Document Revision History

Date	Author	Changes	Notes
27.6.2016	Alex Barchiesi Alberto Colla	Prima Bozza	
28.6.2016	Fulvio Galeazzi	Monitoring	
30.6.2016	Mario Reale	Aggiunta sezione Autenticazione ed Autorizzazione	
1.7.2016	Mario Reale	Aggiunti requisiti da Paolo Velati	Da integrare nel documento
5.7.2016	Alex Barchiesi Alberto Colla Mario Reale	Completati diagrammi	
8.7.2016	Simone Visconti Davide Vaghetti	Ansible Review AAI	
11.7.2016	Fabio Farina	Revisione, commenti e integrazione	
12.7.2016	Alex Barchiesi Alberto Colla	Revisione	
20.7.2016	Alex Barchiesi	Chiusura doc	
21.7.2016	Beppe Attardi	Major restructuring	
23.7.2016	Beppe Attardi	Revisione architettura e aggiunto rationale	
25.7.2016	Alex Barchiesi Alberto Colla	Minor revisions	
18.9.2016	Beppe Attardi	Tradotte parti in inglese	
21.9.2016	Beppe Attardi	Section on public clouds	
29.9.2016	Beppe Attardi	Handling of quotas in domains	
2.10.2016	Beppe Attardi	Added sections on Operations and User Guide	
30.11.2016	Beppe Attardi	Spun off GARR-X section to separate document	
20.2.2017	Alex Barchiesi	Revision Figures/Schemes	
6.4.2017	Beppe Attardi	Change Prometheus in Figure.	

11.04.2017	Fulvio Galeazzi	Revised Ceph description	
17/5/2017	Beppe Attardi	Added Gnocchi and Ceilometer	

Reference Architecture for the GARR Federated Cloud

High Level Design

Table of Contents

1 Introduction

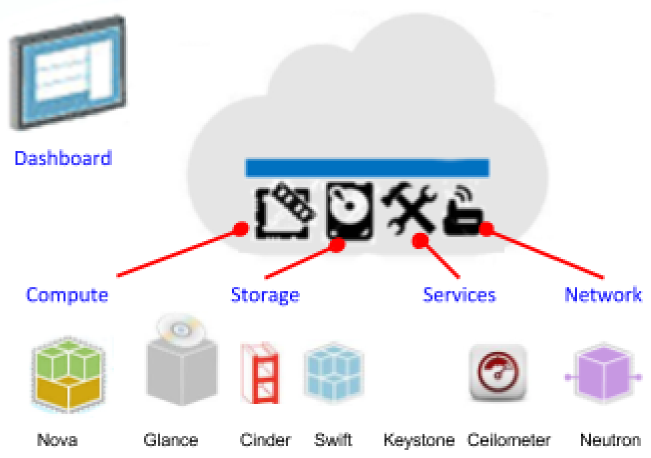
Deploying a cloud infrastructure allows sharing resources through virtualization, simplifying the provisioning of storage and computing services, and providing cloud-based services.

This document presents the high level reference architecture for a federated cloud over multiple regions, possibly managed by different organizations, that complies with the requirements set forth in the document “Requisiti per la Cloud GARR” [1].

The architecture is based on the open-source OpenStack cloud technology, which is currently the de facto standard. As it is typical with open-source solutions, the solution offers reduced investment and operational costs and scalability without licensing limitations. However, building a cloud remains a complex task, which requires the integration of several disparate components. To simplify the process, reduce the required man-power, reduce risk and keep services always up to date, an effort must be placed on using tools for automating the steps needed to create, implement and support a production-ready OpenStack cloud. Below we outline an automation solution based on Juju [8].

2 Cloud Platform: OpenStack

OpenStack is a software platform for cloud computing, that consists in several interconnected components which enable controlling groups of hardware processing, storage and networking resources within datacenters. The platform can be managed either through a web-based dashboard, or through a CLI, or through a RESTful API.



2.1 OpenStack Components

We list here the OpenStack components that are used.

Service	Project name	Description
Compute	Nova	Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decommissioning of virtual machines on demand.

Networking	Neutron	Enables Network-Connectivity-as-a-Service for other OpenStack services, such as OpenStack Compute. Provides an API for users to define networks and the attachments into them. Has a pluggable architecture that supports many popular networking vendors and technologies.
Storage		
Object Storage	Swift	Stores and retrieves arbitrary unstructured data objects via a *RESTful*, HTTP based API. It is highly fault tolerant with its data replication and scale-out architecture. Its implementation is not like a file server with mountable directories. In this case, it writes objects and files to multiple drives, ensuring the data is replicated across a server cluster.
Block Storage	Cinder	Provides persistent block storage to running instances. Its pluggable driver architecture facilitates the creation and management of block storage devices.
Storage Clusters	Ceph	Provides high-performance scalable block storage with advanced features, replication, erasure coding, snapshotting, storage tiering
Shared Services		
Identity service	Keystone	Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.
Image service	Glance	Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.
Telemetry	Ceilometer	Monitors and meters the OpenStack cloud for billing, benchmarking, scalability, and statistical purposes.
Management Services		
Hardware Provisioning	MAAS	Help facilitate and automate the deployment and dynamic provisioning of hyperscale computing environments.
Service Modeling	Juju	Service modeling, configuration and orchestration.
Orchestration	Heat	Orchestrates multiple composite cloud applications by using either the native *HOT* template format or the AWS CloudFormation template format, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.
Container Orchestration	Kubernetes	Kubernetes is an open-source system for automating deployment, scaling, and management of containerized applications.
Dashboard	Horizon	Provides a web-based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses and configuring access controls.

Application Deployment		
Marketplace	Murano	Supports easy deployment of Openstack ready-to-go Heat recipies and Docker-based PaaS
Data Analytics	Sahara	To provision from the OpenStack GUI ready-to-go Hadoop distributions (HDP, CDH, Spark, Storm, Hadoop) with native up/down scalability and object storage integration.

Once OpenStack has been selected as cloud platform technology, a number of consequences arise for the basic software to use:

- Linux Operating System (Ubuntu or CentOS) for server machines (virtual servers may instead use different OSs)
- Linux Containers for *internal virtualization*: i.e. for the services running on the physical servers)

3 Service Architecture

The logical service architecture is organized into the following layers, shown in Figure 1:

1. Physical: consists of the physical hardware and network resources
2. Operating System: corresponds to the services provided by the operating system
3. Virtualization: provides an abstraction of OS services through virtualization
4. Application Services: services provided by the platform

The services can be provided either directly by containers or on VM provisioned by OpenStack.

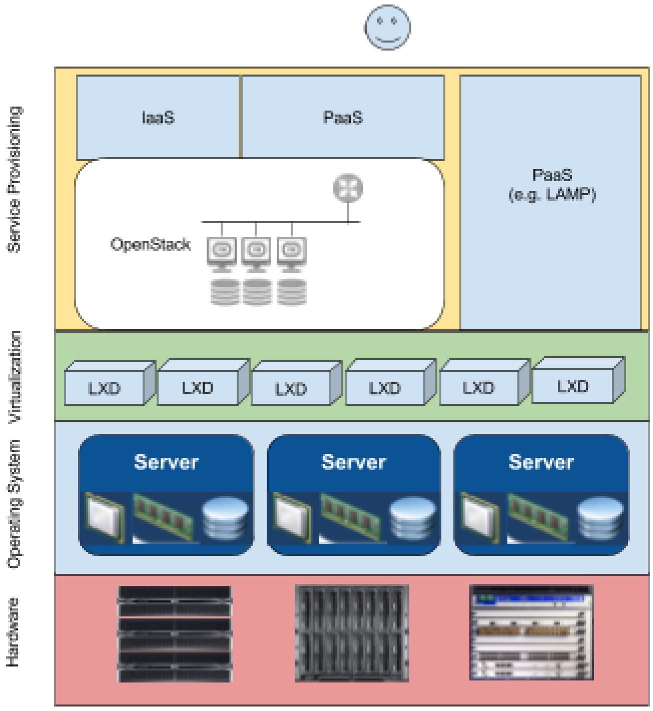


Figure 1. Logical Services Architecture.

The first level of virtualization is based on Linux Containers (LXD), VM or Bare Metal[1]. The OpenStack modules and auxiliary components (database, HA proxy, etc.) are hosted on LinuX Containers placed on physical servers (while staying independent) in order to create a highly reliable and load balanced infrastructure. For example, as the need arises containers can be moved from one server to another.
The services by the automation tools (MaaS controller, Juju bootstrap node, DHCP) are hosted on LXD containers as well.
All servers within a region, besides those assigned to provide backplane or control plane services, will be typically dedicated to provide virtual machines for the projects.

4 Federated Architecture

OpenStack does not have native facilities for building federations among different Cloud Service Providers. Only a blueprint [13] currently exists for exploiting Availability Zones in a federated cloud, but the project has not yet started.

When it will become available, a more complex and more scalable solution might be the one proposed by Tricircle [12], which offers a OpenStack API gateway and networking automation to allow managing several instances of OpenStack as a single cloud, scattered on one or more sites or in a hybrid cloud.

The solution proposed here is based on a multi-region OpenStack model, which is practically feasible assuming to federate cloud designed and implemented in a coordinated manner, based on compatible OpenStack platforms.

The solution leverages the OpenStack mechanisms to scale to thousands of nodes and to expand onto different data centers and geographical areas, exploiting in particular the concepts of Region and Availability Zone.

