OpenStack

Clusters, Grids, Clouds course @ ITMO

Amir RAHAFROUZ

Report of Presentation

Prof. Andrey Shevel - June 5, 2018

Virtualization

Virtualization is creation of virtual --rather than actual-- version of Something, such as OS, Storage, Network Resource. In hardware virtualization, the host machine is the actual machine on which the virtualization takes place, and the guest machine is the virtual machine.

The words host and guest are used to distinguish the software that runs on the physical machine from the software that runs on the virtual machine. The software or firmware that creates a virtual machine on the host hardware is called a hypervisor or Virtual Machine Manager. Citrix, VMWare the most famous companies today for the virtualization technology.

Virtualization is the underlying technology behind the cloud which enables portability of Operating Systems to be managed by cloud controllers.

Scaling UP

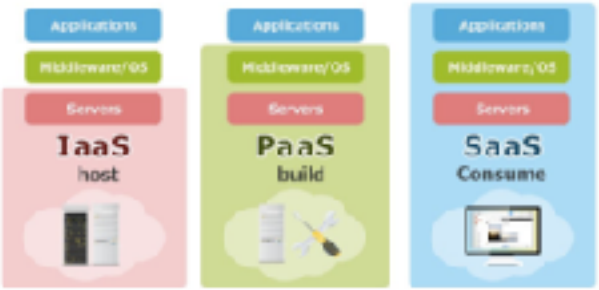
Due to the increasing demand for computation every day, novel method of computation would emerge each day. The first attempts were conducting supercomputers and try to make faster hardware parts such as CPU parts. However, due to technological barriers and complexities, making a single powerful super-computer is not the solution nowadays. The action of adding more resource to a single node is scaling vertically. In contrast, scaling horizontally is the trend nowadays. That is to say, by having a lot of commodity servers and increase the number of computing entities. Current models of clouds and clusters are build around the idea of vertical scaling.

Cloud Computing

The goal of cloud computing is to allow users to take benefit from all of these technologies, without the need for deep knowledge about or expertise with each one of them. The cloud aims to cut costs, and helps the users focus on their core business instead of being impeded by IT obstacles.

In cloud resources are shared and everything is provided as a service to end user. Resources would be available on demand in an automated manner.

There are different abstraction layers for cloud computing. Infrastructure can be provided as a service, so that users can use hardware, storage, and network facilities without prior in depth knowledge of technical specifications. Openstack falls into the category of IaaS and is the most famous software tool for this aim.



Main objectives of Cloud

Users used to use network, hardware, and storage devices before existence of cloud. The benefit that is provided by cloud is ease of access. In order to measure this ease several criteria are used as a yardstick to know the final direction of cloud platforms.

A cloud platform should support **massive scale** of nodes. It should scale up easily without any extra configuration which was conventionally done at the data-centers. A cloud platform also should be **agile.** Migration of Virtual Machines must happen on the fly. In cloud there should not be any predefined configuration to map the running VM to the hardware. Users should be able to move their virtual machine from one hardware to another place with ease. It can be said that by hiding technical difficulties from user, a level of **abstraction** is added to the data center systems. In addition, work at the cloud should be **automated** which implies that regular repetitive jobs of sysadmins are going to be removed. Automation of distributing configuration to the machines and network equipments would be configured automatically . For the users, there should be an **API** to use to develop normal applications without diving into all of the technical details of the computing infrastructure. Since the computational resources are shared, presence of a **metering** platform seems necessary. All of the resources used at cloud can be charged based on the usage amount, and users can the amount they consume. That is because if some resource of a server is available, it can be shared within other users, however conventionally, using idle resources was not that easy.

History of Openstack

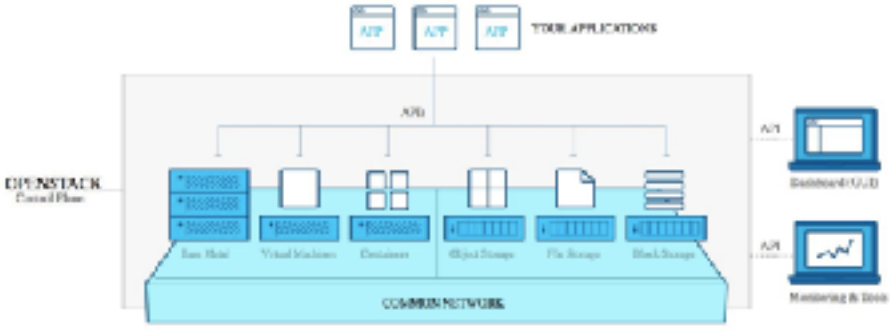
It begun in 2010 as a joint project of Rackspace hosting and NASA to build Cloud based OS. It is a result of a Merge of Swift (Object Storage) of Rackspace with Nebula (Compute platform) of NASA.

