OST-104 - Openstack private cloud workshop

Student guide

Release L rev148

Component Soft Ltd.

November 26, 2016

Contents

Preface

Formatting notes

Here we present some examples of a the formatting applied in this document.

Note:
Here we describe the formatting rules of this book. Take the following commands as examples only. These commands not necessary execute correctly or produce the same output for your actual setup.

Example 1

```
root@controller1 (admin) $> nova list --minimal
+--------------------------------------+--------+
| ID                                   | Name   |
+--------------------------------------+--------+
| 58012abf-1b73-40ec-989c-96f4592cd277 | test_1 |
+--------------------------------------+--------+
```

In this case

- The command runs on the `controller` node, as user `root`.
- The keystone credentials for tenant `admin` are loaded into the shell environment.

In the above case it is required to load the admin credentials in order to make the certain command to work properly. In case of the `admin` user (and tenant), it can be done by sourcing the file `/root/keystonerc_admin`

```
root@controller1 $> source /root/keystonerc_admin
root@controller1 (admin) $> env |grep OS_
OS_REGION_NAME=RegionOne
OS_PASSWORD=makeitso
OS_AUTH_URL=http://10.10.10.51:5000/v2.0/
OS_USERNAME=admin
OS_TENANT_NAME=admin
```

Commands not related to OpenStack (like `ls`) does not take care of the OpenStack credentials at all, so for those, it is irrelevant whether you can `OS_` environment variables or not.

Example 2

```
root@controller1 (admin) $> nova interface-list vm1

// Port ID                              | Net ID                                >>
//-------------------------------------+---------------------------------------->>
// 7912e922-e871-4e7a-a943-34017ee29160 | 41f61be7-40b9-4a67-aeca-feae3a5986ac>>
// 93925c7f-0112-493c-8a39-261449128f5f | e3ee2d62-e216-4e76-b438-733e706f1500>>

 IP addresses //
--------------//
 10.40.40.100 //
 10.30.30.102 //
```

In this one, the output is too long, so the first and last columns are cut off (`//` symbols) and the third column is presented separately (>> symbol)

Example 3

```
root@controller1 (admin)$> nova host-describe compute2.openstack.local
+-------------------------+----------------------+-----+-----------+------+
| HOST                    | PROJECT              | cpu | memory_mb | disk |
+-------------------------+----------------------+-----+-----------+------+
| compute2.openstack.local | (total)              | 8   | 15948     | 4    |
| compute2.openstack.local | (used_now)           | 4   | 2560      | 4    |
| compute2.openstack.local | (used_max)           | 4   | 2048      | 4    |
| compute2.openstack.local | 0cb8d/--/c274e67     | 4   | 2048      | 4    |
+-------------------------+----------------------+-----+-----------+------+
```

In this case the second column `PROJECT` is truncated to 20 characters, so UUID `0cb8d6ab778546bbadc69488dc274e67` is shortened to `0cb8d/--/c274e67`.

Module 1: Introduction

Introduction

- Cloud computing in general
- Overview of Openstack
- Core Projects
- OpenStack Architecture
- Virtual Machine Provisioning Walk-Through

Cloud computing

- a model for enabling ubiquitous network access to a shared pool of configurable computing resources*
  - resources (compute, storage) as services
    - resources are allocated on demand
      - scaling and removal also happens rapidly (seconds-minutes)
    - multi-tenancy
      - share resources among thousands of users
      - resource quotas
  - cost effective IT
    - Pay-As-You-Go model
      - pay per hour/gigabyte instead of flat rate
    - maximized effectiveness of the shared resources
      - maybe over-provisioning
    - lower barriers to entry (nice for startups)
      - focus on your business instead of your infrastructure

*definition by NIST

Cloud computing, in general, provides resources, such as compute instances, storage objects and virtual networks to its customers. These resources can be allocated/resized/dropped any time, and their usage is payed on a per minute/per hour basis (for a public cloud). This unparalleled flexibility allows small companies, like start-ups, to focus rather on their new business, instead of building up a local infrastructure.

Cloud computing also means virtualization of resources, which makes an effective use, or full utilization of hardware resources. In case of resource demanding applications (high CPU utilization, a lot of IO operations), this approach could easy lead to over-utilization.