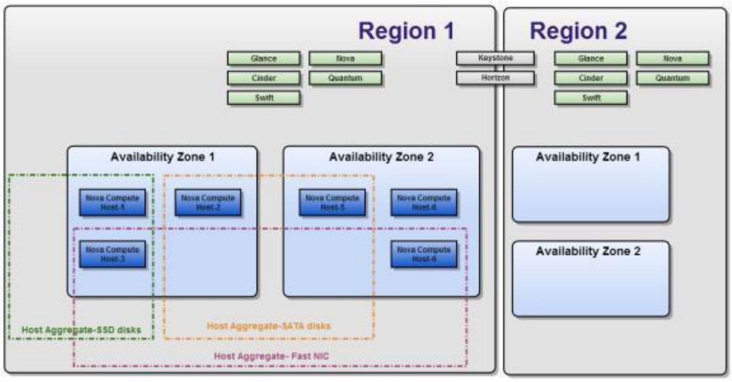


Figure 2. Example of Availability Zones in a multi-region architecture.

Region

[1] Linux Containers are preferable to traditional virtualization platforms (VM) since their execution *overhead* is minimal, while they offer the portability of virtualization. On the other hand LXD may provide a lower degree of isolation with respect to the host machine, but this has no impact on security in our case, since the hardware and LXD level are only accessible for system administration.

Each Region has its own deployment of OpenStack, including endpoint API, network and computing resources, which is linked to other regions using shared centralized services such as OpenStack Identity and dashboard.

The concept of regions is flexible; it may contain OpenStack service endpoints located within a distinct geographic region or regions. It may be smaller in scope, where a region is a single rack within a data center, with multiple regions existing in adjacent racks in the same data center.

Availability Zone

Within each Region, nodes can be logically grouped into Availability Zones (AZ): when a VM is provisioned, one can specify in which AZ the instance should be started, or even within which specific node inside an AZ.

Host Aggregate

Within a Region machines can be grouped into Host aggregates. A machine may belong to multiple Host aggregates.

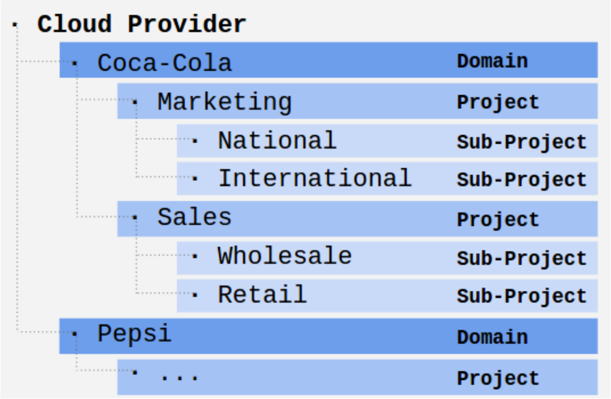
4.1 Segregation and Sharing

The logic of the federation allows each organization to maintain control over the use of its resources, which part to keep for private use and which part to give for sharing with the federation. Technically this is done by associating to each region its own Keystone Domain, through which the local manager will administers its resources.

4.1.1 Domains and Hierarchical Projects

Projects can be arranged in a hierarchy, whose root is a domain. Once you have a project hierarchy created in keystone, nested quotas let you define how much of a project's quota to give to its subprojects. In that way, hierarchical projects can have hierarchical quotas (also known as nested quotas). This feature is used to provide hierarchical multitenancy in the federation.

For example, one can have the situation shown in Figure 3, where two separate domains are assigned to different organization. Each organization is organized internally into projects and subprojects. Subprojects inherit role assignments, so for example the administrator for project Marketing is also an administrator for sub-projects National and International. Through nested quotas it is possible to control how the compute and storage resource can be divided among the two subprojects.



```
· Cloud Provider
    · Coca-Cola                      Domain
        · Marketing                  Project
            · National               Sub-Project
            · International          Sub-Project
        · Sales                      Project
            · Wholesale              Sub-Project
            · Retail                 Sub-Project
    · Pepsi                          Domain
        · ...                        Project
```

Figure 3. Hierarchical projects.

4.2 Central Authentication

The authentication and authorization services, however, are provided by a single central service that exposes a single API and a single web interface for the whole federated cloud. Since authentication itself is performed in a federated form, users are globally recognized and therefore can be enabled to use resources on the whole federation. For efficiency of access, the Keystone database will be replicated in slave mode in each region. The authentication system allows the use of credentials provided by Identity providers in the IDEM federation as well as OpenID Connect providers.

4.3 Architecture Overview

The main features of the federated architecture are:

- Replicability and scalability;
- Integration of resources managed by different organizations;
- High level of automation in deployment and maintenance;
- Elastic allocation of resources assigned to a service, exploiting resources shared by other regions;
- Unified handling of authentication and authorization to access the federation;
- High degree of reliability, ensuring continuity in the services even in case of failures of resources within a region and enabling procedures of disaster recovery in case of unavailability of a whole region;
- Flexible security policies both at the resource and user levels.

The general architecture of the federation is shown in Figure 4. The architecture includes a Master Region, that hosts all components required for managing the federation and the tools for automated installation of the platform on dedicated resources, as well as a number of Regions that host the OpenStack application services.

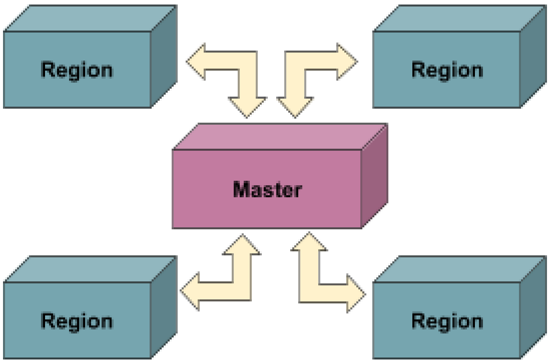


Figure 4. Architecture of the multi-region federated cloud.

The required services present in the Master Region are:

- Compute (Nova)
- Networking (Neutron)
- Block Storage (Cinder)
- Image service (Glance)
- Object Store (Swift)
- Metering (Ceilometer)
- Authentication/Authorization (Keystone)

- Server management (MaaS)
- Service orchestration (Juju)
- Messaging (RabbitMQ)
- Dashboard (Horizon)

In particular the service of federated authentication is present only in the Master Region.

Services present in each Federated Region include:

- Compute (Nova)
- Networking (Neutron)
- Block storage (Cinder)
- Image Service (Glance)
- Object Store (Swift)
- Messaging (RabbitMQ)
- Metering (Ceilometer)
- Server management (MaaS)
- Service orchestration (Juju)

Notice that there is no separate Keystone in the regions.

These components are suitably replicated on each region for redundancy and reliability, as described in the section on High Availability.

4.4 Public and Private Clouds

The federation can seamlessly use a number of public clouds (including Amazon Web Services, Azure, Google Compute Engine, Joyent and Rackspace) to deploy workloads, as well as private clouds (e.g. OpenStack) which you configure. This is achieved by means of the cloud facility in Juju.

4.5 Architecture Design Criteria

In designing the reference architecture we applied the following criteria.

4.5.1 Cloud controller

The cloud controller provides the central management system for multi-node OpenStack deployments. Typically the cloud controller manages authentication and sends messages to all the systems through a message queue. For our example, the cloud controller has a collection of nova-* components that represent the global state of the cloud, talk to services such as authentication, maintain information about the cloud in a database, communicate with all compute nodes and storage workers through a queue, and provide API access. Each service running on a designated cloud controller may be broken out into separate nodes for scalability or availability. It is also possible to use virtual machines for all or some of the services that the cloud controller manages, such as the message queuing.

In this reference architecture, we used three cloud controller servers to host the OpenStack management services.

4.5.2 Database

Most OpenStack Compute central services, and also the Nova compute nodes, use the database for stateful information. Loss of database availability leads to errors. In our deployment the database is arranged in a High Availability cluster to make it failure tolerant.

4.5.3 Message queue

Most OpenStack Compute services communicate with each other using the Message Queue. In general, if the message queue fails or becomes inaccessible, the cluster grinds to a halt and ends up in a "read only" state, with information stuck at the point where the last message was sent. Accordingly, the message queue is clustered, exploiting RabbitMQ built-in abilities.

4.5.4 Images

The OpenStack Image Catalog and Delivery service consists of two parts: glance-api and glance-registry. The former is responsible for the delivery of images and the compute node uses it to download images from the back-end. The latter maintains the metadata information associated with virtual machine images and requires a database.

The glance-api is an abstraction layer that allows a choice of back-end used when providing storage for deployment images.

4.5.5 Dashboard

The OpenStack Dashboard is implemented as a web application accessed through a web browser. Because it uses the service API's for the other OpenStack components it must also be able to reach the API servers (including their admin endpoints) over the network.

4.5.6 Authentication and authorization

OpenStack authentication and authorization rely on the concepts of *users*, *projects* and *roles*. Users are authenticated by submitting their credentials, and they can be assigned roles in one or more projects (previously called tenants), with access subject to *role based control* (RBAC). *Domains* allow defining administrative boundaries for managing resources. Authentication can be delegated to trusted Identity Providers, including those in the IDEM federation and those based on OpenID Connect.

4.5.7 Network requirements

Because the cloud controller handles so many different services, it must be able to handle the amount of traffic that hits it. This reference architecture specifies 10GbE NICs for all network connections between the server and the network switch. Each server must have at least two NICs.

4.5.8 MAAS and Juju

MAAS (Metal As A Service) is a tool that helps managing physical infrastructure with the same ease and flexibility as virtual machines in the cloud. Specially, MAAS allows one to:

- Discover, commission and deploy physical servers
- Dynamically allocate physical resources to match workload requirements.
- Retire servers when they are no longer needed and make them available for new workloads as required.

