Descrizione dell'architettura dei componenti standard di Openstack

Marica Antonacci - INFN Bari

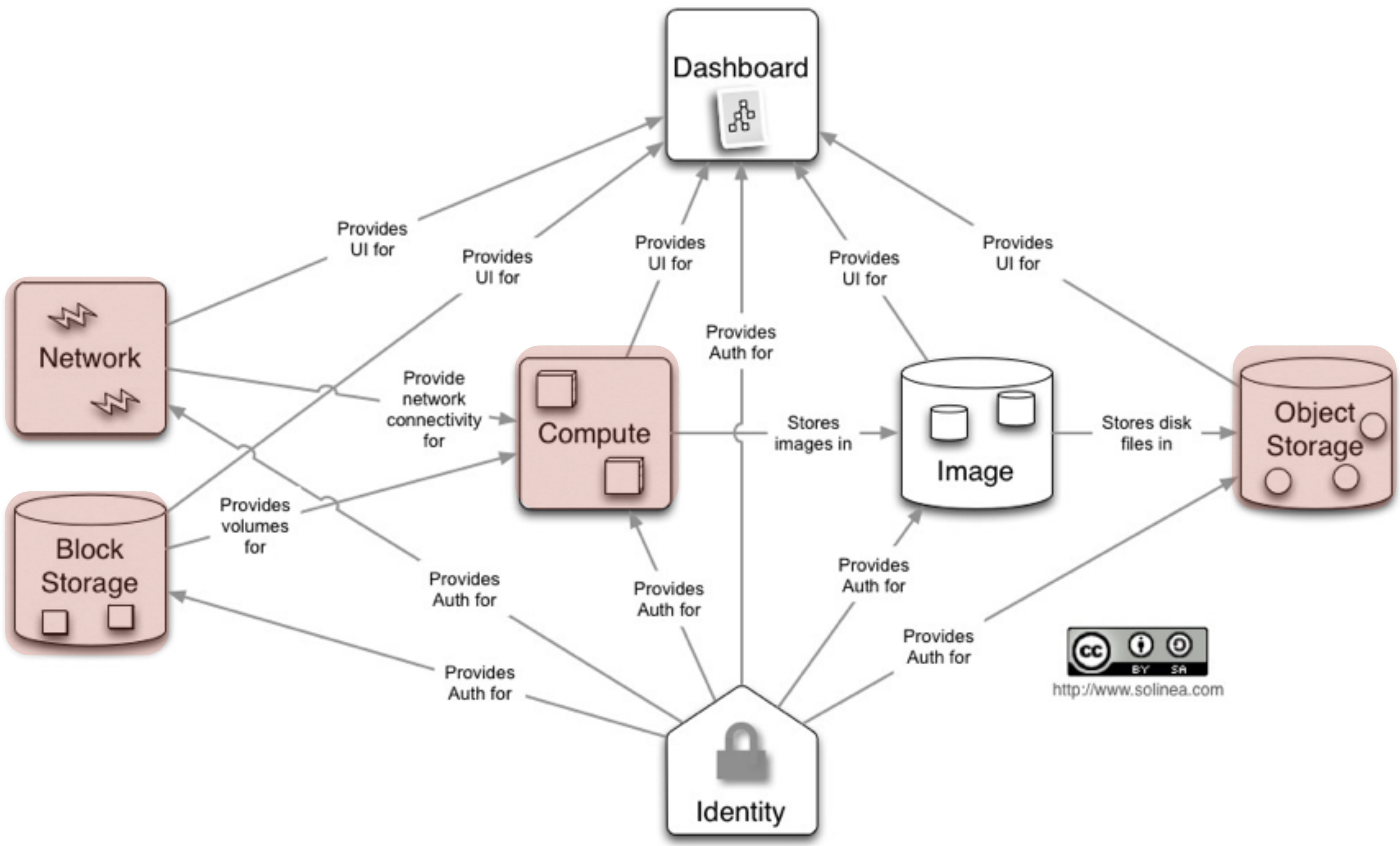
Scuola di Cloud Computing
Bari, 27-30 Aprile 2015

Outline

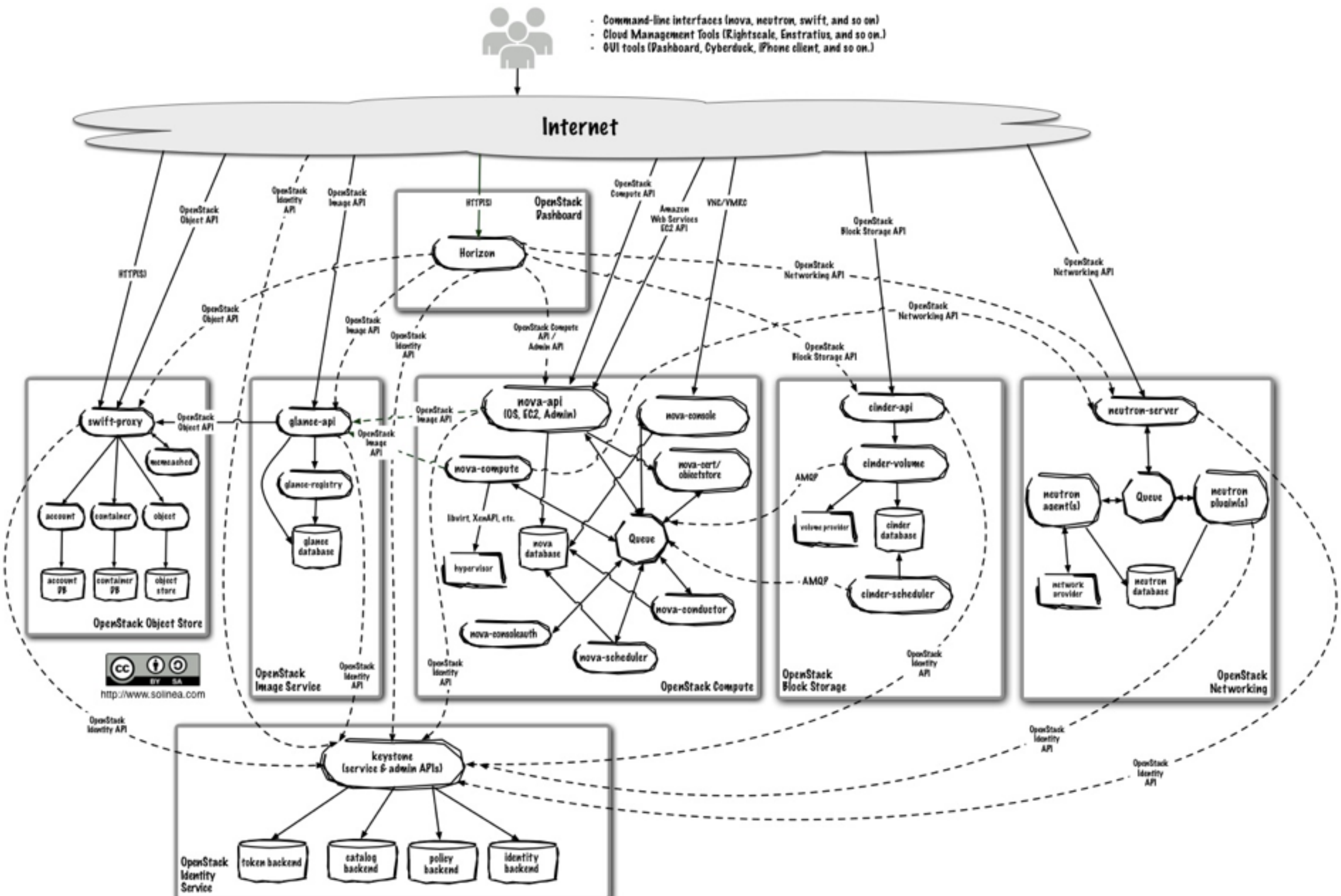
- Openstack architecture
- Openstack: multiple solutions for storage, deployment tools, hypervisors,…
- Installation roadmap