Cloud types

- By service model
  - Infrastructure as a Service (IaaS)
    - Virtual or physical machines (MaaS)
      block storage, virtual networking (FW,LB,VPN), object store
    - Examples: AWS, OpenStack, Azure, VmWare VCenter
  - Platform as a Service (PaaS)
    - provides a middleware (OS, DB,etc maintained by the provider)
    - Examples: OpenShift, Heroku, Google App Engine
  - Software as a Service (SaaS)
    - shared access to a software (like ERP, DB or even desktop)
    - Examples: Gmail, Instagram, Adobe
- By location
  - Public cloud
    - Multi-region, shared deployment of services
  - Private cloud
    - On premise deployment, mainly for security
  - Hybrid cloud

Cloud computing offers different service models depending on the capabilities a consumer may require.

- IaaS (Infrastructure-as-a-Service)

  It provides infrastructure such as computer instances, network connections, and storage so that people can run any software or operating system. IaaS systems usually provide a set of prepared operating system images, from which end-users can start new virtual machine instances with a single click. The networking between these VMs, and additional file/object storage space is also provided by the IaaS infrastructure.

  Several IaaS providers can also provision bare-metal servers (Metal as a Service or MaaS), allowing their customers direct access to a physical hardware.

- PaaS (Platform-as-a-Service)

  With PaaS, the consumer has the ability to deploy applications through a programming language or tools supported by the cloud platform provider. An example of Platform-as-a-Service is OpenShift by RedHat. Built on top of an IaaS (Amazon Elastic Compute), it provides JBoss/FireFly as service where Java developers can deploy their applications

without taking care of the underlying operating system, database or Java runtime.

- SaaS (Software-as-a-Service)

  The consumer uses a software in a cloud environment, without needing to install anything to the local computer. The simplest example of SaaS is a web-based mail service, but recently more resource demanding applications (like Adobe Photoshop) are available as a cloud service.

By the location of the cloud servers, we can also distinguish between

- Public clouds

  Cloud resources are available from everywhere to everyone.

- Private clouds

  In this case the cloud infrastructure is installed on the site of the company, and resources can be allocated by employees only. Such design allows full control over sensitive set of data. In some cases companies required by law to store their data on a server located in the same country.

- Hybrid clouds

  This solution aims to combine the security of private clouds with the flexibility of public clouds.

Clouds – the flip side

- Large software monoculture
  - a single update has effect on thousands of hypervisors
- Not always so cheap
  - Long term TCO for on-premise may gets cheaper
- Security in public clouds
  - loss of control on sensitive data
  - country regulations
    - the data has to be stored in the same country
  - attractive for hackers
    - infinite time for finding security holes
    - hyperjacking
  - “incorrect” privacy policies
    - data altered/deleted by the providers
    - data share with third parties

Using a cloud software, however, can have disadvantages.

- Large software monoculture

  In order to make clouds easy to manage, cloud deployers set up hundreds of machines with (almost) identical software configuration. This approach, however, can be dangerous: if something goes wrong with a software update, it might affects the entire cloud infrastructure.

- Public cloud is not always that cheap

  Public cloud resources are cheap for a smaller number of instances, for short term use. Beyond of the limit of ~100 cores, however, it could be more cost effective to run your own servers, with your own staff, on your local site.

- Security

  Cloud providers are very attractive to hackers. If you find a single security hole for a cloud provider, you can compromise thousands of VMs, and you may access the private data

of several companies.

- Cloud privacy policy

  A common criticism of SaaS providers (like Facebook or Google) is that they handle data privacy incorrectly. Sharing personal data with third parties, or using it for direct marketing is permitted in their privacy policies, but such behavior is not acceptable to everyone

Overview

- OpenStack controls large pools of compute, storage, and networking resources throughout a data center
- Features
  - On-demand self-service
    - Users can automatically provision needed compute/network resources through a REST API/dashboard
  - Network access
    - All computing resources are available over network
    - SDN: User can define complex network topologies (routers, subnets)
    - FWaaS, LBaaS
  - Elastic
    - Provisioning is rapid and scales out as needed
  - Metered or measured service
    - Monitoring and reporting of resource usage for both providers and consumers.

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources through a datacenter. It can all be managed through a dashboard called Horizon, that gives administrators control, while empowering users to provision resources through a web interface on their own.