MAAS is responsible for hardware discovery and for installing the OS, making basic hardware configurations and allowing servers to be recognized by network and system management software. When a new node boots up, MAAS steps in, supplies it with all the required information and provides an OS image install. This is done via PXE and DHCP. MAAS provides both a web interface and an API to manage bare metal systems under its control.

MAAS works in conjunction with Juju. Juju is a service orchestration tool which allows the administrator to configure, manage, maintain, deploy and scale cloud services (workloads) quickly and efficiently on public clouds as well as on bare metal, leveraging MAAS to control the hardware. Juju uses descriptions of services called Charms which understand how to deploy and scale a variety of complex architectures like OpenStack. Juju can be controlled via a web GUI, the command line, or API.

4.6 OpenStack Reference Architecture

The architecture design allows for a medium- to large-size cloud installation that scales well. Adding additional compute capacity is as simple as adding additional compute nodes.

4.6.1 Logical model

For scalability and resilience the critical components are replicated a number N of copies, with N > 2, placed in distinct Availability Zones.

This reference architecture is implemented around the following logical model:

- The controller nodes run the Identity Service, Image Service, Networking Service, dashboard, databases, messaging system and the management portion of Compute, Block Storage, and Object Storage services.
- The Neutron networking service is deployed on N dedicated networking nodes.
- Provisioning is implemented by a node that runs a cluster of two MaaS Region controllers and a MaaS Cluster controller (one for each region)
- The compute nodes run the hypervisor portion of Compute, which operates project virtual machines. By default, Compute uses KVM as the hypervisor. Compute also provisions and operates project networks and implements security groups.
- Object Storage is provided by an N node storage cluster.
- The Object Storage configuration supports both object and image storage. Glance images are stored in the object store.
- Block storage is allocated on N separate storage nodes.

Both block and object storage can be scaled by adding additional storage nodes and/or storage volumes.

The logical architecture and the network topology is presented in Figure 5.

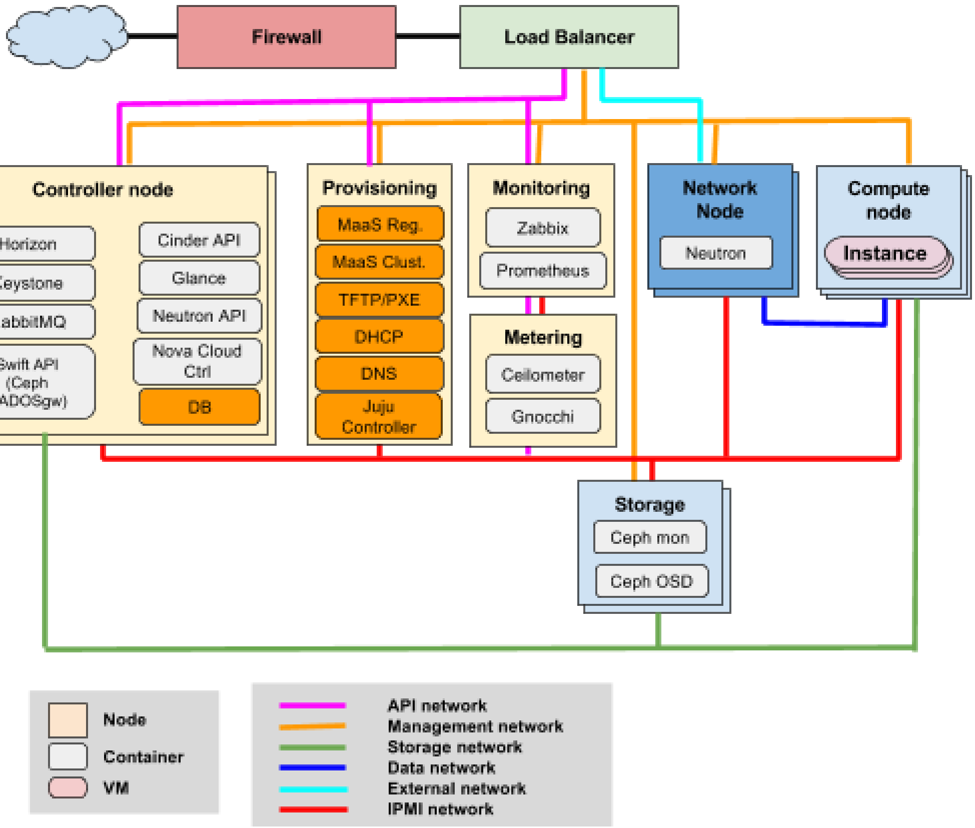


Figure 5. Reference Architecture for OpenStack.

Since deployment of components is assigned to Juju, as described later, the grouping of components shown in the figure is purely logical and does not correspond to a specific placement onto physical nodes (except for storage nodes). The number of replicas of each component is chosen for redundancy and load balancing and are assigned arbitrarily to different nodes. In other words, where the figure shows multiple nodes, this means that the components in those nodes are replicated on different nodes as many times as the nodes. The replicas in **amber** color are configured for High-Availability.

4.7 Networking

The reference architecture requires the availability of at least two separate physical networks:

1. One net for IPMI (network for controlling the physical devices);
2. One net for data traffic.

The following logical networks, isolated at level 2, are overlaid over the data networks for handling different classes of traffic:

1. Management network: private network for communication among the core OpenStack services. This net conveys the service traffic of OpenStack (messaging between modules, database access, etc.);
2. Storage access network: private network used for traffic among storage nodes and compute nodes;
3. API network: public frontend network for the control plane services of the cloud (access,

dashboard, etc);

4. Data network: private network managed by the OpenStack networking service in order to isolate the traffic of different user groups (OpenStack projects);
5. External network: public network managed by the Openstack networking service providing external access to the virtual instances.

The networking service in OpenStack (Neutron) provides outward connectivity to virtual resources. The separation of traffic between the various projects is achieved by means of an OpenFlow controller: within the physical hosts, the gateways of private project networks are connected to virtual Neutron routers, kept within separate VLANs and segregated into separate *netspaces*; the traffic between physical hosts is kept separate by means of the VXLAN/GRE overlay networking protocol.

This mechanism, combined with a configuration that allows interconnecting the internal OpenStack networks, allows communication between instances of projects within different federation regions, i.e. this allows associating a non local compute node to the control panel of a region.

Nodes are externally reachable at *floating IP* addresses allocated in the address space of the external network (public project, of flat type) by a NAT router.

Networking nodes (Neutron) are deployed on dedicated physical servers. Redundancy of virtual networking is achieved through L3 agents in high availability managed by Neutron.

4.8 Storage

In a cloud environment three types of storage are used:

- Object storage, which treats data as “objects”, with associated metadata to describe them.
- Block storage, which organizes data as blocks of sectors and tracks, emulating a physical hard disk.
- File storage, which organizes data in a directory hierarchy.

4.8.1 Object Storage

An Object storage is basically a key/value store where the key is a positive integer: the ID of the object. They are suited for storing massive amounts of unstructured data, since they provide great scalability and rely on relatively inexpensive hardware.

In OpenStack, object storage is typically used for storing backups and virtual machine images (via Glance).

Swift is commonly used to provide object storage in OpenStack. But since our Reference Architecture contemplates the use of Ceph, which provides both block and object storage, Ceph is used for both types of storage, exposing both a Swift and S3 API through a Rados Gateway.

4.8.2 Block Storage

Block storage in OpenStack is provided by the Cinder service, which abstracts storage provided via a number of block storage backend, from LVM to Ceph.

4.8.3 Distributed File Storage

The current architectural design does not include the provisioning of distributed file storage. Smallish sites can rely on sound technologies like GlusterFS or NFS, with HA possibly being guaranteed via Pacemaker/Corosync. For larger installations, besides commercial solutions (like GPFS, Lustre, Quobyte), the most natural choices are OpenStack Manila and CephFS, once they will become mature enough.

4.8.4 Ceph

Ceph exploits inexpensive hardware (JBOD) to provide a high-performance scalable storage with advanced features, such as replication, erasure coding, snapshotting, storage tiering. Ceph can supply storage in the form of either object, block or file storage.

Ceph requires two sets of servers:

- monitor (MON) servers, which need to be in odd number (typically 3): these servers maintain, and serve to clients, global information about the configuration and health of the cluster. Clients reach MON nodes through routers, but data transfers occur directly between clients and OSD servers.
- OSD (Object-based Storage Device) servers, which are the nodes serving the disks. They are connected to the *Storage network*, as well as to a private high capacity network, where replication traffic flows. To achieve maximum performance, a JBOD configuration is preferable to using RAID.

One independent Ceph cluster is present in each region: we are currently at the Jewel release.

In our setup, each computing rack hosts both CPU resources and storage. Storage is provided by a FC-based SAN, which (lacking a global FC switch) can be accessed only by machines within the same computing rack. Moreover, each computing node is equipped with huge resources in terms of CPU and RAM. For all these reasons we decided to configure Ceph using “fat nodes”, namely each OSD serves a large number of disks. Within each rack, one physical server is configured as OSD, to which storage disks are presented. Ceph monitor (MON) nodes are installed as LXC containers on each OSD server (one MON per physical OSD server, 3 in total for a given site). In the near future, we will expand our Ceph installations along two directions:

- add functionalities, for example deploying CephFS (providing distributed filesystems)
- define, possibly on the same physical nodes, a secondary cluster, spanning all three regions

The relevant additional servers, where needed, will also be installed as containers on the OSD nodes.

Installation and configuration of OSD and MON nodes is performed using ceph-ansible: this makes routine operations, like adding more disks, very simple and error-proof. Disks are normally configured with the journal residing in a small partition on the same disk. As soon as the Bluestore technology will be declared production-ready, we will switch to it.