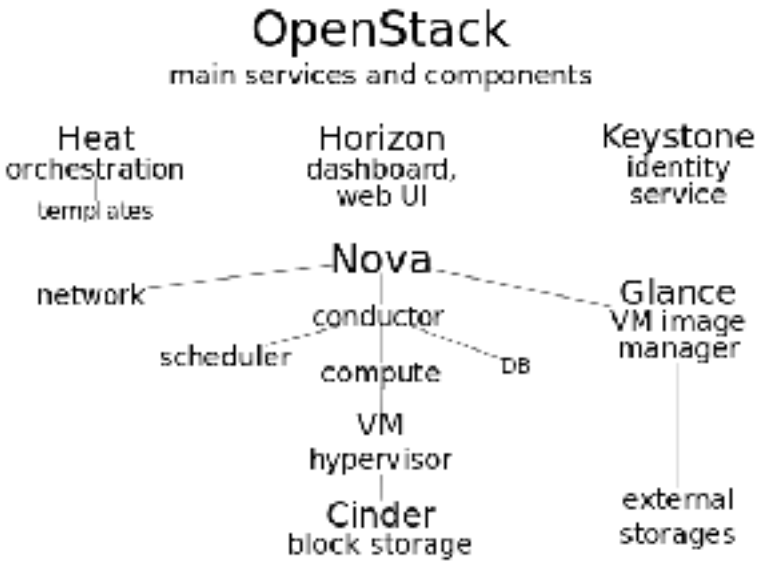
The OpenStack project intended to help organizations offer cloud-computing services running on standard hardware. The community's first official release, code-named Austin, appeared three months later on 21 October 2010, with plans to release regular updates of the software every few months. The early code came from NASA's Nebula platform as well as from Rackspace's Cloud Files platform. The original cloud architecture was designed by the NASA Ames Web Manager, Megan A. Eskey, and was a 2009 open source architecture called OpenNASA v2.0. The cloud stack and open stack modules were merged and released as open source by the NASA Nebula team in concert with Rackspace. Right now the Openstack has a new release every 6 months.

It is licensed under Apache license, so everyone can use it for free. Thank to the huge community behind it, Openstack receives a lot of contribution from developers all around the world and also more than 500 companies that contribute to the project such as: IBM, RedHat, HP, Cisco, Intel, Google, Oracle, Dell, EMC, VMWare, ... Openstack has got more than $10million in funding.

What is Openstack?

In short words: Openstack is a cloud Operating system that controls large pools of compute, storage, networking resources throughout a datacenter. Everything is managed by a dashboard (Horizon) and can be managed through Rest API calls. All services authenticate through a common source. OpenStack has a modular architecture with various code names for its components. In the following pages a brief introduction for the most important modules is brought.





Compute (Nova)

The core: Providing Virtual machines on demand. It schedules virtual machines to run on a set of nodes by defining drivers that interact with underlying virtualization mechanisms. It interacts with identity service to authenticate instance.

Nova is comprised of multiple server processes, each performing different functions. The user-facing interface is a REST API, while internally Nova components communicate via an RPC message passing mechanism. The API servers process REST requests, which typically involve database reads/writes, optionally sending RPC messages to other Nova services, and generating responses to the REST calls. RPC messaging is done via the **oslo.messaging** library, an abstraction on top of message queues. Most of the major nova components can be run on multiple servers, and have a manager that is listening for RPC messages. The one major exception is nova-compute, where a single process runs on the hypervisor it is managing (except when using the VMware or Ironic drivers). The manager also, optionally, has periodic tasks. Nova also uses a central database that is (logically) shared between all components. However, to aid upgrade, the DB is accessed through an object layer that ensures an upgraded control plane can still communicate with a nova-compute running the previous release. To make this possible nova-compute proxies DB requests over RPC to a central manager called nova-conductor. To horizontally expand Nova deployments, they have a deployment sharding concept called cells.

