HPC on OpenStack
the good, the bad and the ugly

Ümit Seren
HPC Engineer at the Vienna BioCenter

Github: @timeu
Twitter: @timeu_s

FOSDEM 2020 - Feb 02, 2020 - Brussels
The “Cloudster” and How we’re Building it!

Shamelessly stolen from Damien François Talk -- “The convergence of HPC and BigData: What does it mean for HPC sysadmins?” - FOSDEM 2019
Who Are We?

- Part of Cloud Platform Engineering Team at molecular biology research institutes (IMP, IMBA, GMI) located in Vienna, Austria at the Vienna Bio Center.
- Tasked with delivery and operations of IT infrastructure for ~ 40 research groups (~ 500 scientists).
- IT department delivers full stack of services from workstations, networking, application hosting and development (among many others).
- Part of IT infrastructure is delivery of HPC services for our campus.
- 14 People in total for everything.
Vienna BioCenter Computing Profile

- Computing infrastructure almost exclusively dedicated to bioinformatics (genomics, image processing, cryo electron microscopy, etc.)
- Almost all applications are data exploration, analysis and data processing, no simulation workloads
- Have all machinery for data acquisition on site (sequencers, microscopes, etc.)
- Operating and running several compute clusters for batch computing and several compute clusters for stateful applications (web apps, databases, etc.)
What We Had Before

- Siloed islands of infrastructure
- Can't talk to other islands, can't access data from other island (or difficult logistics for users)
- Nightmare to manage
- No central automation across all resources easily possible
Meet the CLIP Project

- OpenStack was chosen to be evaluated further as platform for this
- Setup a project “CLIP” (Cloud Infrastructure Project) and formed project team (4.0 FTE) with a multi phase approach to delivery of the project.
- Goal is to implement not only a new HPC platform but a software defined datacenter strategy based on OpenStack and deliver HPC services on top of this platform
- Delivered in multiple phases
What We’re Aiming At

HPC Cluster

Spark

JupyterH

OpenStack private cloud

Compute Node

Compute Node

Compute Node

Compute Node

Compute Node

Compute Node

Compute Node

Compute Node
CLIP Cloud Architecture Hardware

- Heterogeneous nodes (high core count, high clock, large memory, GPU accelerated, NVME)
- ~ 200 compute nodes and ~ 7700 Intel SkyLake cores
- 100GbE SDN RDMA capable Ethernet and some nodes with 2x or 4x ports
- ~ 250TB NVMe IO Nodes ~ 200Gbyte/s
Tasks Performed within “CLIP”

**Plan**
- **Dec. 2017**: POC
- **Feb. 2018**: Analysis
- **Oct. 2018**: Deployment
- **Jan. 2019**: Production

**Actual**
- **Dec. 2017**: POC
- **Jan. 2019**: Analysis
- **Jan. 2019**: Deployment
- **Jul. 2019**: Production

**Basic understanding**
- Small scale

**Deeper understanding**
- Deployment, tooling, operations & benchmarking

**Production deployment**
- Cloud & Slurm payload

**Interactive Application**
- JupyterHub, Rstudio

*Interactive applications on HPC systems* by Erich Birngruber at 16:00 Dez. 2017 – Feb. 2018: 2 months
Deploying and Operating the Cloud
Deploying the Cloud - TripleO (OoO)

- **TripleO (OoO):** Openstack on OpenStack
- **Undercloud:** single node deployment of OpenStack.
  - Deploys the **Overcloud**
- **Overcloud:** HA deployment of OpenStack.
  - Cloud for **Payload**
- Installation with **GUI** or **CLI**?
Deploying the Cloud - Should we use the GUI?
Deploying the Cloud - Should we use the GUI?