Openstack Architecture

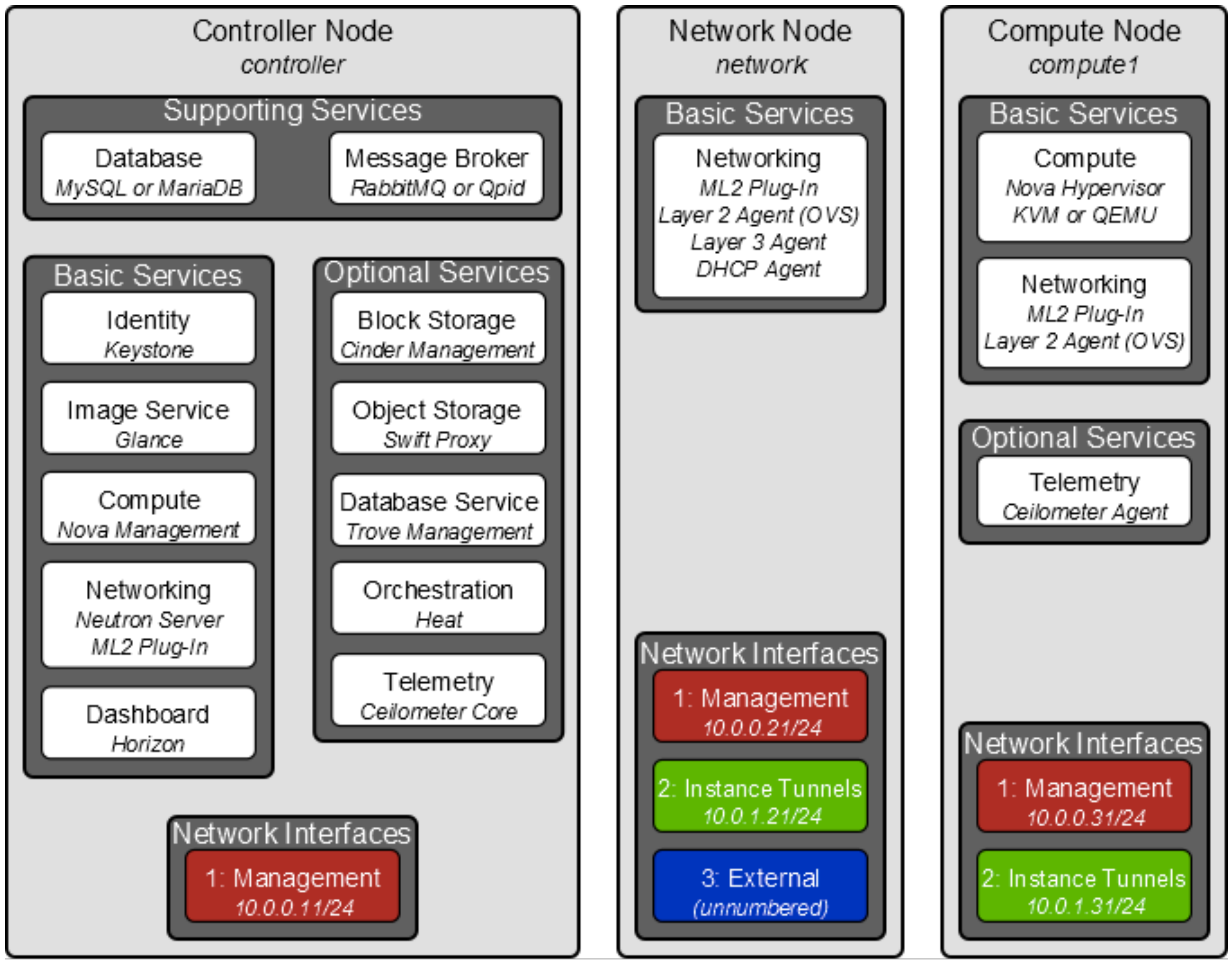
Conceptual architecture



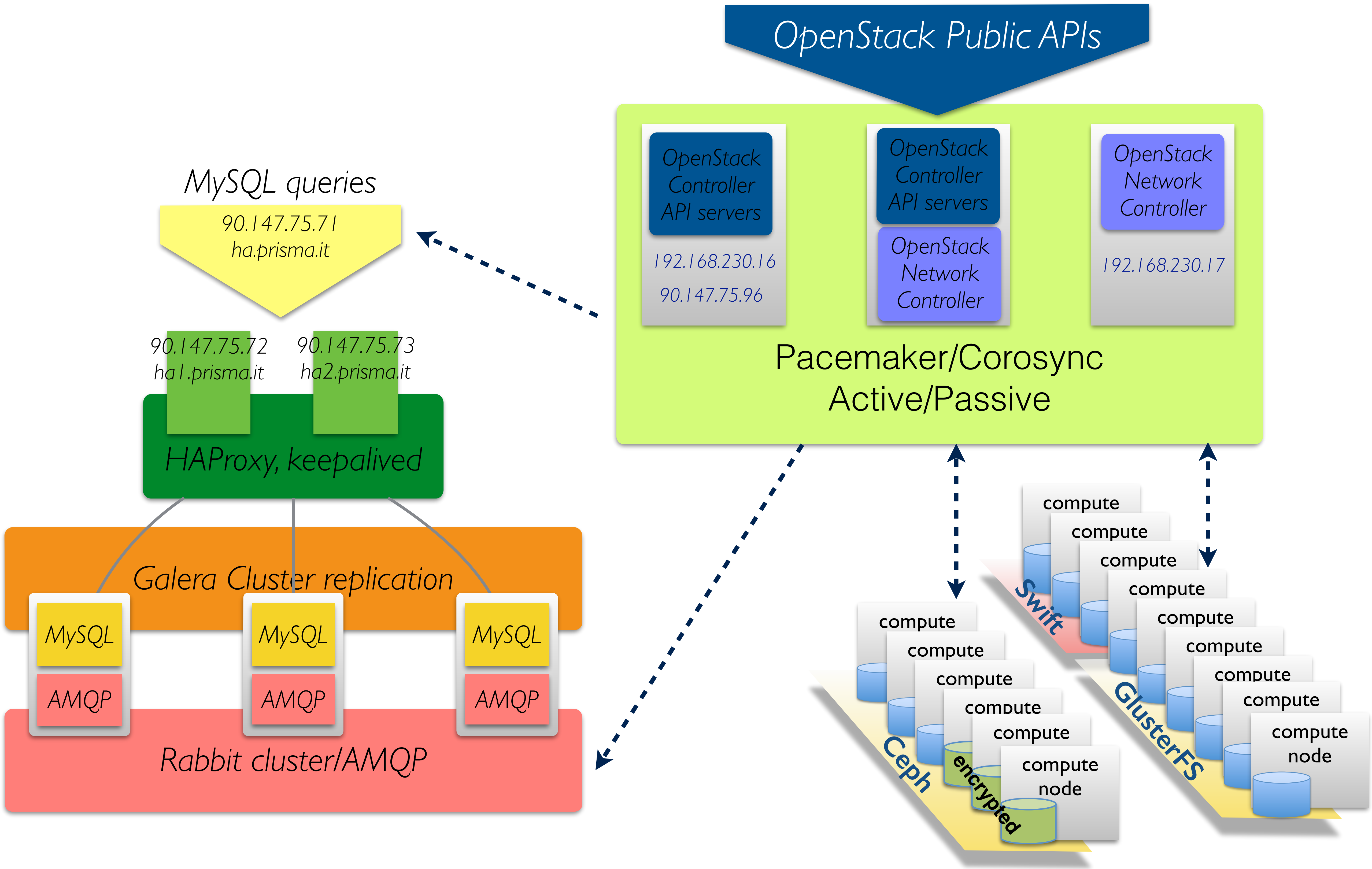
Logical Architecture



A typical service placement

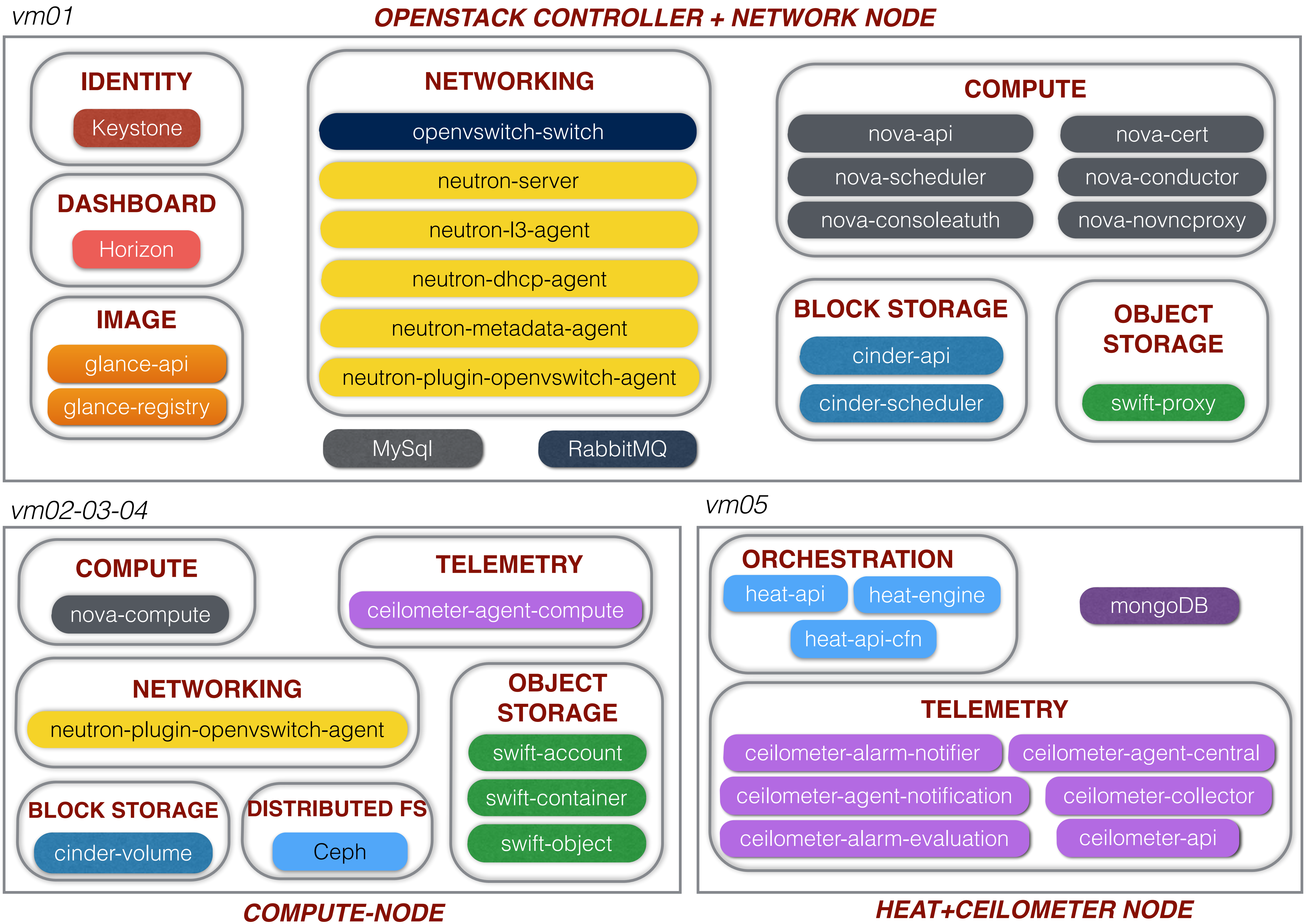


PRISMA testbed architecture



Tutorial Testbed Architecture

https://github.com/infn-bari-school/OpenStack-Icehouse-Installation



Manual Installation Roadmap

Pre-requirements:

- Step 1: Network interfaces configuration
- Step 2: Install Network Time Protocol (NTP)
- Step 3: Install the MySQL Python library
- Step 4: Install Openstack Packages on all nodes
- Step 5: Modify the /etc/hosts file
- Step 6: Distributed Filesystem Installation

Basic services installation:

◉ *Controller and Network node installation:*

- Step 1: MySQL installation/configuration
- Step 2: Install the message broker service
- Step 3: Install Identity service
- Step 4: Install Image service
- Step 5: Install Compute service
- Step 6: Install Networking service (Neutron)
- Step 7: Install the dashboard (Horizon)

◉ *Compute node installation:*

- Step 1: Install Compute components