OpenStack is a global collaboration of developers and cloud computing technologists, producing a ubiquitous open source cloud computing platform for public and private clouds. The project aims to deliver solutions for all types of clouds by being

- Simple to implement
- Massively scalable
- Feature rich

OpenStack Foundation

- Founded by RackSpace Hosting and NASA back in 2012
  - promotes the development, distribution and adoption of the OpenStack
  - attracted more than 28000 individuals members and over 1000 organizations (keeps growing)
- A “software meritocracy”
  - Openstack is composed of dozens of core projects
    - Each project has a Program Technical Lead (PTL)
  - All developments are controlled by The Technical Committee
    - an elected group that represents the contributors
- Members of the Foundation
  - Individual members
    - Anyone who wants to contribute (code, documentation,bug reports, testing)
  - Corporate Members and sponsors
    - Provide dedicated resources (developers, infrastructure) and funding for ongoing activities

The OpenStack Foundation was established in September of 2012 as an independent body, providing shared resources to help achieve the OpenStack Mission by protecting, empowering, and promoting OpenStack software and the community around it. This includes users, developers and the entire ecosystem. As the independent home for OpenStack, the Foundation has already attracted more than 28000 individual members from 140 countries and over 1000 different organizations.

The governance type of OpenStack Foundation is a “software meritocracy”. Technical decision making is placed in the hands of technical leaders who strive to put the interests of the projects and software ahead of corporate affiliation.

- Program Technical Leads (PTLs) lead individual programs.

  A PTL is ultimately responsible for the direction for each OpenStack Core Projects, makes tough calls when needed, organizes the work and teams in the program and determines if other forms of program leadership are needed. The PTL for core project is elected by the body of contributors to that particular project.

- The Technical Committee oversees the entire set of OpenStack projects.

  The TC is one of the governing bodies of the OpenStack project. It is an elected group that represents the contributors to the project, and has oversight on all technical matters.

Members of the OpenStack Foundation are

- Individual members

  Individuals contributing to OpenStack in a variety of ways such as code, documentation, translations, bug reports, testing.

- Corporate members
  - "Platinum" or "Gold" member companies

    Provide dedicated resources (developers, infrastructure) and funding for ongoing activities, and elect/appoint members to the Board of Directors

  - Corporate sponsors

    Corporate Sponsors provide additional funding to support the Foundation's mission of protecting, empowering and promoting OpenStack

Contributing to Openstack

- Open for anyone
  - Bugfixes, tests
  - Blueprints/specifications
  - Documentation
- Become good developer
  - Communicate
    - IRC, Mailing lists, Project Meetings
  - Do code review (anyone can be a reviewer)
  - Work on bugs/blueprints
  - Contribute regularly
- Getting started
  - Need to setup accounts
    - Launchpad/Github
  - Need to sign the CLA (Contributor License Agreement)
  - Clone a project, make a branch, change, test, commit

Contributing to OpenStack is open for anyone. After setting up an account you can be involved in the development.

- Report bugs

  Most users just report bugs/problems found in OpenStack.

- Fixing bugs

  The first area where you can help is bug fixing. You can contribute instructions on how to fix a given bug, and set it to *Triaged*. Or you can directly fix it: assign the bug to yourself, set it to *In progress*, branch the code, implement the fix, and propose your change for merging into trunk.

  Some easy-to-fix bugs may be marked with the *low-hanging-fruit*, good targets for a beginner.

- Blueprints/specifications

Before implementing a huge change, try to summarize your idea in a blueprint, and discuss that with project members. If the blueprint is accepted, you can assign it to yourself, and work on it.

- Documentation

  Documentation is also developed by the community. Young projects are usually poorly documented, so writing a good documentation can give a big help.

To become a highly valued contributor, you should do several things

- Communicate

  Be active on the IRC channel and mailing list of a project, attend the regular project meetings

- Do code review

  Every patch submitted to OpenStack gets reviewed before it can be approved and merged. Everyone can - and is encouraged to - review existing patches. If you are planning on submitting patches of your own, it is a great way to learn about what the community cares about and to learn about the codebase.