Ceph allocates bits of data according to so-called CRUSH maps. The default configuration for CRUSH map has been modified in two respects:

- we have a separate entry point (‘root bucket’, in Ceph terminology) for each kind of disk, currently three: SSD (root=ssd), single SAS disks (root=default), large disks (root=big, these are RAID0 spanning 2 SAS disks)
- we introduced a new bucket type, called ‘storage’, sitting right below the ‘host’ bucket, to reflect the fact that in our installation there are two independent storage systems within each rack. Each storage system is actually composed of two distinct enclosures, but we

decided not to go to such level of detail.

By managing the CRUSH maps we can optimally place Ceph pools. Normally we configure 'fast' pools on SSD disks with replica 2, 'regular' pools on HDD disks with replica 3 while 'erasure-coded' pools are backed by 'big' disks.

Erasure-coding is roughly a generalization of the RAID6 concept. Each data block is split into *k* data and *m* parity chunks, which are stored into separate disks: *k* chunks are enough to reconstruct the information, and the system can tolerate the loss of up to *m* disks. With respect to replica 3, erasure coding provides even higher levels of data availability at only a fraction of the cost. Our setup uses *4+2* erasure coding, which reduces the overhead of wasted space for replication to just 50%, rather than 200% for the 3-replica pools. Erasure-coding is used for object storage pools, needed by the Glance and Swift services. Erasure-coding could be used for block devices too, provided a replicated pool is used in front of it: however, the performance penalty for such a configuration is rather high and it was not deemed appropriate to our use case.

Each cluster has three physical OSD servers, in distinct racks: as such servers have 4 10 Gbit/s interfaces, network traffic for each of the public and private networks flows through distinct bondings of two 10 Gbit/s interfaces. Disks are picked from the two storage systems and presented individually to the server as RAID0 (this is the closest approximation to JBOD that the storage systems in use at GARR can achieve). Each OSD server has access to a limited quantity of disk space on SSD disks which is used for hosting Ceph journals and to build a high-performance pool on which to create volumes to be used as tiered storage or for extremely demanding applications. Monitor nodes are installed as an LXC container on each OSD server.

4.9 Users and Access Authorization

The mechanisms for authorizing access to resources are based on these concepts:

- A *Project* (formerly known as *tenant*) owns a set of resources (servers, storage, etc.).
- A *User* represents an individual entity and belongs to a single *Domain*.
- A *Group* represents a collection of users within a single domain. Users in a group inherit the roles assigned to the group.
- A *Role* includes a set of rights and privileges. A *User* assuming a role inherits its rights and privileges.
- *Domains* are a collection of projects, users, groups and roles[2] that define administrative boundaries for managing resources.

Users or Groups are given access to a Project or to a Domain by assigning them a Role on such Project or Domain. Roles determine the type of access and capabilities the User or Group is entitled to.

A *role assignment* is a triple: the combination of an *actor* (either user or group), a *project* (domains are a special case of projects) and a *role*. For example the role "admin" is *assigned to* the user "bob" and is *assigned on* the project "development".

Users can be associated with one or more projects by granting them roles on a project, including projects owned by other domains. A user may hence have a different *role* in different projects.

[2] Since Mitaka.

By default, there are two predefined roles: a *Member* role that gets attached to a project, and an *Admin* role to enable administering a domain.

The *Cloud administrator* can manage domains, projects, users, groups and roles of the whole cloud.

The administrator can add, update, and delete projects and users, assign users to projects, and change or remove the assignments. To enable or temporarily disable a project or user, he can update that project or user. He can also change quotas at the project level.

Users can be granted the administrator role for a domain. A *Domain administrator* can create projects, users, groups and roles in a domain and assign roles to users and groups in that domain.

The following Figure illustrates the various concepts. Roles are represented by arcs of different colors. User1 has the Admin role for the whole Domain A. User2 has access to both Proj1 and Proj 2. User3 belongs to Domain A, but it is granted the role of Member also on Proj4 in Domain B.

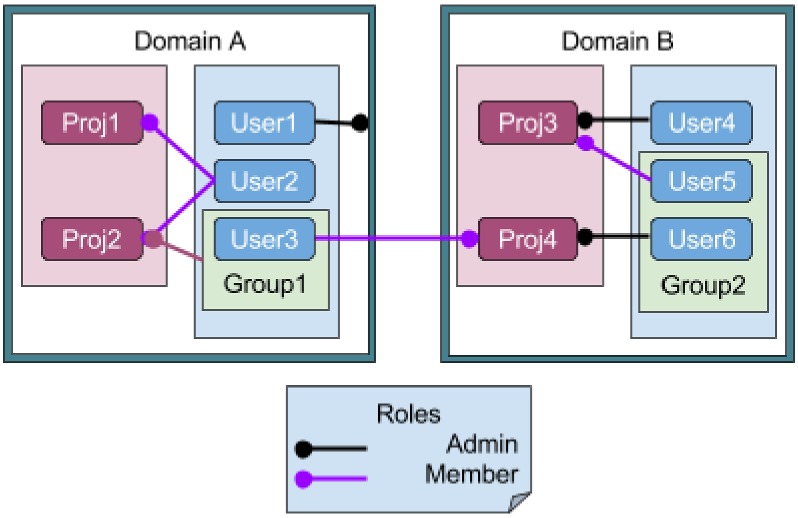


Figure 6. Users, roles, projects, domains.

In the Mitaka release of Keystone v3, projects and users are domain-specific, roles can optionally be domain-specific, while services and endpoints are Keystone-instance-wide.

Projects can be nested, inheriting resources from their parents and dividing them among siblings.

4.10 Authentication and Authorization

We designed the Authentication and Authorization mechanism in the federation so as to fulfill the following requirements:

1. Separation of roles, between the cloud administrator and the domain administrators. The cloud administrators create the domains and assign resources to them, the domain administrators are delegated to fully administer their domains, creating projects, users and groups and assigning roles to users and groups on projects.
2. The federated Identity providers are delegated only for authentication and they need just to provide sufficient properties in order to identify uniquely a user (typically through an email address)
3. No authorization information should be stored outside of keystone, in order to avoid:

    a. Having to check reliability and consistency of such information
    b. Having to map it to internal keystone entities
    c. Force users to act on an IdP not under their personal control
4. Users can be granted rights on any project of the federation, irrespective of their affiliation and under the sole control of the administrator for that project
5. Deploy the simplest solution, relying as much as possible on native OpenStack capabilities avoiding any extra non necessary component.

The following describes the chosen solution for federated authentication and authorization.

Users of the federated cloud must have an identity provided by a trusted *Identity Provider*.

Currently the accepted Identity providers are:

- IdPs participating to the IDEM [2] federation, which will typically be the IdP of the organization to which the user belongs (Home Organization).
- An OpenID Identity Provider (e.g. Google).

Enabling federated authentication entails:

1. Configuring OpenStack Keystone for Federation
2. Enabling the federation protocols, currently just the two major ones:
    a. SAML implemented by Shibboleth, see Setup Shibboleth;
    b. OpenID Connect, see Setup OpenID Connect.

The external Identity Provider is responsible for authenticating users, and communicates the result of authentication to keystone including their identity properties. Keystone maps these values to keystone user groups and assignments created in Keystone by each Domain Administrator.

The details of the interaction between the Federated Keystone and the actors in the identity federation are described in the following Figure.

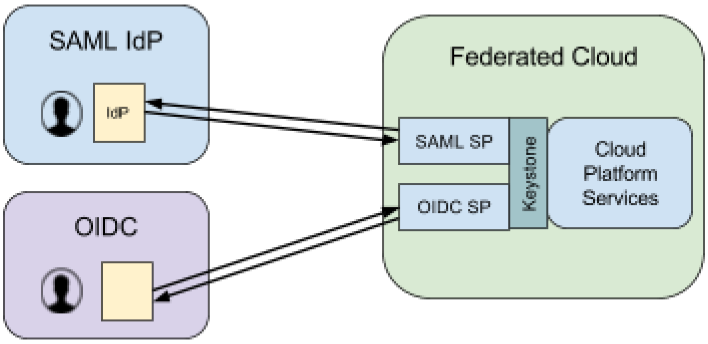


Figure 7. Federated Authentication.

4.11 Monitoring

The monitoring system is charged of providing exhaustive information about the performance of the whole federated cloud, in terms of resource allocation and consumption (CPU/disk/memory), per region, per computing node or per instance. Data from the monitoring system must be analyzed for detecting capacity usage and trends. The monitoring system must integrate easily with existing monitoring tools installed on the nodes, like Ceilometer and other data collectors.

Monitoring the operations of the cloud platform is performed at several levels, using the following tools.

4.11.1 Hardware Monitoring

- Monitoring the hardware infrastructure. Performed by means of Zabbix, an open-source monitoring tool, that provides both agent and agent-less data collection. Zabbix provides data storage, graphical visualization, alarm handling, email notifications and it allows defining customized metrics.

4.11.2 Troubleshooting

- Log collection and analysis. Performed by means of Monasca, a monitoring-as-a-service solution, and ELK (ElasticSearch, Logstash, Kibana), a platform for the collection, analysis and display of log data from various sources. Logs from all levels of the cloud can be analyzed, from the infrastructure level (network equipment logs) to operating system and application logs.

4.11.3 Software Monitoring

- Update management. Performed by means of Katello, which checks for the availability of upgrades for the packages installed on the servers, in particular those critical for solving security problems, and sends email notifications to the administrators when this occurs. Administrators can use its web interface to control the installation and removal of packages.
- Monitoring and alarm handling of containers that host the OpenStack services. Performed by means of Prometheus, a monitoring and alerting toolkit.

The monitoring and alarm system focuses on events that may prevent the proper operation of user resources and how malfunctions may impact other resources in the cloud.