Networking (Neutron)

This component manages the networking infrastructure totally. It enables

- VLANs
- DHCP
- Floating IP, Load Balancing
- SDN
- OpenFlow
- IDS, VPN, Firewall

Each OpenStack cloud includes its own Nova, Cinder and Neutron, the Neutron servers in these OpenStack clouds are called local Neuron servers, all these local Neutron servers will be configured with the Tricircle Local Neutron Plugin. A separate Neutron server will be installed and run standalone as the coordinator of networking automation across local Neutron servers, this Neutron server will be configured with the Tricircle Central Neutron Plugin, and is called central Neutron server.

Storage: Cinder

Cinder is an OpenStack project to provide “block storage as a service”. The block storage system manages the creation, attaching and detaching of the block devices to servers.

- **Component based architecture**: Quickly add new behaviors
- **Highly available**: Scale to very serious workloads
- **Fault-Tolerant**: Isolated processes avoid cascading failures
- **Recoverable**: Failures should be easy to diagnose, debug, and rectify
- **Open Standards**: Be a reference implementation for a community-driven api

Storage: Swift

Swift is a highly available, distributed, eventually consistent object/blob store. Organizations can use Swift to store lots of data efficiently, safely, and cheaply. Its architecture is comprised of different parts:

Proxy Server

The Proxy Server is responsible for tying together the rest of the Swift architecture. For each request, it will look up the location of the account, container, or object in the ring (see below) and route the request accordingly. For Erasure Code type policies, the Proxy Server is

also responsible for encoding and decoding object data. The public API is also exposed through the Proxy Server.

A large number of failures are also handled in the Proxy Server. For example, if a server is unavailable for an object PUT, it will ask the ring for a handoff server and route there instead.

When objects are streamed to or from an object server, they are streamed directly through the proxy server to or from the user – the proxy server does not spool them.

The Ring

A ring represents a mapping between the names of entities stored on disk and their physical location. There are separate rings for accounts, containers, and one object ring per storage policy. When other components need to perform any operation on an object, container, or account, they need to interact with the appropriate ring to determine its location in the cluster.

The Ring maintains this mapping using zones, devices, partitions, and replicas. Each partition in the ring is replicated, by default, 3 times across the cluster, and the locations for a partition are stored in the mapping maintained by the ring. The ring is also responsible for determining which devices are used for handoff in failure scenarios.

The replicas of each partition will be isolated onto as many distinct regions, zones, servers and devices as the capacity of these failure domains allow. If there are less failure domains at a given tier than replicas of the partition assigned within a tier (e.g. a 3 replica cluster with 2 servers), or the available capacity across the failure domains within a tier are not well balanced it will not be possible to achieve both even capacity distribution (balance) as well as complete isolation of replicas across failure domains (dispersion). When this occurs the ring management tools will display a warning so that the operator can evaluate the cluster topology.

Data is evenly distributed across the capacity available in the cluster as described by the devices weight. Weights can be used to balance the distribution of partitions on drives across the cluster. This can be useful, for example, when different sized drives are used in a cluster. Device weights can also be used when adding or removing capacity or failure domains to control how many partitions are reassigned during a rebalance to be moved as soon as replication bandwidth allows.

Storage Policies

Storage Policies provide a way for object storage providers to differentiate service levels, features and behaviors of a Swift deployment. Each Storage Policy configured in Swift

is exposed to the client via an abstract name. Each device in the system is assigned to one or more Storage Policies. This is accomplished through the use of multiple object rings, where each Storage Policy has an independent object ring, which may include a subset of hardware implementing a particular differentiation.

Object Server

The Object Server is a very simple blob storage server that can store, retrieve and delete objects stored on local devices. Objects are stored as binary files on the filesystem with metadata stored in the file's extended attributes (xattrs). This requires that the underlying filesystem choice for object servers support xattrs on files. Some filesystems, like ext3, have xattrs turned off by default.