- Work on bugs/blueprints

  Summit bugs, write blueprints, assign them to yourself. Try to provide a fix or solution that is consistent with the codebase. When making a change, do it little by little, make smaller patches, as those are easier for review.

- Contribute regularly

  Dedicate time (if you can) regularly to contribute code.

If you want to contribute your first line of code to OpenStack, you have to setup accounts.

- You will need a Launchpad account, since this is how the Web interface for the Gerrit Code Review system will identify you. This is also useful for automatically crediting bug fixes to you when you address them with your code commits.
- You also have to visit https://review.openstack.org/, and sign in with your Launchpad ID. You have to choose a unique user name, and upload SSH keys, in order to commit you changes later.
- You also have to agree to the Individual Contributor License Agreement and provide contact information.
- Once you are done with the administration part, you can start coding. The high level overview of steps are

  - You can clone the code for the desired project you want to work on (`git clone`)
  - You should create your own branch to work on (*git branch*)
  - Make your changes
  - Run the unit tests of the project to be sure that you change does not break anything.
  - Send your code to code review (`git review`)
  - The code review is done in Gerrit, and the procedure itself is meritocratic
    * Any developer can do review, and vote one the given change
    * Your code needs at least 2 core reviewer votes to be accepted into the main branch (+2 votes)

Note:
Do not expect your first code review be successful

References:

- How to contribute to OpenStack
- Developers guide
- 7 Habits of Highly Effective Contributors
- OpenStack Infrastructure and Project Status

Certified Openstack Administrator (COA)

- Official exam directly from the OpenStack Foundation
  - Available since May 2016
- Performance-based
  - 150 minutes, ~ 30 tasks
  - $300, 1 free retake, renewal after 2 years
  - Hands-on experience is really required
    - No time for googling out the solution
  - Partial completion of a task is possible
- Virtual (browser-based)
  - Shared desktop, webcam, microphone, remote proctor
  - No other windows on the desktop
    - Just a single browser window with two tabs (Terminal,Dashboard)
    - No online docs (use man, or CLI help)

Certified OpenStack Administrator (COA) is the first professional certification offered by the OpenStack Foundation. It's designed to help companies identify top talent in the industry, and help job seekers demonstrate their skills.

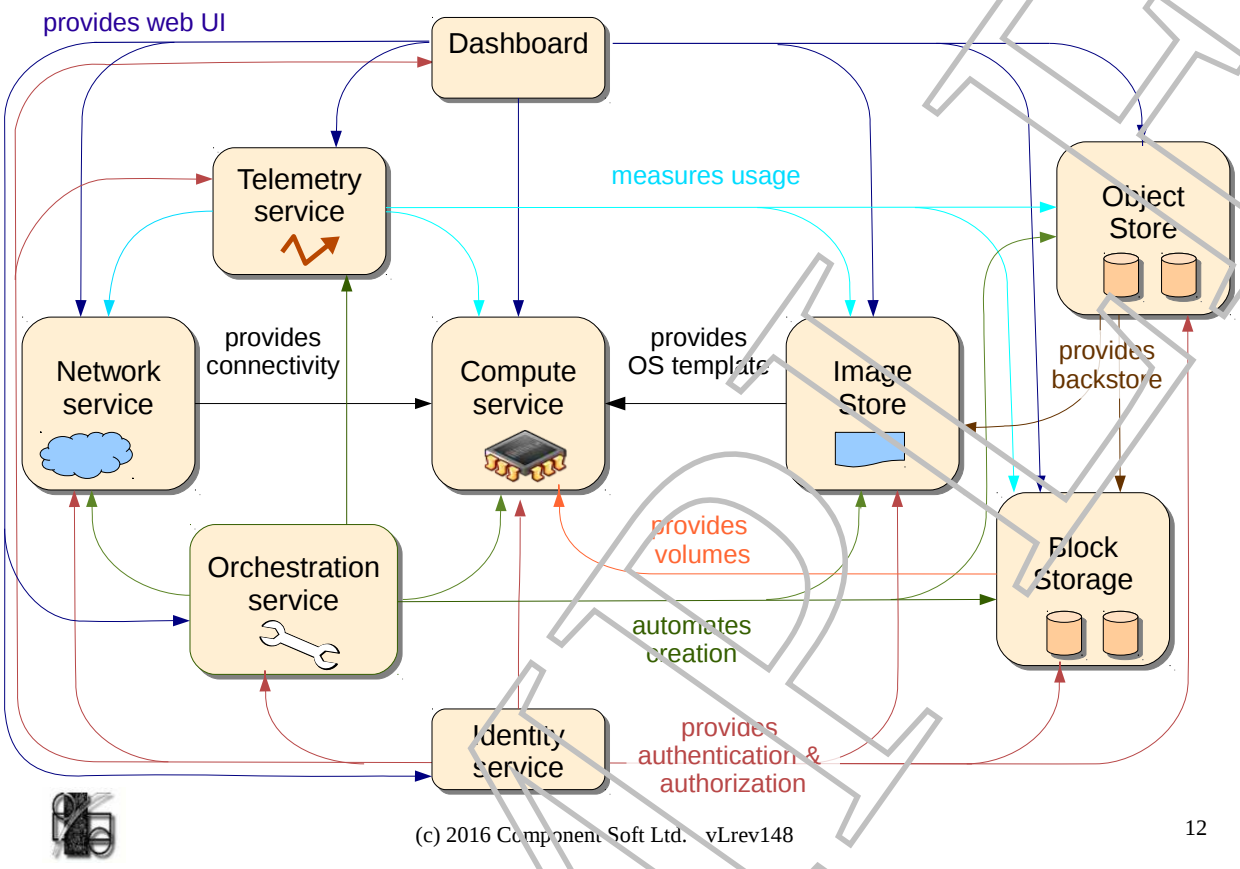
To take the exam, one needs several months of OpenStack experience, and skills required to provide day-to-day operation and management of an OpenStack cloud. The complete list of knowledge requirements is available at http://www.openstack.org/coa/requirements.