4.11.4 Metering and Billing

Metering of resource consumption up to the user/project level is performed by means of Ceilometer [ref. http://docs.openstack.org/developer/ceilometer/architecture.html], according to the scheme presented in Figure 7.

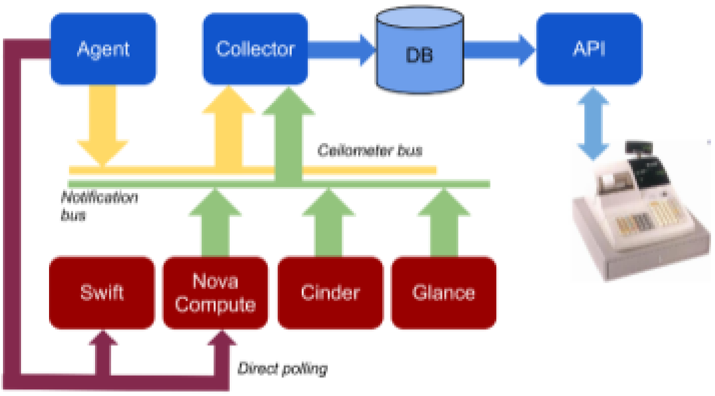


Figure 8. Scheme of Ceilometer operations from notifications to data publishing.

Ceilometer delivers a Single Point Of Contact for billing systems, providing all the counters they need to establish customer billing, across all OpenStack components.

The Aodh Alarming service is a Ceilometer sub-project that provides a more general alarm mechanism. AODH provides instant response times by specifying a separate event listener on the OpenStack message queue. This listener emits an alarm when a pre-defined event pattern occurs, an alarm which, in turn, enables immediate issue detection and possible response.

Charging the costs of resources used, it is split into three parts: metering, rating and billing. Billing is assigned to Ceilometer, configured to work in pipeline, streaming data to Gnocchi as a publisher, which aggregates them and stores them in a time series database and uses a database for indexing them and allowing queries on resources usage over time, over projects and over resources.

Rating could be performed by means of CloudKitty, which allows to set pricing for various types of resources, including volume or other type of discounts, through a plugin on the Horizon Dashboard.

4.12 High Availability

Within each region, for redundancy and reliability, the following nodes are replicated:

- 2 copies of the monitoring nodes
- 2 copies of the provisioning nodes, including two Juju bootstrap nodes
- 3 copies of OpenStack (?)

Come subentrano l'un l'altra?

The GARR Cloud setup is based on a logical HA model. Such a deployment uses a minimum of five nodes, and uses a variety of methods to ensure HA. This section covers the standard and recommended HA model.

In order to guarantee high availability and continuous operations, three instances of each service are deployed in the physical infrastructure. The following diagram shows the relations among the instances of services.

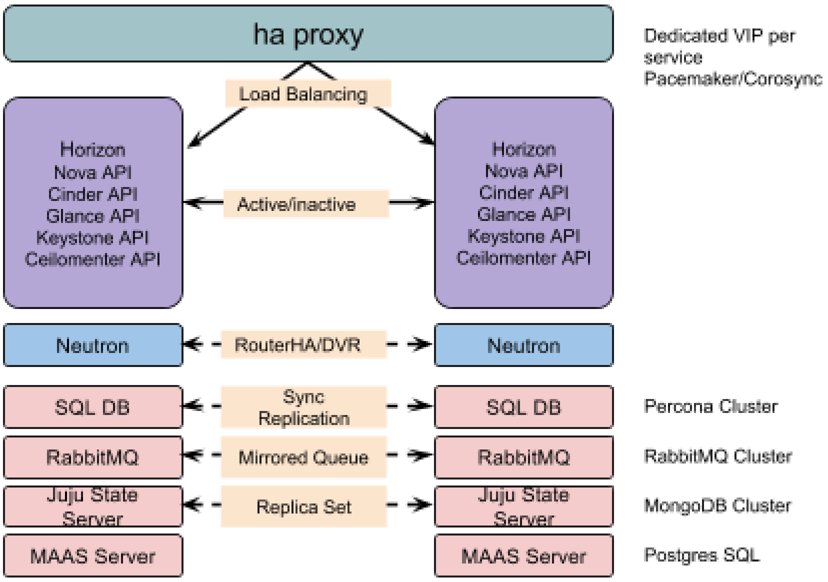


Figure 9. HA model.

Illustrare l’uso di DNS e proxy

Replicazione DB Juju

4.13 Security

- OpenStack services hosted on LXC: iptables on physical hosting nodes
- Sulle VM: Neutron security groups (iptables)
- ACL on the frontier router

Gestione chiavi e certificati?

5 Orchestration and automation

In this section we describe the automation techniques used for the installation, deployment and maintenance of the cloud platform and the deployed services, starting from the hardware resources, according to the architecture presented in Figure 4.

5.1 Automation Tools

We introduce briefly the automation tools used.

5.1.1 MaaS

Metal as a Service – MAAS – allows treating physical servers like virtual machines in the cloud. Rather than having to manage each server individually, MAAS turns bare metal into an elastic cloud-like resource.

MAAS can automatically boot machines, check the hardware and have them ready for use. Nodes can be pulled up or teared down just as with virtual machines in the cloud. MAAS can be a provider for Juju, providing the nodes it needs to power a service.

5.1.2 Juju

Juju is open source tool for automation and service orchestration. Juju has two components: a *bootstrap node* and a *client* present in each administered node. Juju uses a Postgres database for storing the state of configurations, which must be subject to HA.

Juju allows you to model, configure, manage, maintain, deploy and scale cloud services quickly and efficiently on public clouds as well as on MaaS, OpenStack, and LXD containers.

Service modelling allows quick and easy deployment and management of services (whether it's a cloud infrastructure like OpenStack, or a workload such as Hadoop), connecting those services, and quickly scaling them up or down, all without disruption to the cloud environment.

Juju provides dynamic configuration, which allows re-configuring services on the fly, adding, removing, or changing relationships between services, and scaling in or out.

Once a client Juju has been installed, Juju environments can be bootstrapped on different providers, which can be cloud services from multiple vendors. For local installation, in our case the provider used is Maas.

Juju is responsible for orchestrating services to be installed according to a high-level description in terms of charms (deployment recipes of a single service) or bundles (aggregations of charms in relation to each other). For each service, based on the architecture of figure 1, Juju takes care of:

- Deploying the service
- coordinate Event based reactions (e.g. elasticity of resource allocation in case of overload)
- integrating services among themselves

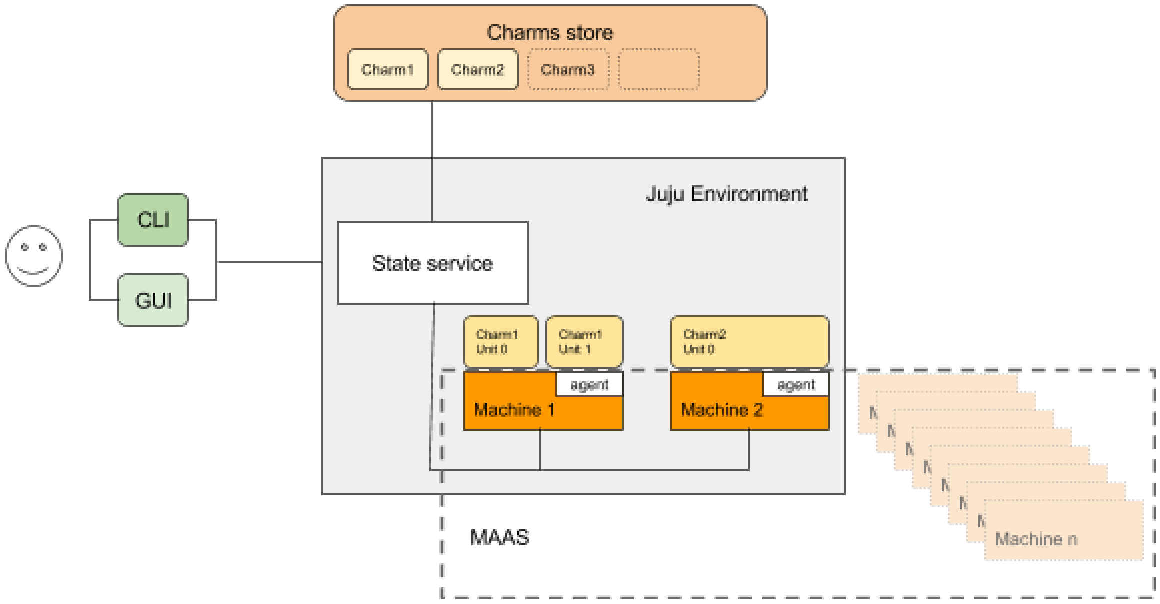


Figure 10. Juju architecture.

The Juju Charm Store (http://jujucharms.com) offers a wide range of charms and bundles for deployment of services on LXC, bare metal or KVM on a private, public or hybrid cloud.