- Step 2: Install Networking components

Basic Services:

Database

Message Broker

Basic services: Database & Messaging Server

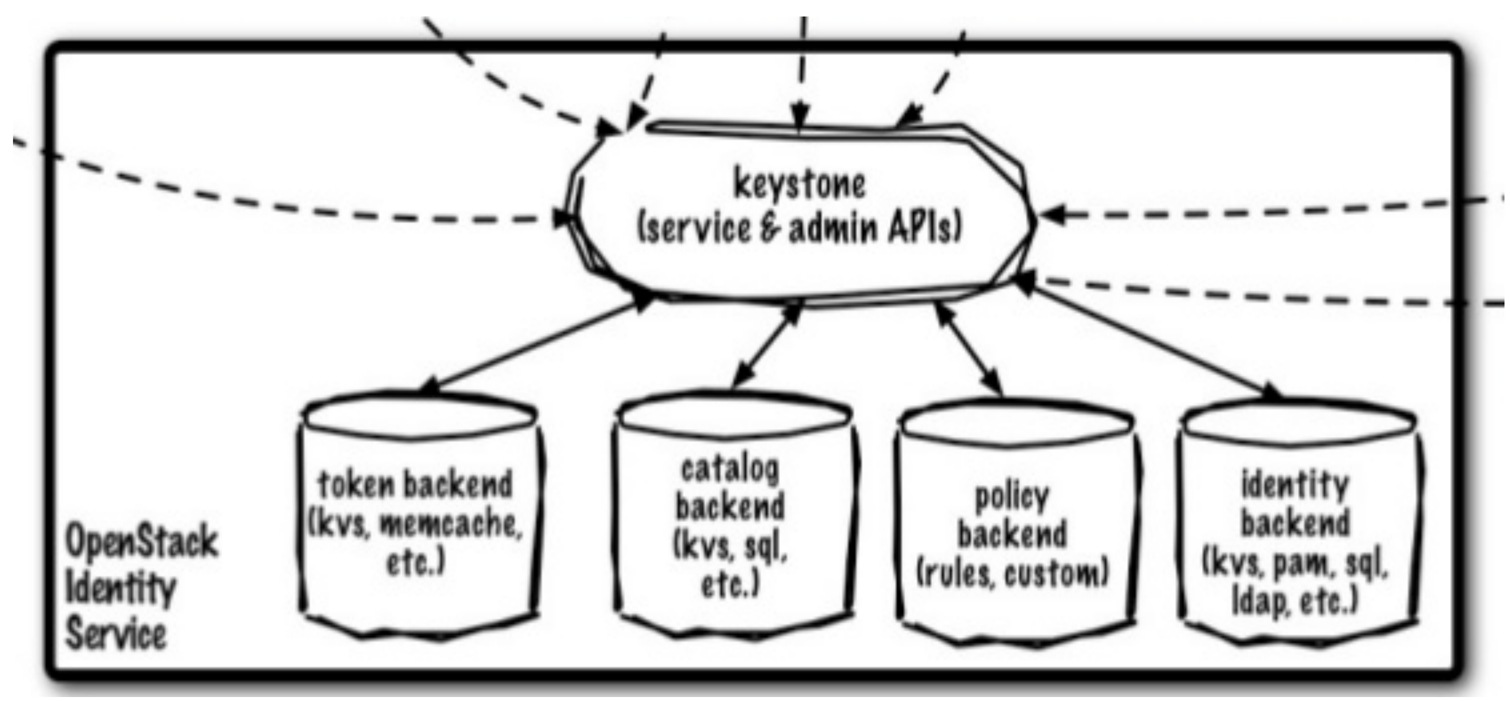
- OpenStack components use an SQL database and an AMQP-compatible system to share the current status of the cloud and to communicate to each other.
    - Multiple SQL databases and AMQP systems are supported. The most common used are MySQL and RabbitMq
- Since the architecture is highly distributed, a request done on an API service (for instance, to start a virtual instance), will trigger a series of tasks, possibly executed by different services on different machines. Usually the status of the request is saved on the database, and whenever some additional task is needed by a different service, the AMQP system is used to request it.
- MySQL and RabbitMQ are therefore very important and very basic services, used by all the OpenStack components. If something is not working here, or not working properly, the whole cloud could be unresponsive or broken. —> Highly recommended: High-Availability implementation