The exam is performance-based: candidates have 150 minutes to solve problems, and perform configuration tasks on a live OpenStack environment. One really needs to have hands-on experience with OpenStack to solve all tasks on time.

A partial completion of a task is possible, so that a percentage of the total score is granted, even if the candidate did not manage solve the task completely.

The exam is performed virtually (at your home, or at your office), and requires Chrome or Chromium browser. Candidates are monitored virtually by a proctor during the exam session via streaming audio, video, and screensharing feeds (including all monitors). Exam task has to be performed in the Terminal or in Horizon dashboard window provided by the exam system. Most of the tasks can be performed from the dashboard.

During the exam, the participant cannot use her/his notes, or the external documentation (like docs.openstack.org). He or she must rely on manual pages of CLI help.



Apparently, the architecture of OpenStack is quite complex.

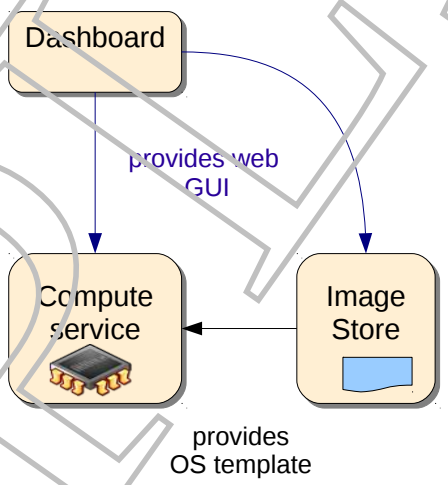
OpenStack is defined as a collection of independently developed software projects, glued together. Each service has an HTTP REST interface to communicate with OpenStack clients. Check the latest list of services in the OpenStack Roadmap

At first glance, all services use the identity service, since every single service has to validate the authentication *token* received from its clients. On the other hand, the Identity service also provides a catalog of REST endpoints, so every service has to register itself into the catalog to make itself known to OpenStack clients.

The other general service is the Dashboard, nicknamed **Horizon**, that is a Python/Django based Web GUI, with plugins for all other projects in OpenStack.

Core Projects

- Nova (Compute)
  - Provides Compute service (now) and network/block storage (earlier)
  - Mainly written in Python (uses Eventlet, Kombu (AMQP), SQLAlchemy)
  - Works with widely available virtualization technologies
    - (KVM, Xen, HyperV, VmWare, LXC, Qemu)
  - Runs on ARM
- Horizon (Dashboard)
  - Django based web GUI
  - Customizable by vendors
- Glance (Image)
  - Bootable images for instances
    - predefined images by admin
    - user created images/snapshots
  - Support for variety of formats
    - Raw, VDI,VMDK, OVF, qcow2, AMI



- The heart of the whole OpenStack installation is the Compute service (code name **Nova**).