For example, using a Juju charm it is possible to provision a PaaS service like WordPress, which uses a MySQL backend on a cluster. The service requires three types of nodes: load-balancing, WordPress (Apache/PHP) and MySQL database. Using the Juju charm, the user describes the number of nodes for each service, their dimensions (CPU, RAM, etc.) and how the nodes are related. The charm is sent to the bootstrap node, which interacts with OpenStack to implement the application. The process is shown in this figure:

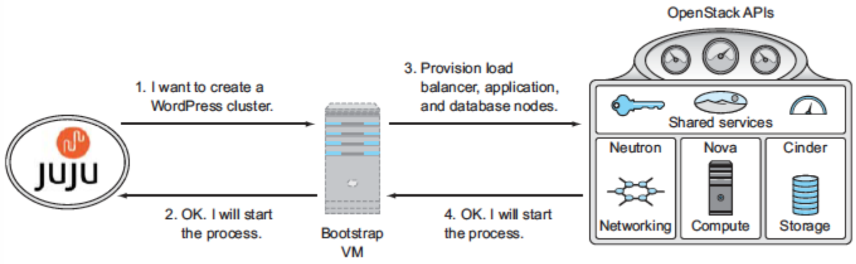

juju
1. I want to create a WordPress cluster.
2. OK. I will start the process.
Bootstrap VM
3. Provision load balancer, application, and database nodes.
4. OK. I will start the process.
OpenStack APIs
Shared services
Neutron
Networking
Nova
Compute
Cinder
Storage

Figure 11. Provisioning a PaaS through Juju.

OpenStack carries out the tasks requested to it through the bootstrap node and deploys the requested virtual infrastructure. The bootstrap node, once the VMs are available, starts a process outside of OpenStack to complete the installation. From now on, OpenStack simply provides the virtual infrastructure without being aware of the application roles assigned to each node.

Juju can also drive other configuration management tools, such as *Puppet* and *Ansible*.

5.1.3 Ansible

Ansible is an automation tool based on Python and YAML capable of performing configuration and maintenance tasks in parallel on multiple machines. Ansible is used as a maintenance tool for specific changes both at physical server level and at the control plane service level.

6 Building the Cloud

6.1 Installation of the Cloud Platform

Installing the cloud platform involves these steps:

1. Bare Metal provisioning with MAAS.
2. OpenStack provisioning with Juju.

For the details of these step we refer to the appendix.

6.1.1 Bare Metal Provisioning

To meet the requirement of being able to carry out (cross data center) **bare metal provisioning** with a unified management via either GUI or API, we opted for the tool Maas (Metal-as-a-Service [9]) by Canonical.

Maas integrates the FTP/PXE boot service, and optionally the DHCP and DNS services, according to the back-end architecture shown in Figure 12.

A Region Controller in HA is present in each region, which exposes the administrative functions through both an API and a Web GUI. Physical resources are organized in clusters, each managed by a Cluster Controller.

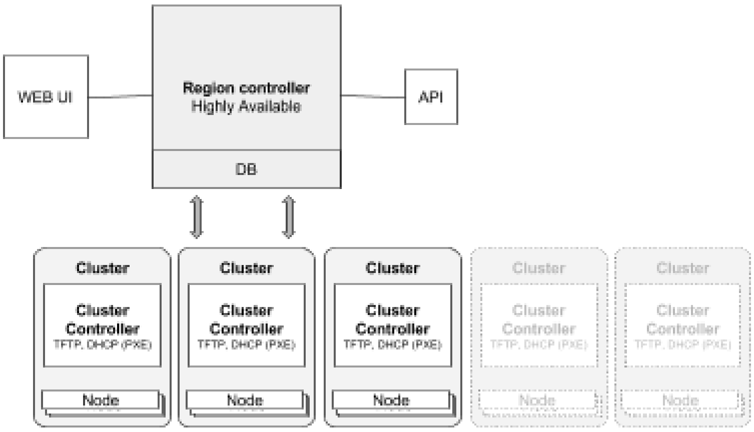


Figure 12. Scheme of Bare Metal provisioning in a region of the federation.

The Region Controller is configured in high availability and through the cluster controller orchestrates via DHCP and TFTP (PXE boot) a group of machines of which it takes charge:

- Enlisting the hardware
- Commissioning the machines
- Provisioning them

6.1.2 OpenStack Provisioning

Provisioning di OpenStack is done by means of Juju.

ESPANDERE

I charm di deployment sono disponibili su GitLab:

https://gitlab.global.garrservices.it/csdmgr/openstack-juju

6.2 Application Services Provisioning

The provisioning of application services is done by means of Juju.

ESPANDERE

The repository of the GARR bundles for Juju is available on GitLab:

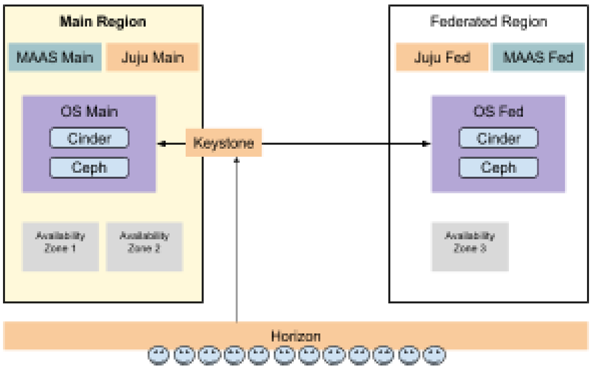
https://gitlab.global.garrservices.it/csdmgr/openstack-juju

7 Building a Region

Building a Region involves the following steps:

1. deploy MAAS/Juju/OpenStack on the group of servers and storage units that will form the region.
2. add a Keystone endpoint for the region. The URL should be that of the Keystone endpoint in the master region.
   Even better: force a keystone replica to a region node.
3. assign Availability Zones to the compute nodes.
4. Create domains with quotas set to given Availability Zones and Volumes?

The logical infrastructure is shown in the following figure:



Each region has its own installation of MAAS/Juju/OpenStack: they share a single Keystone. Users interact through Horizon or the Keystone API, requesting services according to their credentials. A user can be enabled for example to access both Availability Zone 1 and Availability Zone 3, according to the projects to which he is associated. Hence he can ask to start instances on either of those zones.

Volumes are provided through separate Ceph/Cinder on each region: hence volumes can be created on them and associated to an instance.

We need to figure out how a user can specify which Cinder to use.

Deploying PaaS with Juju

The architecture supports the deployment of PaaS through Juju as follows. A user can connect to the Juju client or GUI set with env:openstack, and asks for the deployment of a Juju bundle. For example an Hadoop bundle can be deployed, setting the number of nodes in the requested cluster, the volume sizes. Juju takes care of interfacing to the OpenStack clouds for provisioning the VM and installing the application packages on them.

7.2 Isolating Volumes among projects

Restricting volume access to projects within a region can be done as follows.

http://www.hitchnyc.com/openstack-multi-tenant-isolation/

Create new aggregate for new tenant

```
$ nova aggregate-create <name>
```

Take note of the ID of the aggregate just created.

Add hosts to new host aggregate

Add hosts to aggregate, one command for each host. Use the ID from the aggregate just created above.

```
$ nova aggregate-add-host <aggregate_name> <host_name>
```

Example:

```
$ nova aggregate-add-host <aggregate_name> compute01
$ nova aggregate-add-host <aggregate_name> compute02
$ nova aggregate-add-host <aggregate_name> compute03
```

Update host aggregate metadata

This step adds the project ID filter to the host aggregate, which will tell Nova Scheduler to restrict access to this host aggregate based on the project ID supplied.
Update aggregate metadata to include project ID filter:

```
$ nova aggregate-set-metadata <aggregate_ID> filter_tenant_id=<project_ID>
```

Create new flavor to include project ID filter

```
$ nova flavor-create <flavor name> <id> <ram> <disk> <vcpus>
$ nova flavor-key <flavor name> set filter_tenant_id=<project_ID>
```

Apply project quota for volume types

Cinder allows defining block storage based volume types. These volume types can be tied to separately defined Cinder backends. The process is explained here.
Update default project quota to allow and/or restrict a particular volume type.

Grant access rights

To allow access to a volume type:

```
$ cinder quota-update --volumes 100 --volume-type netapp_US <project_ID>
```

To restrict access to a volume type:

```
$ cinder quota-update --volumes 0 --volume-type netapp_US <project_ID>
```

Workout:
https://www.openstack.org/summit/vancouver-2015/summit-videos/presentation/multi-tenant-isolation-quarantine-your-tenants

https://github.com/wbentley15/os-summit-multitenant-wk/blob/master/multi_tenant_iso_lab.txt

7.3 Managing availability zones

An OpenStack availability zone is a logical partition of hosts or volume services within a single OpenStack deployment. Compute service availability zones are defined at the host configuration level, providing a method to segment compute nodes by arbitrary criteria, such as hardware characteristics, physical location, or task designation.

To create an availability zone in OpenStack, run the following steps:

1. On each Compute Node that you want include in the availability zone, add the following statement in the `/etc/nova/nova.conf`:
   ```
   node_availability_zone=availability_zone_name
   ```
2. Run the following commands:
   ```
   $ service openstack-nova-metadata-api restart
   $ service openstack-nova-compute restart
   ```

For more information about the OpenStack services, see the OpenStack documentation.

8 Administering the Cloud

8.1 Managing a Domain

Domains represent a team or an organization in a multi-project environment. The Cloud Administrator can perform the following steps for on-boarding a team in this environment.

8.1.1 Overview

1. Create a domain resource. This step automatically creates a default project for the domain to facilitate user on-boarding.
2. Ensure that the domain has access to at least one deployment availability zone. This allows users in that domain to access virtual images and deploy virtual servers when logged in to the domain projects. The availability zones that are assigned to the domain are then visible to be assigned to projects within the domain.
3. To delegate the domain administration, ensure that at least one user is assigned to the domain with **domain_admin** role. With this role, the Cloud Administrator can delegate the administrative tasks of the domain to the Domain Administrator who can then start creating projects and assigning users.

8.1.2 Creating a domain

The Cloud Administrator creates domains to organize projects, groups, and users.