Identity Service

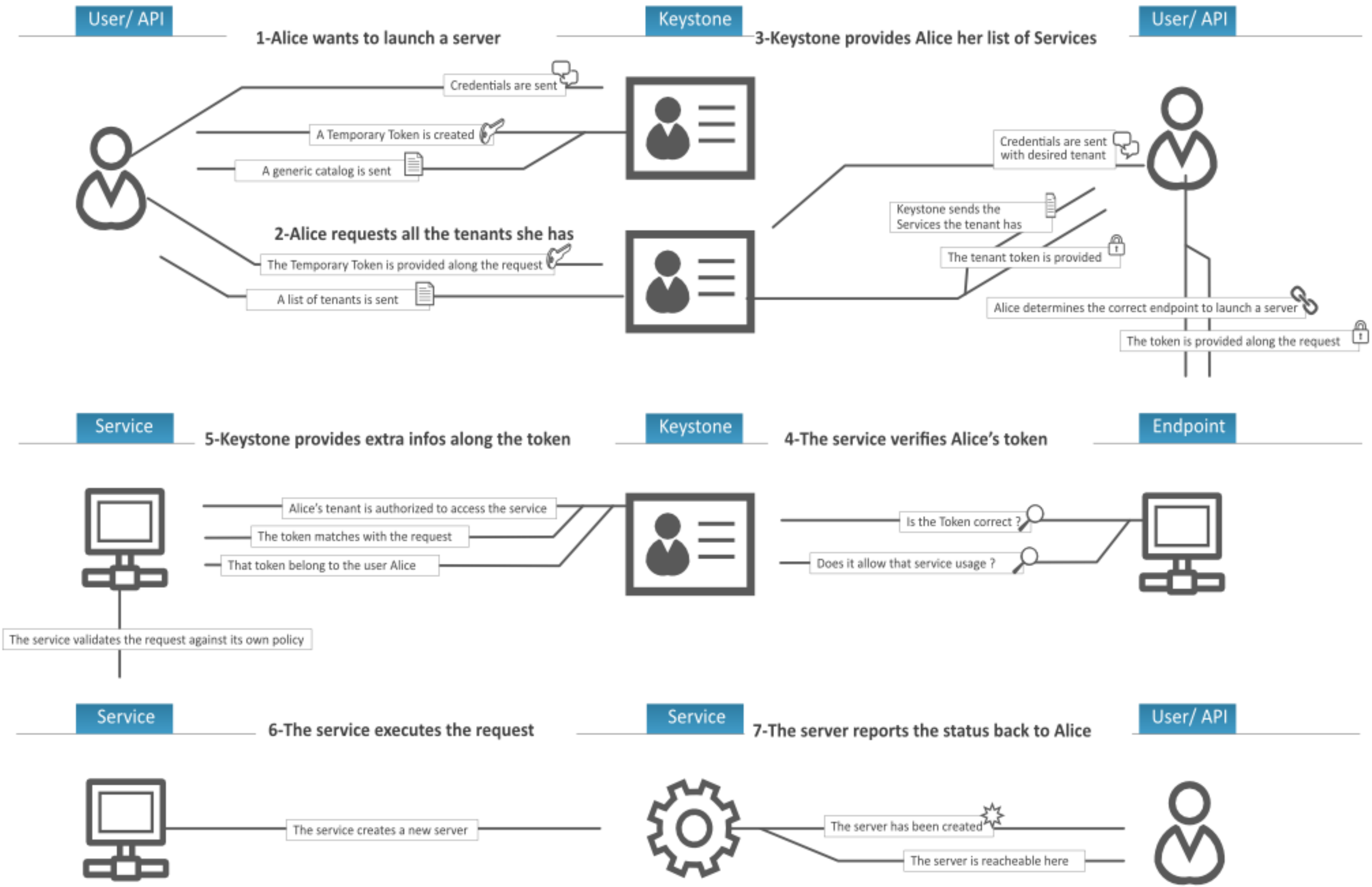
The Identity Service: keystone

Keystone stores information about different, independent services:

- users, passwords and tenants
- authorization tokens
- service catalog
- policy



The Keystone Identity Manager



Keystone installation

1. Install the required packages
2. Create a database and a user “keystone”
3. Edit the configuration file */etc/keystone/keystone.conf*
4. Create the database tables running the command:

    keystone-manage db_sync

5. Restart the service
6. Verify the process is running

```
root@vm01:~# netstat -pln | grep `pgrep keystone-all`
tcp        0      0 0.0.0.0:35357           0.0.0.0:*               LISTEN      1746/python
tcp        0      0 0.0.0.0:5000            0.0.0.0:*               LISTEN      1746/python
```

The chicken and egg problem

- In order to create users, projects or roles in keystone you need to access it using an administrative user (which is not automatically created at the beginning), or you can also use the "admin token", a shared secret that is stored in the keystone configuration file and can be used to create the initial administrator password.

- Keystone listens on two different ports, one (5000) is for public access, while the other (35357) is for administrative access. You will usually access the public one but when using the admin token you can only use the administrative one.

- In our case, since we don't have an admin user yet we need to use the admin token

Credentials

- Admin Token & Service Endpoint variables:

```
export OS_SERVICE_TOKEN=`grep admin_token /etc/keystone/keystone.conf | cut -d= -f2`
export OS_SERVICE_ENDPOINT=http://controller:35357/v2.0
```

- User credentials & auth_url variables:

```
export OS_USERNAME=admin
export OS_PASSWORD=xxxx
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://controller:5000/v2.0
```

Service catalog

- Keystone is not only used to store information about users, passwords and projects, but also to store a catalog of the available services the OpenStack cloud is offering.

    - **publicurl** is the URL of the client API, and it's used by command line clients and external applications.

    - **internalurl** is similar to the publicurl, but it's meant to be used by other OpenStack services, that might not have access to the public address of the API, but might be able to access directly the internal interface of the API node.

    - **adminurl** is used to expose the administrative API. For instance, in keystone, creation and deletion of a user is considered an administrative action and therefore will use this URL.