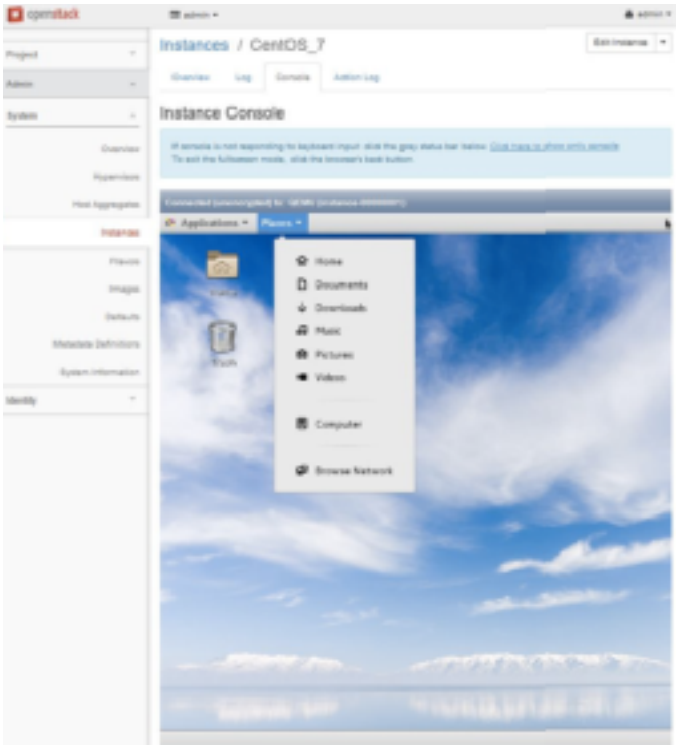
There are however more parts of the swift which bringing all of the info is more than the scope of this note.

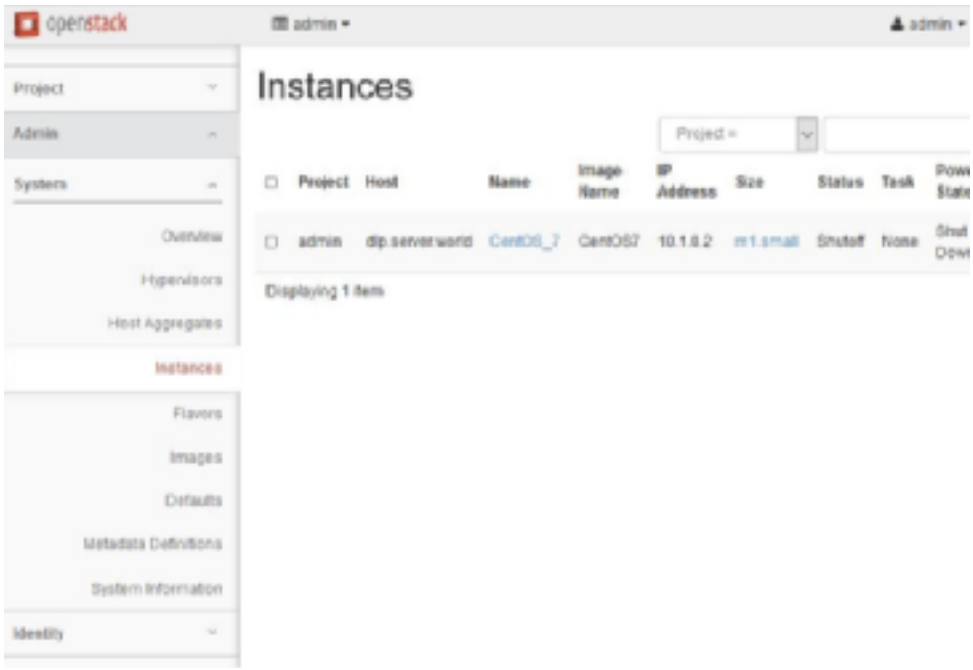
dashboard (Horizon)

It is the main management point of the architecture. Horizon started life as a single app to manage OpenStack's compute project. As such, all it needed was a set of views, templates, and API calls. From there it grew to support multiple OpenStack projects and APIs gradually, arranged rigidly into "dash" and "syspanel" groupings.

Horizon holds several key values at the core of its design and architecture:

- Core Support: Out-of-the-box support for all core OpenStack projects.
- Extensible: Anyone can add a new component as a "first-class citizen".
- Manageable: The core codebase should be simple and easy-to-navigate.
- Consistent: Visual and interaction paradigms are maintained throughout.
- Stable: A reliable API with an emphasis on backwards-compatibility.
- Usable: Providing an awesome interface that people want to use.





Telemetry (Ceilometer)

The Ceilometer project is a data collection service that provides the ability to normalize and transform data across all current OpenStack core components with work underway to support future OpenStack components.

Ceilometer is a component of the Telemetry project. Its data can be used to provide customer billing, resource tracking, and alarming capabilities across all OpenStack core components.

The demo was presented during the presentation time.