  This service is responsible for the creation of virtual machines on the top of a hypervisor (such as KVM, Xen, VMWare, etc.).

The Compute service is discussed in Module 3.

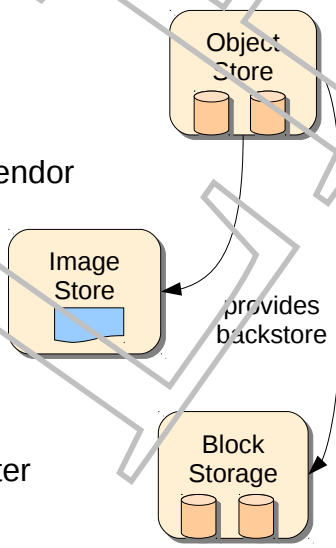
- The Image service (code name **Glance**)

  This is a simple storage for pre-installed OS images. Images can either be stored locally on the server where Glance is running, or uploaded to remote storage, such as the Object Store.

Glance is discussed in Module 2.

Core projects (2)

- Cinder (Block storage)
  - Persistent Block storage/ bootable volumes
  - Create/attach/detach/delete is integrated to Compute/Dashboard
  - Uses LVM as default,
    - but Ceph (RDS), NetApp/Nexenta (ISCSI), or FC vendor solutions are available
  - Snapshot/clone support
- Swift (Object store)
  - Scalable, redundant object store
    - objects are written to multiple disk drives spread throughout servers
    - data replication and integrity is ensured across cluster (rsync)
  - Uses inexpensive/commodity HW
  - Acts as backend for Glance,Keystone, (Cinder)
    - other options are Ceph, GlusterFS



- Block Storage service (code name **Cinder**)

  Besides OS images, we also need additional volumes to store data. These volumes are provided by **Cinder**. This service can also use the Object store to create backups of data volumes.

Cinder is discussed in Module 2.

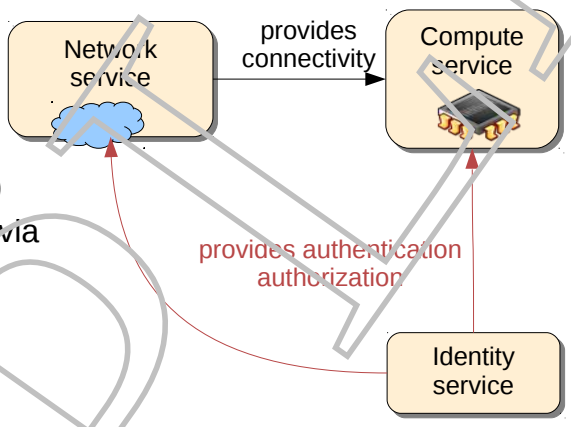
- Object Store (code name **Swift**)

  The Object Store is a petabyte scale storage built from up to hundreds of commodity servers and cheap disks. Files are up/downloaded via a simple REST/HTTP interface.

Project Swift is discussed in detail in Module 7.

Core projects (3)

- Keystone (Identity)
  - Common authentication/authorization service for other services
    - supports user/pass, token, EC2 type credentials
  - Users/tenants/roles, RBAC for each service (policy.json)
  - Quotas
  - Can integrate to external resources
    - such as LDAP,SQL,PAM
- Neutron/Quantum (Network)
  - Tenant-specific networks (since Grizzly)
  - Complex network topologies, tunneling via GRE,VLAN,VxLAN
    - works with non-smart switches
    - vendor plugins for smart ones
  - Private/Floating IPs
  - Allows additional services as FwaaS, LBaaS,CloudPipe



- The central and most important component is the identity service called **Keystone**.

  This service acts as a Single Sign On Service, and works similar to Kerberos. OpenStack clients (the python based CLI, or even the Horizon dashboard) first authenticate to Keystone, then they use the Keystone provided token (“ticket”) to talk to the desired service (such as Compute). There is no need to re-authenticate until the token expires.