Policy.json

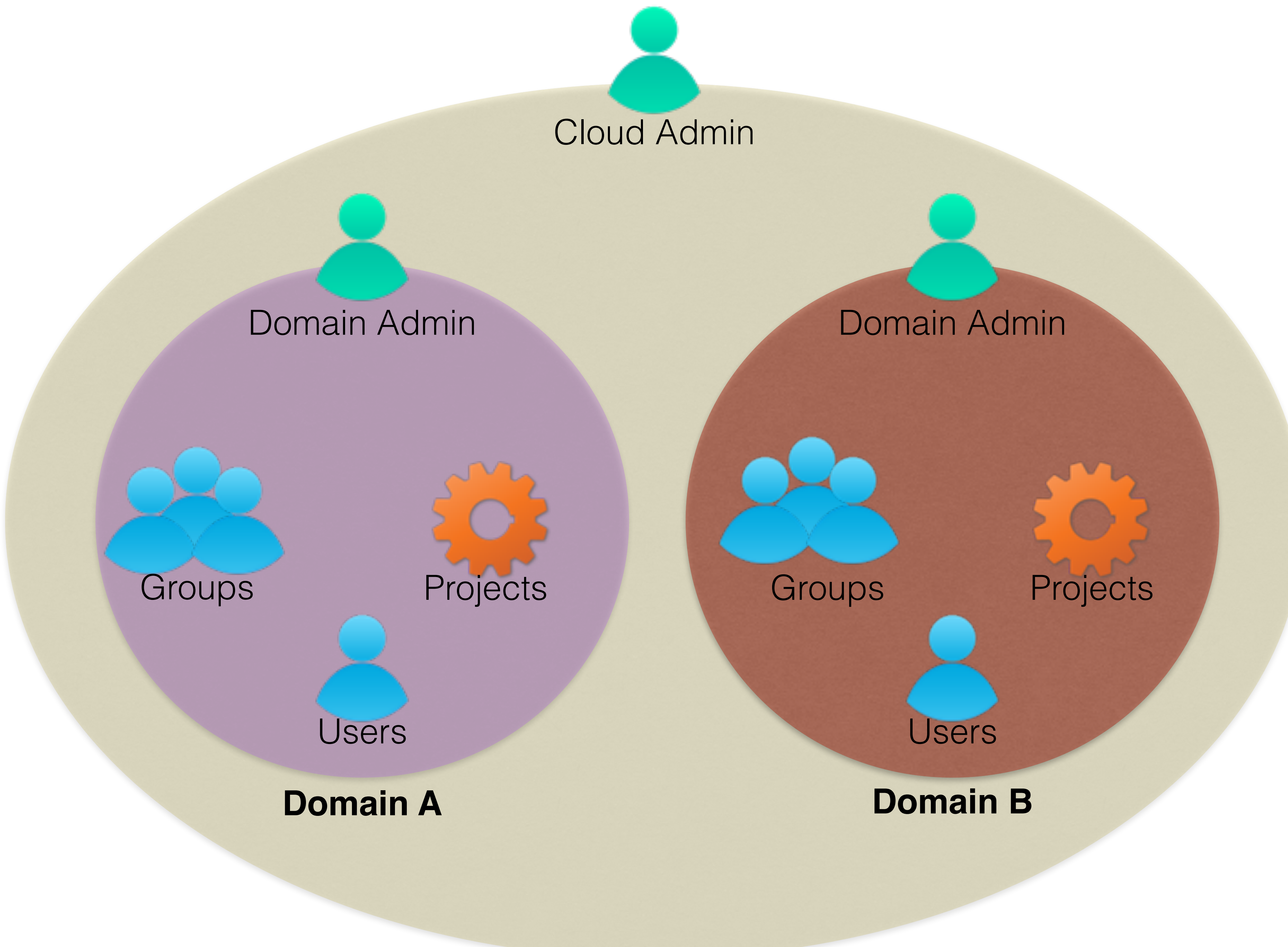
- The Policy service provides a rule-based authorization engine and the associated rule management interface.

- Rules are stored in /etc/keystone/policy.json

```
{
    "admin_required": [["role:admin"], ["is_admin:1"]],
    "service_role": [["role:service"]],
    "service_or_admin": [["rule:admin_required"], ["rule:service_role"]],
    "owner" : [["user_id:%(user_id)s"]],
    "admin_or_owner": [["rule:admin_required"], ["rule:owner"]],
    "default": [["rule:admin_required"]],

    "identity:get_service": [["rule:admin_required"]],
    "identity:list_services": [["rule:admin_required"]],
    "identity:create_service": [["rule:admin_required"]],
    "identity:update_service": [["rule:admin_required"]],
    "identity:delete_service": [["rule:admin_required"]],
```

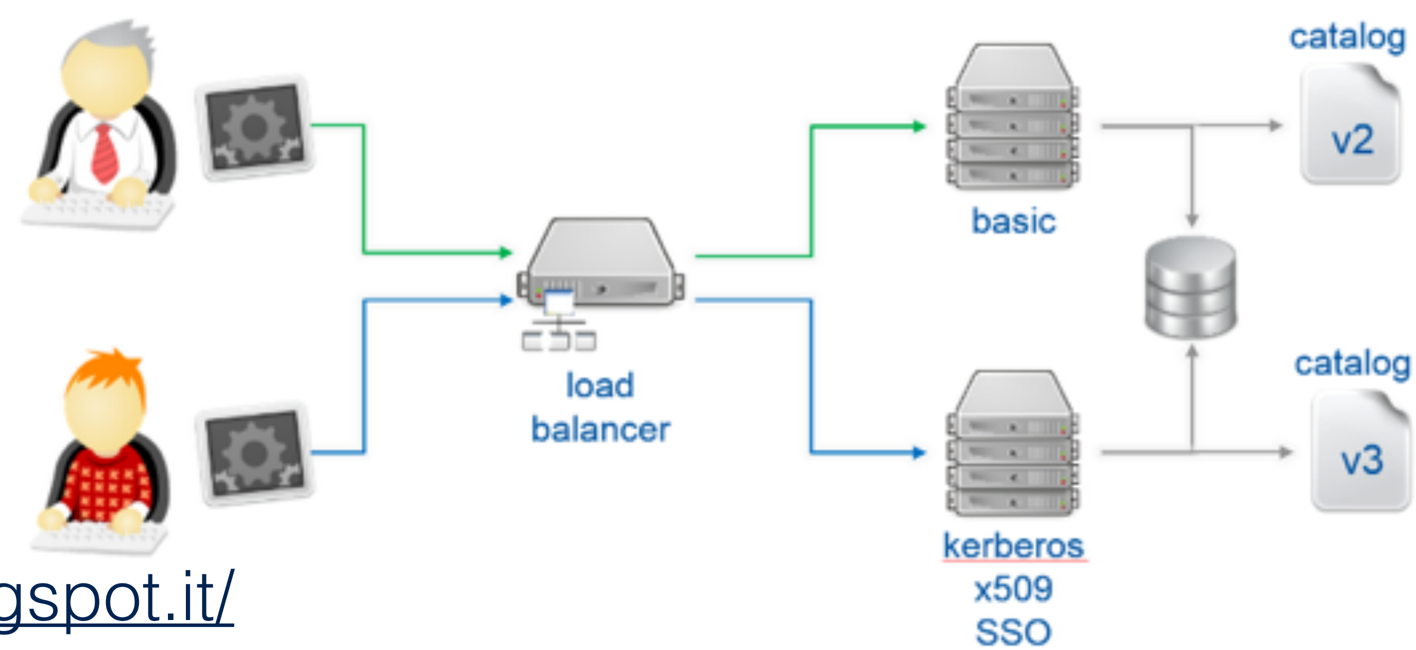
Keystone v3 API: domains



- Domain aware policy allows for delegation of administration tasks (policy.v3cloudsample.json)

keystone API version: compatibility issues in Icehouse

- V2 and V3 keystone API endpoints can coexist, but…
  - some openstack components do not fully support v3 API (neutron, horizon)
- Load-balancer (CERN solution*)
  - calls two different backend machines
  - ensures the services migration from one API version to the next one with no downtime



* http://openstack-in-production.blogspot.it/

Federated Identity

- keystone has generic federated capabilities, which allows external users to be recognized
- Users are not persistent in keystone, but are created on the fly from external user information provided by trusted identity provider
- Use cases:
    - existing internal IdP, Single-Sign-On
    - Inaccessible LDAP identity source
    - Non-LDAP identity source

Federated Identity: Keystone & SAML

- keystone in Apache HTTPD
  - leverages mod_shib
  - OS-FEDERATION extension
  - mapping: convert assertion to a user group