### Plan overcloud deployment

- **Deployment in progress**: 31%

### Resources

<table>
<thead>
<tr>
<th>Name</th>
<th>Status</th>
<th>Updated Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>MysqlRootPassword</td>
<td>CREATE_COMPLETE</td>
<td>2016-11-24T07:00:06Z</td>
</tr>
<tr>
<td>PcsdPassword</td>
<td>CREATE_COMPLETE</td>
<td>2016-11-24T07:00:06Z</td>
</tr>
<tr>
<td>VipMap</td>
<td>CREATE_COMPLETE</td>
<td>2016-11-24T07:00:06Z</td>
</tr>
<tr>
<td>RabbitCookie</td>
<td>CREATE_COMPLETE</td>
<td>2016-11-24T07:00:06Z</td>
</tr>
<tr>
<td>Controller</td>
<td>INIT_COMPLETE</td>
<td>2016-11-24T07:00:08Z</td>
</tr>
<tr>
<td>ObjectStorage</td>
<td>INIT_COMPLETE</td>
<td>2016-11-24T07:00:08Z</td>
</tr>
<tr>
<td>ObjectStorage/ipListMap</td>
<td>INIT_COMPLETE</td>
<td>2016-11-24T07:00:08Z</td>
</tr>
<tr>
<td>Controller/ipListMap</td>
<td>INIT_COMPLETE</td>
<td>2016-11-24T07:00:08Z</td>
</tr>
<tr>
<td>BlockStorageServiceChain</td>
<td>CREATE_IN_PROGRESS</td>
<td>2016-11-24T07:00:08Z</td>
</tr>
<tr>
<td>ComputeHostsDeployment</td>
<td>INIT_COMPLETE</td>
<td>2016-11-24T07:00:08Z</td>
</tr>
<tr>
<td>RedisVirtualIP</td>
<td>CREATE_COMPLETE</td>
<td>2016-11-24T07:00:08Z</td>
</tr>
<tr>
<td>CloudInit</td>
<td>CREATE_COMPLETE</td>
<td>2016-11-24T07:00:08Z</td>
</tr>
</tbody>
</table>
Deploying the Cloud - Code as Infra & GitOps!

- Web GUI does not scale
  - → Disable the Web UI and deploy from the CLI
- TripleO internally uses **heat** to drive **puppet** that drives **ansible**  \( (ツ)_/\)
- Use **ansible** to drive the TripleO installer and rest of infra
- Entire end-2-end deployment from code
Deploying the Cloud - Pitfalls and Solutions!

- TripleO is slow because Heat → Puppet → Ansible !!
  - Update takes ~ 60 minutes even for simple config change
- Customize using ansible instead ? Unfortunately not robust :-(
  - Stack update (scale down/up) will overwrite our changes
  - → services can be down
- → Let’s compromise: Use both
  - Iterate with ansible → Use TripleO for final configuration
- Ansible everywhere else !
  - Network, Moving nodes between environments, etc
Operating the Cloud - Package Management

- 3 environments & infra as code: reproducibility and testing of upgrades
- What about software versions? → **Satellite/Foreman** to the rescue!
- Software Lifecycle environments <-> Openstack environments

Lifecycle Environment Paths

<table>
<thead>
<tr>
<th>Library</th>
<th>Content Views</th>
<th>Products</th>
<th>Yum Repositories</th>
<th>Docker Repositories</th>
<th>Packages</th>
<th>Errata</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>7</td>
<td>14</td>
<td>99</td>
<td>52289</td>
<td>5100</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>Dev</th>
<th>Staging</th>
<th>Prod</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content Views</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>Content Hosts</td>
<td>8</td>
<td>8</td>
<td>199</td>
</tr>
</tbody>
</table>
Operating the Cloud - Package Management

1. Create **Content Views** (contains RPM repos and containers)

2. **Publish** new versions of Content Views

3. **Test** in dev/staging and **roll** them **forward** to production

<table>
<thead>
<tr>
<th>Version</th>
<th>Status</th>
<th>Environments</th>
<th>Content</th>
<th>Description</th>
<th>Actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version 17.0</td>
<td>Published (2020-01-10 14:05:04 +0100)</td>
<td>Library</td>
<td>51050 Packages 5026 Errata (893 🔄 3346 🔴 787 🔴)</td>
<td>Updated RHEL OS base packages, OSP13 RPMs and Cisco ACI RPMs (4.2.3)</td>
<td>Promote</td>
</tr>
<tr>
<td>Version 11.0</td>
<td>Promoted to Library (2019-10-21 15:50:38 +0200)</td>
<td>Staging Prod</td>
<td>46413 Packages 4474 Errata (766 🔄 2983 🔴 725 🔄)</td>
<td>Upgrade OSP13 packages</td>
<td>Promote</td>
</tr>
</tbody>
</table>
Operating the Cloud - Tracking Bugs in OS

- How to keep track of bugs in OpenStack?
- Track bugs, workaround and the status in JIRA project (CRE)
Deploying and operating the Cloud - Summary