GUI

1. Log in to the OpenStack Dashboard as the Cloud Administrator.
2. In the left navigation pane, click **IDENTITY > Domains**. The Domains page opens.
3. Select **Create Domain**. The Create Domain window is displayed.
4. Specify the domain name and, optionally, the domain description.
5. Optional: Clear the **Enabled** check box to disable the domain. If the domain is disabled, the Domain Administrator cannot create, update, or delete resources that are related to the domain. New domains are enabled by default.
6. Click **Create Domain.**

CLI

```
$ openstack role add --domain <domain> --user admin Admin
$ openstack domain create --description "description" <domain>
```

The second step is required in order to ensure that the cloud administrator can manage the new domain.

8.1.3 Assigning a zone to a domain

Assigning a zone to a domain enables users in a domain to access the resources available within a zone.

GUI

1. Log in to the OpenStack Dashboard as the Cloud Administrator.
2. Open the domains page by clicking **IDENTITY** > **Domains** in the navigation pane.
3. In the domains page, find the entry for the domain and click the arrow icon in the **Actions** column. Then click the **Edit** option to open the Edit Domain window.
4. Click the **Availability Zones** tab. The **Available Zones** and the **Assigned Zones** are listed in the

following format: `Zone_Name` - `Region_Name`

5. To assign a zone to a domain, from the list of **Available Zones**, click the plus button beside the zone name. The selected zone moves to the **Assigned Zones** list. To return an **Assigned Zone** to an **Available Zone**, select the minus button beside the zone name. Use the **Filter** field to search for specific zones.
6. When you have assigned all zones, click **Save**.

CLI

```
$ openstack domain create --description "description" <domain>
$ openstack role add --domain <domain> --user admin Admin
```

8.1.4 Managing Domain Quotas

The Cloud Administrator can manage quotas for a domain, specifying the maximum amount of a certain resource that is available to the domain. The Domain Administrator can then distribute that quantity among all the projects in the domain.

Using the command-line interface, the administrator can manage the following types of quotas.

- Compute service quotas
  - Set Compute quotas for a domain

    ```
    $ openstack quota-set --QUOTA_NAME QUOTA_VALUE <domain>
    ```

    For example, to set the maximum number of instances to 20, issue:

    ```
    $ openstack quota-set --instances 20 <domain>
    ```

    To see the current quotas values, do:

    ```
    $ nova quota-show --tenant <domain>
    +-----------------------------+-------+
    | Quota                       | Limit |
    +-----------------------------+-------+
    | instances                   | 10    |
    | cores                       | 20    |
    | ram                         | 51200 |
    | floating_ips                | 10    |
    | fixed_ips                   | -1    |
    | metadata_items              | 128   |
    | injected_files              | 5     |
    | injected_file_content_bytes | 10240 |
    | injected_file_path_bytes    | 255   |
    | key_pairs                   | 100   |
    | security_groups             | 10    |
    | security_group_rules        | 20    |
    | server_groups               | 10    |
    | server_group_members        | 10    |
    +-----------------------------+-------+
    ```

- Block Storage service quotas
  - View Block Storage quotas
  - Set Block Storage service quotas

- Networking service quotas
  - Basic quota configuration
  - Configure per-project quotas

8.1.5 Editing the domain quotas

The Cloud Administrator can change the quotas of a domain to set limits on the operational resources that a Domain Administrator can distribute among all the projects in the domain.

GUI

1. Log in to the OpenStack Dashboard as the Cloud Administrator.
2. In the navigation pane, click **IDENTITY > Domains**.
3. On the Domains page, find the entry for the domain that you want to modify. In the **Actions** column for that entry, click **More** > **Edit**.
4. In the **Edit Domain** window, click the **Quota** tab.
5. Edit the quota values as wanted.
6. Click **Save**.

8.1.6 Setting the Domain Administrator

Adding or removing users from the list of Domain Administrators to control a domain.

GUI

1. Log in to the OpenStack Dashboard as the Cloud Administrator.
2. In the navigation pane, click **IDENTITY** > **Domains**.
3. In the Domains page, select the entry for the domain. In the **Actions** column, click **More** > **Edit**. The **Edit Domain** window opens.
4. Click the **Domain Administrators** tab.
5. To add a Domain Administrator, click **+**. The user is promoted from Domain User to Domain Administrator for the default project only. You must manually add the Domain Administrator user to all other projects in the domain, as described in *Modifying user assignments for a project*.
6. To remove a Domain Administrator, click **-**. The user is demoted from Domain Administrator to Domain User for all projects in the domain, but is not removed from any projects.
7. Click **Save**.

CLI

```
$ openstack role add --domain $domain --user $id_of_user admin
```

8.1.7 Setting a domain context

Cloud administrators can set the context of a domain in order to limit its visibility, rather than having visibility across all domains. This allows the Cloud administrator to identify the projects, users, groups, and roles that are associated with a domain.

GUI

1. Log in to the OpenStack Dashboard as the Cloud Administrator.
2. In the left navigation pane, click **IDENTITY** > **Domains**.
3. In the domains page, find the entry for the domain and click **Set Domain Context**.

8.1.8 Clearing the domain context

Cloud administrators can clear the scope of all domains, enabling visibility across all domains.

GUI

1. Log in to the OpenStack Dashboard as the Cloud Administrator.
2. In the left navigation pane, click **IDENTITY > Domains**.
3. In the domains page, select **Clear Domain Context** from the top right-hand corner.

8.1.9 Managing security groups

The Cloud Administrator can create, modify, or delete security groups in a domain.

GUI

1. Log in to the OpenStack Dashboard as the Cloud Administrator.
2. In the navigation pane, click **PROJECT** > **Access & Security**. In the Access & Security panel, you can create, modify or delete a security group.
3. To modify a security group, click **Manage Rules** for the group that you want to modify and add or delete rules for the security group.

8.2 Managing Projects

The Cloud or Domain administrator can set the level of access for each project with the user interface.

8.2.1 Creating a project

The administrator can assign individual zones to a domain with the OpenStack Dashboard.

GUI

1. Log in to the OpenStack Dashboard as a Cloud or Domain Administrator.
2. Open the projects page by clicking **IDENTITY** > **Projects** in the navigation pane.
3. Click **Create Project**. The Create Project window is displayed.
4. Specify the name for the project.
5. Optional: Enter a description for the project
6. Optional: By clearing the **Enabled** check box, the project is disabled and users cannot log into it.
7. Click **Create Project**.

CLI

```
$ openstack project create $project --domain $domain
```

See http://docs.openstack.org/mitaka/install-guide-obs/keystone-users.html

8.2.2 Enabling a project

Enabling a project allows you to set that project as your default project. The action only appears if the project is disabled.

8.2.3 Editing a project

You can modify the name and description of a project.

8.2.4 Disabling a project

Disabling a project in a domain means that the users who previously had that project set as their default cannot log in to it anymore. Other users also cannot switch to this project anymore.

8.2.5 Deleting a project

Delete a project in the OpenStack Dashboard as the Cloud or Domain Administrator.

8.2.6 Assigning a zone to a project

Assigning a zone to a project enables users within a zone to access a specific project.

GUI

1. Log in to the OpenStack Dashboard as the Cloud or Domain Administrator.
2. Open the domains page by clicking **IDENTITY** > **Domains** in the navigation pane.
3. In the domains page, find the entry for the domain and select **Set Domain Context** in the **Actions** column. The **Identity Panel** group is now in the context of the selected domain and the **Domains** page is also changed. You are now working within the context of the domain that you created.
4. Select **IDENTITY** > **Projects**.
5. In the **Actions** column in the table for the project, click the arrow icon then click the **Edit Project** option.
6. Click the **Availability Zones** tab. The available zones and the assigned zones are listed in the following format: `Zone_Name` - `Region_Name`.
7. To assign a zone to a domain, from the list of **Available Zones**, click the plus button beside the zone name. The selected zone moves to the **Assigned Zones** list. To return an **Assigned Zone** to an **Available Zone**, select the minus button beside the zone name. Use the **Filter** field to search for specific zones.
8. When you have assigned all zones, click **Save**.

8.2.7 Configuring project quotas

The Cloud or Project Administrator can configure the project quotas, in order to limit the following resources:

- Number of volumes that can be created
- Total size of all volumes within a project as measured in GB
- Number of instances that can be started
- Number of processor cores that can be allocated
- Publicly accessible IP addresses

Quotas can be enforced at both the project and the project-user level.

- Compute service quotas
    - Set Compute quotas for a project (project)
    - Set Compute quotas for a project user
- Block Storage service quotas
    - View Block Storage quotas
    - Set Block Storage service quotas
- Networking service quotas
    - Basic quota configuration
    - Configure per-project quotas

8.3 Managing Nested Quotas

Nested quotas allow deciding how to divide resources among hierarchical sub-projects.

For example, the following commands updates the quota of volumes assigned to a project:

```
$ cinder quota-update PROJECT_ID --volumes 10
+-----------------------+-------+
|        Property       | Value |
+-----------------------+-------+
|    backup_gigabytes   |   0   |
|        backups        |   0   |
|       gigabytes       |   0   |
| gigabytes_lvmdriver-1 |   0   |
|  per_volume_gigabytes |   0   |
|       snapshots       |   0   |
| snapshots_lvmdriver-1 |   0   |
|        volumes        |   10  |
|  volumes_lvmdriver-1  |   0   |
+-----------------------+-------+
```

Once these volumes are assigned to the project, the amount of volumes that can be used by sibling projects is reduced correspondingly.

8.4 Managing Groups

You can manage the level of access for each group in the Cloud with the user interface.

8.5 Managing Users

You can manage the level of access for each individual user.

8.5.1 Create a user

8.5.2 Add a user to a project

Add a user to a project and assign the user the role "Member" in the project:

```
$ openstack role add --project $project_id --user $id_of_user Member
```

8.6 Managing Networks

As a cloud administrator you can manage networks in the Cloud with the user interface.