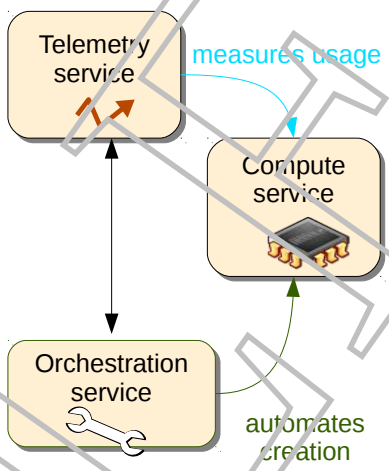
Keystone is discussed in detail in Module 2.

- The Network service (code name **Neutron**)

  It provides Software Defined Networking (SDN) functionality for OpenStack. With Neutron, users can build up complex network topologies to connect their VMs.

Neutron is discussed in detail in Module 4.

Core projects (4)



- Heat (Orchestration)
  - Orchestration:
    - automate the launch of VMs
    - install/configure software stacks
    - monitor services, scale out on demand
      basic CloudWatch support
  - Templates (stacks):
    - support for AWS CloudFormation
- Ceilometer (Telemetry)
  - monitoring and metering instance activity (network/cpu/etc)
    - provides data for usage based billing (instead of flatrate)
      billing itself is out of scope
    - sample: meter + resource_id +metadata
    - openstack services send samples via AMQP
      - or via UDP to external systems like statd
    - central processing is done at collector
      - transforming, store to back-end (MongoDB,MySQL,HBase)
      - alarming capabilities (log, web callback)

- The telemetry service (called **Ceilometer**)

  This service collects resource usage data, such as CPU cycles for virtual machines, storage space used out of the Block Storage. This telemetry data can be used to create bills for customers. It service also capable of raising alarms in case of resource overuse (such as a run of a CPU hog application).

Ceilometer is discussed in detail in Module 5.

- The orchestration service, nicknamed **Heat**

  It makes it easier to provision a larger set of dependent resources. With Heat you can download and launch templates (called “stacks”). A stack can start, for example, multiple running VMs provisioned with the same software stack. In collaboration with the telemetry service, it could also scale up/down a stack on demand.

Project Heat is discussed in detail in Module 6.

Core projects (5)

- Trove (Database)
  - DBaaS
    - starts a separate VM with special image/settings/HW
    - different pricing applies (like per I/O)
    - allows db configuration outside the VM
  - Based on MySQL (mainly)
    - since Juno, one could choose Redis, MongoDB, etc.

- Project **Trove** appeared first in the IceHouse release. It provides Database As A Service, mainly based on MySQL.

  The advantages of using Trove are:

  – It is running on a machine with a stronger IO capability.

  – Pricing is made on IOPS instead of CPU cycles.

  – Built-in replication/HA of the database is provisioned by the OpenStack infrastructure.

Trove is not discussed in detail on this training. Read more about Trove at the documentation.

Openstack releases

- Time based development cycles
  - New release every 6 months (Apr/Oct)
    - http://status.openstack.org/release
    - https://wiki.openstack.org/wiki/Releases
    - https://en.wikipedia.org/wiki/OpenStack#Release_history

1.	Austin	21/01/2010	Nova,Swift
2.	Bexar	03/02/2011	+Glance
3.	Cactus	15/04/2011	-
4.	Diablo	22/09/2011	-
5.	Essex	05/04/2012	+Horizon, +Keystone
6.	Folsom	27/09/2012	+Quantum, +Cinder
7.	Grizzly	04/04/2013	-
8.	Havanna	17/10/2013	Quantum=Neutron,+Ceilometer +Heat
9.	Icehouse	17/04/2014	+Trove
10.	Juno	16/10/2014	(+Ironic,+Zaqar, +Sahara)
11.	Kilo	30/04/2015	+Ironic (+Manila,+Barbican,+Designate,+Murano)
12.	Liberty	15/10/2015	+Murano, +Manila,(+Barbican,+Designate)
13.	Mitaka	06/04/2016	+Senlin,(+Congress)
14.	Newton	06/10/2016	Cloudkitty, Congress, Freezer, Mistral,Panko,Senlin,Solum,Tacker,Vitrage,Watcher
15.	Ocata	22/02/2017	
16.	Pike	TBD	
17.	Queens	TBD	

Here we talk about the releases of OpenStack. Each release is given a code name, such as the “A” release is called “Austin”, similar to the Ubuntu Linux naming pattern. Some of the releases introduced new core projects, for example “Bexar” was the first release where project Glance appeared.