Lessons learned and pitfalls of OpenStack/TripleO:

- OpenStack and TripleO are complex pieces of software
  - Dev/staging environment & package management
- Upgrades can break the cloud in unexpected ways.
  - OSP11 (non-containerized) → OSP12 (containerized)
- Containers are no free lunch
  - Container build pipeline for customizations
- TripleO is a supported out of the box installer for common cloud configurations
  - Exotic configurations are challenging
- “Flying blind through clouds is dangerous”:
  - Continuous performance and regression testing
- Infra as code (end to end) way to go
  - Requires discipline (proper PR reviews) and release management
Cloud Verification & Performance Testing
Cloud verification & Performance Testing

- How can we make sure and monitor that the cloud works during operations?
- We leverage OpenStack’s own tempest testing suite to run verification against our deployed cloud.
- First smoke test (~ 128 tests) and if this is successful run full test (~ 3000 tests) against the cloud.
Cloud verification & Performance Testing

- How can we make sure and monitor that the cloud works during operations?
- We leverage OpenStack’s own tempest testing suite to run verification against our deployed cloud.
- First smoke test (~ 128 tests) and if this is successful run full test (~ 3000 tests) against the cloud.
Cloud verification & Performance Testing

- Ok, the Cloud works but what about performance? How can we make sure that OS performs when upgrading software packages etc?
- We plan to use **Browbeat** to run **Rally** (control plane performance/stress testing), **Shaker** (network stress test) and **PerfkitBenchmarker** (payload performance) tests on a regular basis or before and after software upgrades or configuration changes.
Cloud verification & Performance Testing

- Ok, the Cloud works but what about performance? How can we make sure that OS performs when upgrading software packages etc?
- We plan to use Browbeat to run Rally (control plane performance/stress testing), Shaker (network stress test) and PerfkitBenchmarker (payload performance) tests on a regular basis or before and after software upgrades or configuration changes
Cloud verification & Performance Testing

- Ok, the Cloud works but what about performance? How can we make sure that OS performs when upgrading software packages etc?
- We plan to use **Browbeat** to run **Rally** (control plane performance/stress testing), **Shaker** (network stress test) and **PerfkitBenchmarker** (payload performance) tests on a regular basis or before and after software upgrades or configuration changes
Cloud verification & Performance Testing

- Grafana and Kibana dashboard can show more than individual rally graphs:

- Browbeat can show differences between settings or software versions:

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Action</th>
<th>conc.</th>
<th>times</th>
<th>0b5ba58c</th>
<th>2b177f3b</th>
<th>% Diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>create-list-router</td>
<td>neutron.create_router</td>
<td>500</td>
<td>32</td>
<td>19.940</td>
<td>15.656</td>
<td>-21.483</td>
</tr>
<tr>
<td>create-list-router</td>
<td>neutron.list_routers</td>
<td>500</td>
<td>32</td>
<td>2.588</td>
<td>2.086</td>
<td>-19.410</td>
</tr>
<tr>
<td>create-list-router</td>
<td>neutron.create_network</td>
<td>500</td>
<td>32</td>
<td>3.294</td>
<td>2.366</td>
<td>-28.177</td>
</tr>
<tr>
<td>create-list-router</td>
<td>neutron.create_subnet</td>
<td>500</td>
<td>32</td>
<td>4.282</td>
<td>2.866</td>
<td>-33.075</td>
</tr>
<tr>
<td>create-list-port</td>
<td>neutron.list_ports</td>
<td>500</td>
<td>32</td>
<td>52.627</td>
<td>43.448</td>
<td>-17.442</td>
</tr>
<tr>
<td>create-list-port</td>
<td>neutron.create_network</td>
<td>500</td>
<td>32</td>
<td>4.025</td>
<td>2.771</td>
<td>-31.165</td>
</tr>
<tr>
<td>create-list-port</td>
<td>neutron.create_port</td>
<td>500</td>
<td>32</td>
<td>19.458</td>
<td>5.412</td>
<td>-72.189</td>
</tr>
<tr>
<td>create-list-subnet</td>
<td>neutron.create_subnet</td>
<td>500</td>
<td>32</td>
<td>11.366</td>
<td>4.809</td>
<td>-57.689</td>
</tr>
<tr>
<td>create-list-subnet</td>
<td>neutron.create_network</td>
<td>500</td>
<td>32</td>
<td>6.432</td>
<td>4.286</td>
<td>-33.368</td>
</tr>
<tr>
<td>create-list-subnet</td>
<td>neutron.list_subnets</td>
<td>500</td>
<td>32</td>
<td>10.627</td>
<td>7.522</td>
<td>-29.221</td>
</tr>
<tr>
<td>create-list-network</td>
<td>neutron.create_network</td>
<td>500</td>
<td>32</td>
<td>15.154</td>
<td>13.073</td>
<td>-13.736</td>
</tr>
<tr>
<td>create-list-network</td>
<td>neutron.list_networks</td>
<td>500</td>
<td>32</td>
<td>10.200</td>
<td>6.595</td>
<td>-35.347</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>UUID</th>
<th>Version</th>
<th>Build</th>
<th>Number of runs</th>
</tr>
</thead>
<tbody>
<tr>
<td>938dc451-881-4f2f8-a6cb-ad502b177f3b</td>
<td>queens</td>
<td>2018-03-20.2</td>
<td>1</td>
</tr>
<tr>
<td>6b50b6f7-445a-4c53-78200b5ba58c</td>
<td>ocata</td>
<td>2017-XX-XX.X</td>
<td>3</td>
</tr>
</tbody>
</table>
Deploying the Payload
Deploying the Cloud - SLURM Cluster