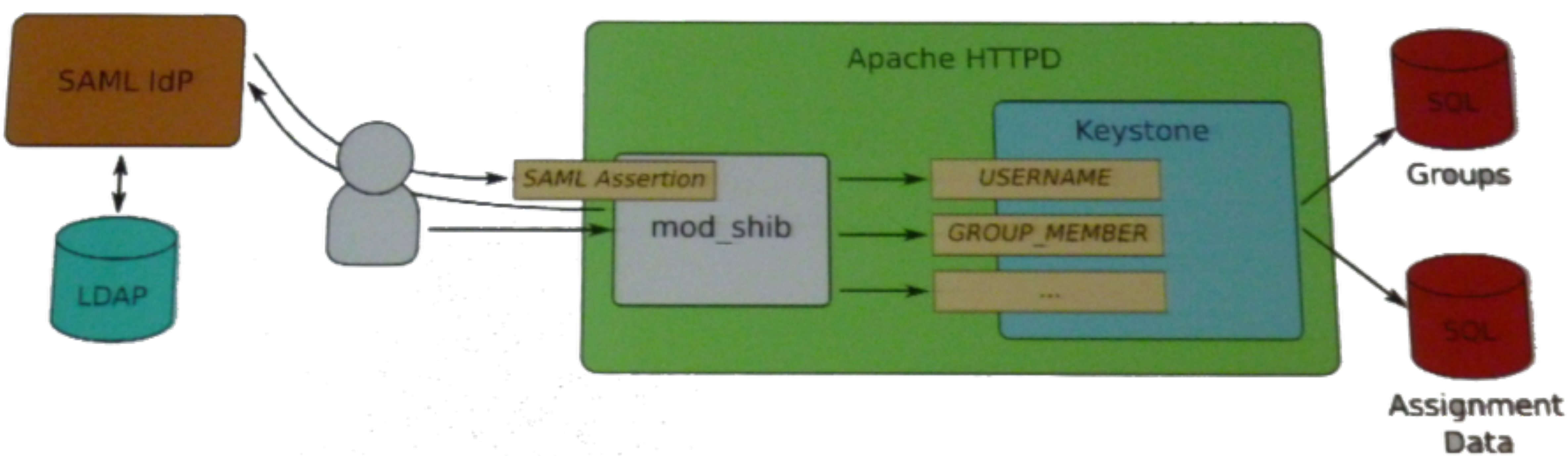
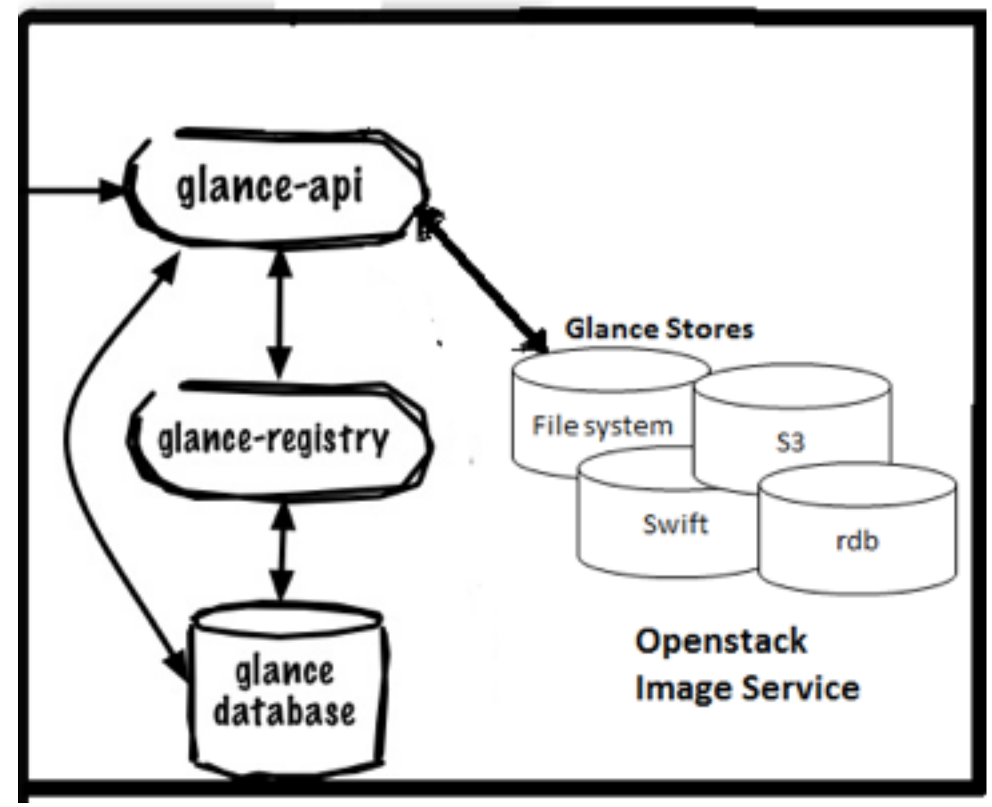


Image Service

The Image Service: Glance

- The primary objective of Glance is to publish a catalog of virtual machine images.



- Main components:
  - **glance-api**: accepts Image API calls for image discovery, retrieval and storage
  - **glance-registry**: stores, processes, and retrieves metadata for images
  - **storage backend** (filesystem, rbd, swift, s3, cinder, etc.)

Glance installation

1. Install the required packages
2. Create a database and a user “glance”
3. Edit the configuration files *glance-api.conf* and *glance-registry.con in /etc/glance/*
4. Create the database tables running the command:

   glance-manage db_sync

5. Restart the service
6. Create a keystone user named “glance” in the “service” tenant with admin role
7. Add the image service in the keystone catalog and define its endpoint
8. Verify the installation

Image creation

```
# glance help image-create
usage: glance image-create [--id <IMAGE_ID>] [--name <NAME>] [--store <STORE>]
                           [--disk-format <DISK_FORMAT>]
                           [--container-format <CONTAINER_FORMAT>]
                           [--owner <TENANT_ID>] [--size <SIZE>]
                           [--min-disk <DISK_GB>] [--min-ram <DISK_RAM>]
                           [--location <IMAGE_URL>] [--file <FILE>]
                           [--checksum <CHECKSUM>] [--copy-from <IMAGE_URL>]
                           [--is-public {True,False}]
                           [--is-protected {True,False}]
                           [--property <key=value>] [--human-readable]
                           [--progress]
```

- with option ***--location*** Glance simply tracks where that data resides. For example, if the image data is stored in swift, you could specify

  'swift://account:key@example.com/container/obj'.

- ***--property*** option allows to associate "core" properties and user-defined metadata to the image. Example of core properties:
  - architecture, kernel_id, ramdisk_id, os_distro, os_version, hw_disk_bus, hw_vif_model, etc.
- **Image Property Protection**: access to image meta properties may be configured using a Property Protections Configuration file.
- **Image membership**: authorize a tenant to access a private image

Image creation & contextualization

- The simplest way to create a virtual machine image is to use the **virt-manager** GUI, which is installable as the virt-manager package on both Fedora-based and Debian-based systems. This GUI has an embedded VNC client in it that will let you view and interact with the guest's graphical console.