Since the Grizzly release, release dates follow the Ubuntu release cycle: there is a new release every 6 months, one in April and another in October.

New releases usually introduce new projects. Here we introduce some of them:

- **Ironic first published in Juno release**
  - A project for the so-called “Bare-Metal Provisioning”: instead of creating virtual machines, a dedicated physical machine is provisioned for the user. It is useful when one needs so large a set of CPUs or memory, that it takes all physical resources of a physical node, so using a hypervisor is just an unnecessary overhead.
  - The other reason to use Ironic is when you need to access physical resources of a machine (such as PCI cards), but using PCI-passthrough is not an option.

Learn more of Ironic at the Ironic User Guide

- Sahara first published in Juno release
  - This is a project for “Data Processing As A Service” in OpenStack. it aims to provide the same functionality as Amazon Elastic MapReduce (Amazon EMR), based on the data crunching software Hadoop or Spark
- Barbican first published in Kilo release
  - Barbican is the OpenStack Key Manager service. It provides secure storage, provisioning and management of secret data. This includes keying material such as Symmetric Keys, Asymmetric Keys, Certificates and raw binary data.
- Designate first published in Liberty release
  - Designate provides DNSaaS services for OpenStack:
    - REST API for domain/record management
    - Multi-tenant
    - Integrated with Keystone for authentication
    - Framework in place to integrate with Nova and Neutron notifications (for auto-generated records)
    - Support for PowerDNS and Bind9 out of the box
- Senlin first published in Mitaka release
  - The mission of the Senlin project is to provide a generic clustering service for an OpenStack cloud. Such a service is capable of managing the homogeneous objects exposed by other OpenStack components, such as Nova, Heat, Cinder etc.

```
controller1> nova-manage version 2>/dev/null
12.0.1-1.el7
```

At the time of writing, the next, “O” (Ocata) release is due to February 24, 2017.

Distribution of services

- All-in-one
  - no scale out, for test only
- Distribution to several nodes
  - REST API services scale out, but one API endpoint is needed
    use load-balancers such as HAProxy
  - Non-API components offer built-in support of scaling
    - (neutron|cinder|nova)-scheduler
      availability test, load distribution
    - swift proxy
  - Certain components do not scale out naturally
    - DB: Mysql needs to be replicated/clustered (Mysql fabric, Galera)
    - MQ: RabbitMQ can have cluster support, or use self defined failover cluster (corosync/cman, RHCS)
    - Keystone: both keystone and back-end services (LDAP) have to be clustered/replicated

The next key question is where to install the previously mentioned OpenStack components.

Most automated installers (such as packstack provide an “All-In-One” installation method, so all software components of OpenStack are installed on the same node. That is nice for playing with OpenStack, but it does not scale at all, so you have to distribute the service components to more than one node.

Most communication in OpenStack can be sorted into two categories:

- The client sends REST calls to the service endpoint.
- The client sends a message via Advanced Message Queue Protocol, and jobs are processed from the queue by the service.

Most components have a process acting as a REST API endpoint (processes like `nova-api`, `glance-api`, etc.). All of these processes are expected to run on the **controller** node. If there are a lot of clients, the load increases on the single API process. The solve this issue, one can setup more nodes with other API endpoint processes running on them, and distribute client requests using a load balancer(such as HAProxy).

The non-API components (such as `nova-scheduler`), receive requests via the message queue (AMQ). In this case the location of the component is irrelevant and jobs can be processed by several processes (producer-consumer model). This provides a good horizontal scale-out even without requiring a load balancer.

There are however “single-instance” components as well

- The message queue service.
  - One can use RabbitMQ , Qpid,or ZeroMQ. Of these, only RabbitMQ provides a natural clustering/scaling feature.
- The database service
  - Usually MySQL is used. Along with the Galera replication it can scale-out to many nodes, by also providing high availability.
- The Identity service
  - Scale out depends on the actual back-end used. If using Galera for both authentication and token validation backend, Keystone scales out to many nodes as well.

