec)</th>
<th>min (sec)</th>
<th>90 percentile</th>
<th>95 percentile</th>
<th>success/total</th>
<th>total times</th>
</tr>
</thead>
<tbody>
<tr>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>n/a</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

---

9/12/2014
Configuring Docker LXC Container For 2CPU x 4G RAM

• Pin container to 2 CPUs / Mem
  – Create cpuset cgroup
  – Pin group to cpuset.mems to 0,1
  – Pin group to cpuset.cpus to 0,1
  – Add container root proc to tasks

• Limit container memory to 4G
  – Create memory cgroup
  – Set memory.limit_in_bytes to 4G
  – Add container root proc to tasks
Docker Command Line

• Start docker image from docker CLI

    root@devstack-compute:~/devstack$ time docker run -d guillermo/mysql
    859f2a88adeb7190eb855eb060172fc2f137bff609f2f3be8ad8ee069d7e88f4

    real    0m0.191s
    user    0m0.004s
    sys     0m0.004s