9 Operations

This section should cover the steps to be performed for operating the cloud platform.

The book OpenStack Operations Guide by T. Fifield et al. O'Reilly 2014, presents a full guide on OpenStack operations.

9.1 Front Desk

Describe interaction with users.

Ticketing system and escalation procedures.

9.2 Monitoring

Describe how to configure and setup the monitoring tools (Zabbix, Monarca) and how to keep an eye on them.

9.3 Maintenance, Failures and Debugging

9.3.1 Cloud controller

9.3.2 Compute Nodes

9.3.3 Storage Nodes

9.3.4 Databases

9.4 Backup and Recovery

The OpenStack Guide on Backup and Recovery, describes how to back up the configuration files and databases that the various OpenStack components use:

- Database Backups
- File System Backups
    - Compute
    - Image Catalog and Delivery
    - Identity
    - Block Storage
    - Object Storage
    - Telemetry
    - Orchestration

Freezer is a distributed backup restore and disaster recovery as a service platform, capable of performing automated backups of:

- MySQL
- Nova
- Cinder
- MongoDB

10 Cloud Facilities for Users

This section sketches which facilities are exposed to users of the cloud. A more detailed *User's Guide* will be published separately.

10.1 Images

10.1.1 Creating, selecting, sharing images

10.2 Instances

10.2.1 Choosing, creating flavors

10.2.2 Starting, deleting instances

10.2.3 Security groups

10.2.4 Taking snapshots

10.3 Volumes

10.3.1 Creating, attaching volumes

10.4 PaaS

10.4.1 Deploying a Juju bundle

Acknowledgments

We greatly acknowledge the contributions from the members of the GARR Task Force on Federated Cloud: Marco Aldinucci, Massimo Coppola, Paolo De Rosa, Davide Salomoni, Giovanni Ponti. Very helpful and critical comments were provided by Silvio Cretti and Giuseppe Cossu.

12 References

1. GARR CSD. Requisiti per piattaforma Cloud GARR. 2016. https://docs.google.com/document/d/14tlba-ohpKdqHT0te1MvdmuSfRKCRt9HlYilY6qf0Zk/
2. GARR. Federazione Nazionale di Identità IDEM. http://www.idem.garr.it
3. Juniper. Contrail Architecture. http://www.juniper.net/us/en/local/pdf/whitepapers/2000535-en.pdf
4. Federated keystone. http://docs.OpenStack.org/security-guide/identity/federated-keystone.html
5. Configuring Keystone for Federation http://docs.OpenStack.org/developer/keystone/configure_federation.html
6. Keystone support for OpenID Connect. http://docs.OpenStack.org/developer/keystone/federation/openidc.html

7. Migrating a legacy cloud infrastructure to Contrail SDN based infrastructure. http://www.opencontrail.org/migrating-a-legacy-cloud-infrastructure-to-contrail-sdn-based-infrastructure/
8. Canonical. Juju architecture and details. http://www.ubuntu.com/cloud/juju
9. Canonical. MaaS architecture and details. http://MaaS.io
10. FIWARE. Monitoring a Multi-region Cloud Based on OpenStack. http://superuser.openstack.org/articles/monitoring-a-multi-region-cloud-based-on-openstack
11. HP. HP Reference Architecture for OpenStack on Ubuntu 14.04 LTS. http://h20564.www2.hpe.com/hpsc/doc/public/display?docId=c04330703
12. OpenStack. Tricicle. https://wiki.openstack.org/wiki/Tricircle
13. Aswin Nares. 2015. Simple Cloud Federation using Availability Zones. https://blueprints.launchpad.net/nova/+spec/simple-cloud-federation-using-availability-zones

13 Appendix - Installation of the Cloud Infrastructure

This section provides step by step instructions for installing the cloud infrastructure for a region to be part of the GARR Federated Cloud.

Requirements

It is recommended to use servers with:

- CPU: 2+ cores (physical or vcpu)
- RAM: 4+ GB
- NET: 2+ networks (one dedicated to IPMI

13.1 Installing the MAAS server

1. Choose a server, or better an LXC container, to become the MAAS server. Follow the steps provided at http://maas.ubuntu.com/docs/install.html to install MAAS on the server. Major setup steps include:
    - Install maas, maas-dhcp, and maas-dns packages
    - Create super user account
    - In "Settings" tab:
        i. under "Ubuntu" section, add "http://mi.mirror.garr.it/ubuntu/" as <<Main archive>>
        ii. under "Boot Images" section, add "http://maas.ubuntu.com/images/ephemeral-v2/daily/" as <<Sync URL>>
    - Import boot images
    - Setup DHCP service

2. Set the rest of the servers to PXE boot.
    - Connect to each server through iLO.
    - Set the Server Boot Order to boot from 'Network Device 1' first.
    - Reboot each server. This will cause the node to PXE from the MAAS server and initial discovery will take place.

3. After the nodes are discovered, log on to the MAAS server web interface at `http://${my-maas-server}/MAAS`, where `${my-maas-server}` should be replaced with the hostname of your MAAS server. Edit each node to verify MAAS has correctly detected configuration information for the iLO. You can associate the iLO ip-address with the node by comparing the mac addresses of the nodes that are discovered with those reported in the iLO System Information->Network tab. The following information in the MAAS Edit node screen should be accurate to ensure nodes are fully manageable:
    - Power Type: IPMI
    - Power Driver: LAN_2_0 [IPMI 2.0]
    - IP Address <The IP Address of the iLO>
    - Username: <iLO User Name>
    - Password: <iLO Password>

4. If you are using IPMI connection for connecting hardware:
    - From the MaaS GUI, click on "Add Hardware" and then "Machine"
    - Insert name in all the form boxes
    - Select IPMI from the "Power Type" list and compile forms
    - Press the "Save Machine" button. The hardware will reset and boot automatically.

5. In the MAAS web interface, set default distro series used for deployment commissioning and deployment to Ubuntu 16.04 LTS. Accept and commission each node via the web interface to install the selected series of Ubuntu.

6. Follow the steps documented at http://maas.ubuntu.com/docs/tags.html to add tag "controller" and

"storage" to each controller and storage server, respectively. We will later use these tags to ensure OpenStack services get deployed to the appropriate nodes.

13.2 Install Juju

Follow the steps provided at https://maas.ubuntu.com/docs/juju-quick-start.html to install Juju on the MAAS server.

1. Create SSH keys

   Juju requires SSH keys to be able to access the deployed nodes. In case those keys do not exist, then we have to create them before we bootstrap our environment:
   ```
   $ ssh-keygen -t rsa
   ```

2. Get API key

   You'll need an API key from MAAS so that the Juju client can access it. To get the API key, go to your MAAS home page `http://${my-maas-server}:80/MAAS/` and choose Preferences from the drop-down menu that appears when clicking your username at the top-right of the page.

3. Adding an SSH key

   While you are still on the MAAS preferences page, add your SSH key by clicking Add SSH key. Use the public half of your SSH key, the content of `~/.ssh/id_rsa.pub` for example; do not paste the private half

4. Install juju client needed to deploy juju controllers on nodes
   ```
   $ sudo apt-get install juju
   ```

5. Creating environments.yaml

   Create or modify `~/.juju/environments.yaml` with the following content:
   ```
   environments:
        Maas:
            type: maas
           maas-server: 'http://${my-maas-server}:80/MAAS'
            maas-oauth: '${maas-api-key}'
            admin-secret: ${your-admin-secret}
            default-series: trusty
            bootstrap-timeout: 3600
   ```
   Substitute the API key from earlier into the `${maas-api-key}` slot, and the hostname of your MAAS server into the `${my-maas-server}` slot.

   The `${your-admin-secret}` slot should be replaced with a random pass-phrase, there is no default. You can later use this pass-phrase to login to Juju node or Juju GUI.

   "bootstrap-timeout" increases the default timeout value from 10 minutes to 1 hour.

6. Bootstrap the MAAS configuration

   Execute the bootstrap step to deploy a controller node:
   ```
   $ juju bootstrap --constraints tags=controller
   ```
   The master node may take a long time to come up. It has to completely install Ubuntu and Juju on it and reboot before it will be available for use. It is probably worth following the install on the node directly via iLO remote console.

   After bootstrap is completed, as an optional step, you can install the Juju GUI to help with the tasks of managing and monitoring your Juju environment:
   ```
   $ juju deploy juju-gui --to <<node>> #insert name of the desired node
   ```

13.3 Deploying the Cloud Infrastructure with Juju

1. Download the GARR Reference Architecture bundle <nome del bundle> from <URL del bundle>.

2. Edit the configuration file bundle.yaml to suite your needs.

For example you can increase the number of Nova Compute units.
There are more options for each charm, look at each respective charm's `config.yaml`.

3. Deploy OpenStack with Juju:

   ```
   $ juju deploy -c bundle.yaml
   ```

   If the deployment stops or you see errors, try repeating the command adding “--debug” in order to see detailed output messages.

4. Expose the services you want (optional)

   The last step is to expose the services that should be made available to outside requests and opening the required firewall ports in the security group. Depending on charm versions, this step is optional since corresponding ports may be opened up by default.

   ```
   $ juju expose openstack-dashboard
   $ juju expose nova-cloud-controller
   ```

13.4 Validation

At this point, the Openstack cloud has been deployed and should be functioning.

1. Point your browser to the public address of the openstack-dashboard node, `http://${node-address}/horizon`. Use the command "`juju status openstack-dashboard`" to get its IP address.
2. Login using `admin/${admin-password}` (password defined in `openstack.cfg` above) and you can begin using the cloud, adding users, etc.