- Install **cloud-init** (Linux) or **Cloudbase** (Windows) to fetch metadata (and user-data) from the metadata server or configuration drive

  - root-fs resize

  - ssh-key injection

- use virt-sysprep tools to clean your image before uploading onto Glance (e.g. remove files like persistent-net.rules and persistent-net-generator)

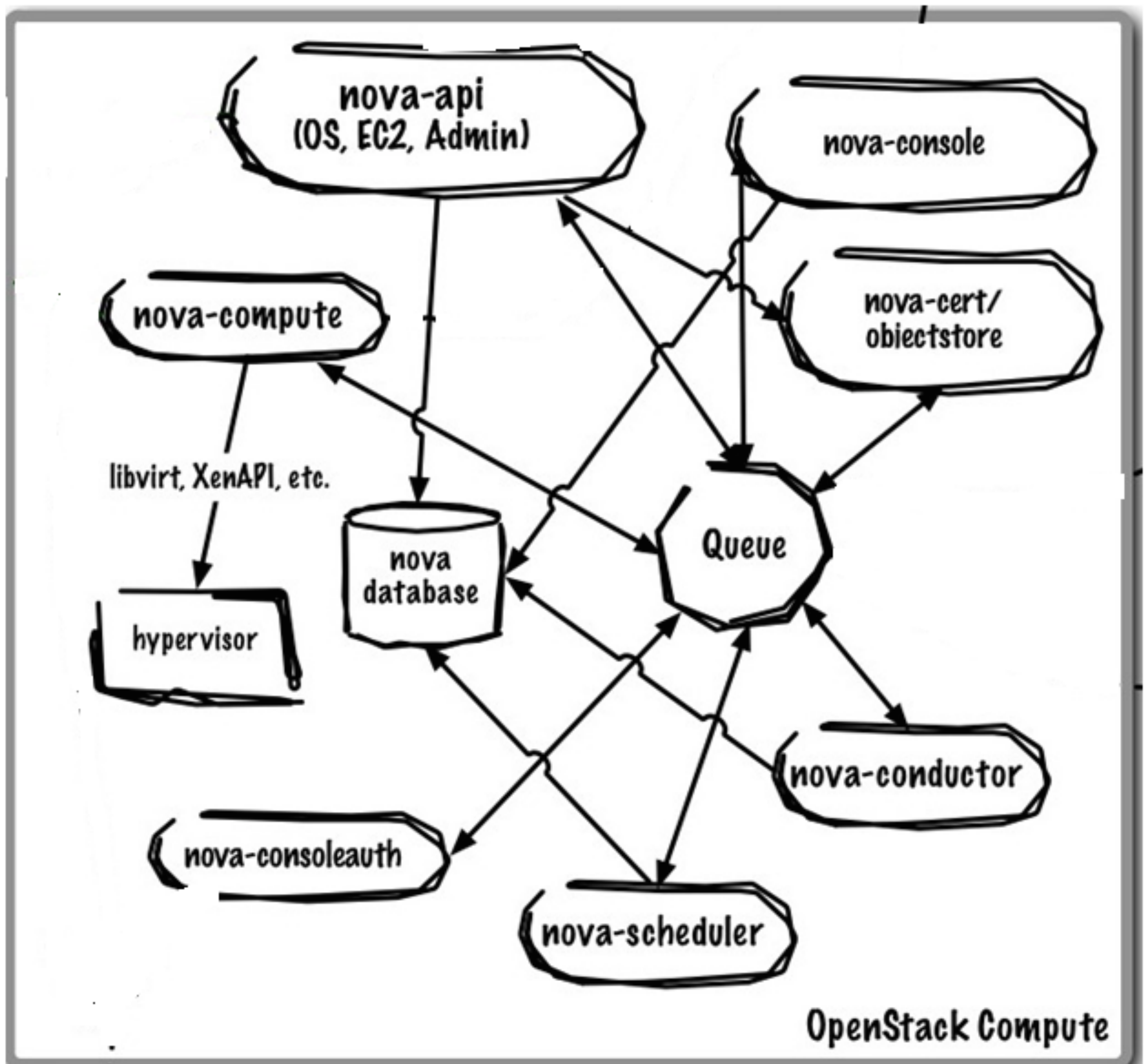
- Tool for automated image creation: Oz, VMBuilder, imagefactory

Image synchronization

- VMcatcher/VMcaster toolset
- **VMcaster** is a simple tool for publishing, managing and updating virtual ma- chines image lists which follows the HEPix image list specifications. All the image lists and metadata created by VMcaster are signed and trusted with a X.509 certificate. This provides a mechanism by which a virtual image can be checked for validity by any subscriber.
- **VMcatcher** utility allows image consumers to subscribe to VM image list generated by VMcaster. Using this utility users can select and download trusted images.

Compute Service

Nova architecture



The compute service: Nova

- The Compute service is a cloud computing fabric controller, which is the main part of a IaaS system.
- Main components:
    - **nova-api**: accepts and responds to end user compute API calls
    - **nova-scheduler**: determines on which compute server host it should run.
    - **nova-conductor**: mediates interactions between nova-compute and the database
    - **nova-novncproxy**, **nova-consoleauth** allow end users to access their virtual instances through a proxy
    - **nova-compute** manages virtualization (hypervisor drivers for KVM, Xen, VMware, Hyper-V, Docker, etc.)

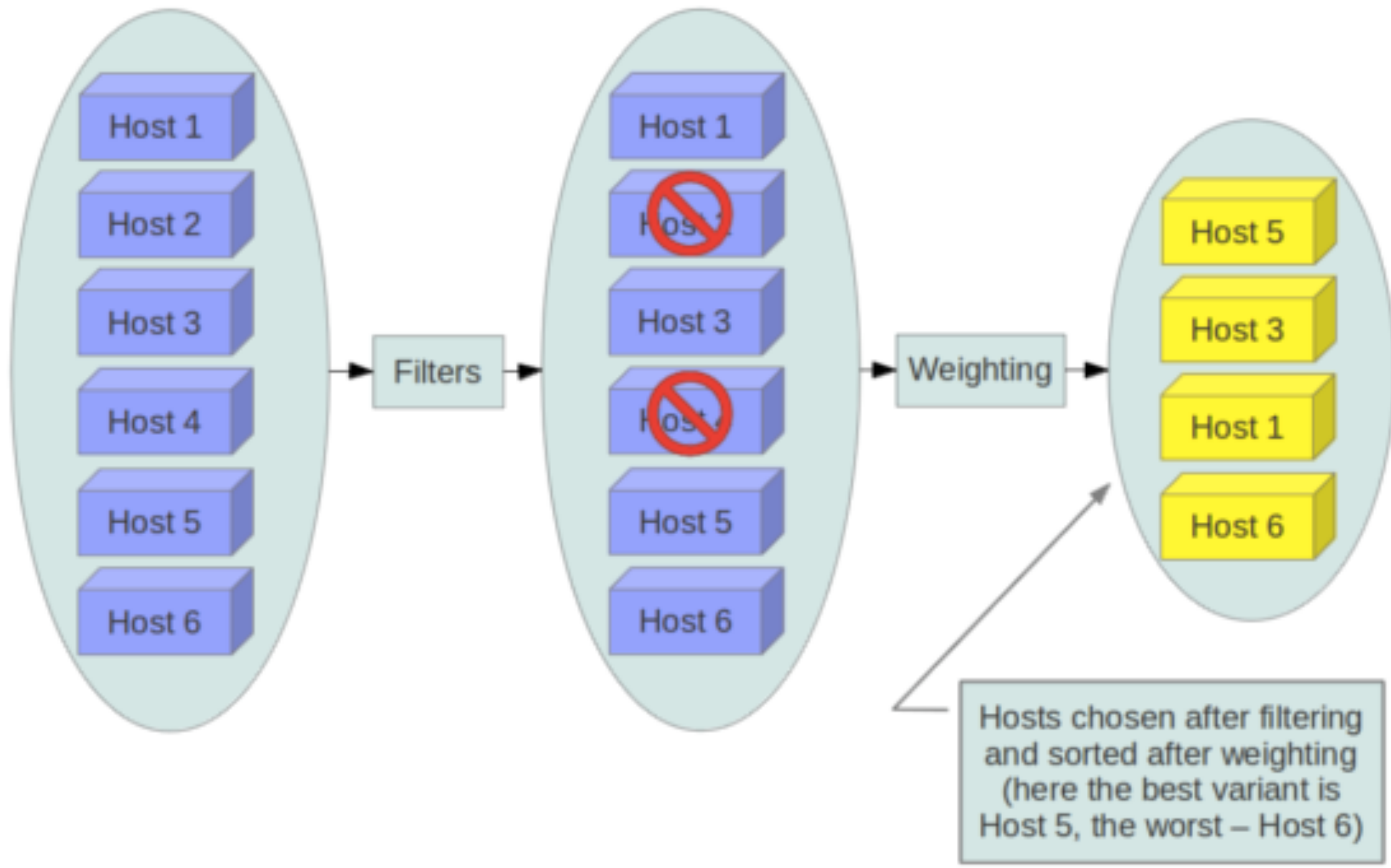
VM placement: nova scheduler

- Compute uses the nova-scheduler service to determine how to dispatch compute requests.