- 2 step process:
  - OpenStack **Heat** to provision → **Ansible inventory**
  - **Ansible** playbook/roles\(^1\) for config → **SLURM cluster**

- Satellite for package management

- Dev & staging env for testing → roll over to production

- Deploy other complex systems (Spark cluster, k8s, etc)

[1] - StackHPC ansible roles: https://github.com/stackhpc
Deploying the Cloud - Tunings for HPC

- Tuning, Tuning, Tuning required for excellent performance

<table>
<thead>
<tr>
<th>Tuning</th>
<th>Caveats / Downside</th>
</tr>
</thead>
</table>
| NUMA clean instances (KVM process layout)   | No live migrations  
No mixing of different VM flavors                                                  |
| Static huge pages (KSM etc.) setup          | If not enough memory is left to hypervisor  
→ swapping or host services get OOM.  
No mixing of different VM flavors            |
| Core isolation (isolcpus)                   | Performance drop in virtual networking  
performance → SR-IOV                                                                  |
| PCI-E passthrough (GPUs, NVME) and SR-IOV (NICs) | No live migrations and less features  
compared to fully virtualized networking                                              |
Deploying the Cloud - Pitfalls and Issues

- Ansible is slow: Slurm playbook takes ~1 hour (clean 2nd run !)
  - Use tags for recurring day 2 operations (i.e new mount points, change of QOS, etc)
- Satellite 👍 for software versions but remove upstream Centos repos after install
- Some issues only hit under scale:
  - SDN scaling issues when provisioning more than 70 nodes. Workaround: scale in batches
- Isolation of environments ends with shared infra components especially when tightly integrating with OpenStack
  - Update of DEV environment caused datacenter wide network outage (bug in SDN)
- Beware of unintended consequences of code changes
  - Triggered accidental re-deploy of payload because of single line change in heat template
# HPC on OpenStack - Lessons Learned

<table>
<thead>
<tr>
<th>Bad &amp; Ugly</th>
<th>Good</th>
</tr>
</thead>
<tbody>
<tr>
<td>OpenStack is <em>incredibly</em> complex</td>
<td>Open source software with commercial support</td>
</tr>
<tr>
<td>OpenStack is not a product. It is a framework.</td>
<td>OpenStack integrates well with existing datacenter infrastructure</td>
</tr>
<tr>
<td>You need 2-3 OpenStack environments (development, staging, prod in our case) to practice and understand upgrades and updates.</td>
<td>API driven software defined datacenter</td>
</tr>
<tr>
<td>Scaling above certain amount of nodes will be an issue</td>
<td>Easily deploy multiple payloads side by side like in a Cloud 😏</td>
</tr>
<tr>
<td>Cloud networking is really hard (especially in our case)</td>
<td>Covers a wide range of use cases ranging from virtualized &amp; baremetal HPC clusters to container orchestration engines</td>
</tr>
</tbody>
</table>
Acknowledgements

HPC Team

Erich Birngruber
Petar Forai
Petar Jager
Ümit Seren

Thanks