- All compute nodes periodically publish their status, resources available and hardware capabilities to nova-scheduler through the queue. Nova-scheduler then collects this data and uses it to make decisions when a request comes in.

- By default, the compute scheduler is configured as a filter scheduler

Filter Scheduler



- Default Filters: RetryFilter, AvailabilityZoneFilter, CapacityFilter, CapabilitiesFilter
- Useful filters: AggregateInstanceExtraSpecsFilter, AggregateCoreFilter, RamFilter, ImagePropertiesFilter

Segregating your cloud

- **Availability zones**
    - ❖ Logical separation within your nova deployment for physical isolation or redundancy.
- **Host aggregates**
    - ❖ To schedule a group of hosts with common features.
- A common use of host aggregates is to provide information for use with the nova-scheduler. For example, you might use a host aggregate to group a set of hosts that share specific flavors or images.

Nova installation

1. Install the required packages
2. Create a database and a user “nova”
3. Edit the configuration files */etc/nova/nova.conf*
4. Create the database tables running the command:

   nova-manage db sync
5. Create a keystone user named “nova” in the “service” tenant with admin role
6. Add the compute service in the keystone catalog and define its endpoint
7. Restart the service
8. Verify the installation

Live migration

- To enable live-migration:
  - share the instances folder (/var/lib/nova/instances) among the compute nodes

    OR

  - use RBD shared storage backend
- Requirements:
  - enable ssh login passwordless for nova user (check /var/lib/nova/.ssh folder)

Networking Service

The Networking Service: Neutron

- Neutron is a pluggable, scalable and API-driven system for managing networks and IP addresses in your cloud.

- The Neutron project is considered to be one of the most exciting projects – with great innovation around network virtualization and software-defined networking (SDN).

- The Networking service also provides an API to configure and manage a variety of network services ranging from L3 forwarding and NAT to load balancing, edge firewalls, and IPsec VPN.

Neutron components

- **neutron-server**: accepts and routes API requests to the appropriate OpenStack Networking plug-in for action.

- **plug-ins and agents**: plug and unplug ports, create networks or subnets, and provide IP addressing. OpenStack Networking ships with plug-ins and agents for Cisco virtual and physical switches, NEC OpenFlow products, Open vSwitch, Linux bridging, and the VMware NSX product.

    - The common agents are L3 (layer 3), DHCP (dynamic host IP addressing), and a plug-in agent.

- The **Modular Layer 2** (ML2) plugin is a framework allowing OpenStack Networking to simultaneously utilize the variety of layer 2 networking technologies found in complex real-world data centers. It currently works with the existing Open vSwitch, Linux bridge, and Hyper-V L2 agents as what ML2 define as 'MechanismDrivers', and is intended to replace and deprecate the monolithic plugins associated with those.

    - It currently includes drivers for the **local**, **flat**, **VLAN**, **GRE** and **VXLAN** network types

Neutron installation

1. Install the required packages: neutron-server, neutron-plugin-ml2
2. Create a database and a user “neutron”
3. Edit the configuration files */etc/neutron/neutron.conf*
4. Configure the ML2 plugin (/etc/neutron/plugins/ml2/ml2_conf.ini)
5. Create the database tables running the command:

    neutron-manage db sync

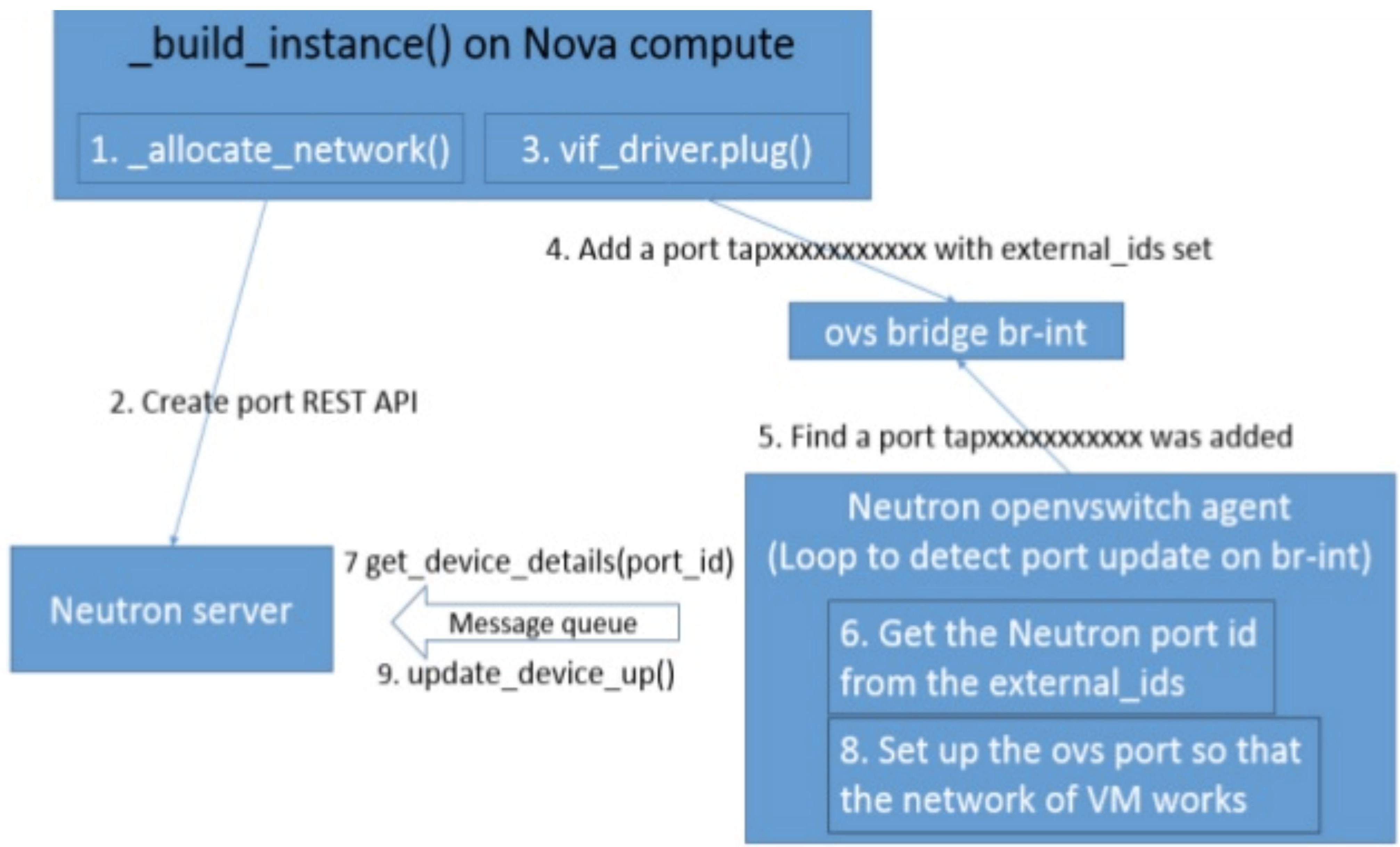
6. Create a keystone user named “neutron” in the “service” tenant with admin role
7. Add the networking service in the keystone catalog and define its endpoint
8. Restart the service
9. Verify the installation

Some neutron options in nova.conf

```
network_api_class = nova.network.neutronv2.api.API
neutron_url = http://network:9696
neutron_auth_strategy = keystone
neutron_admin_tenant_name = service
neutron_admin_username = neutron
neutron_admin_password = xxxxx
neutron_admin_auth_url = http://auth-node:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSInterfaceDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver
security_group_api = neutron
```

- These parameters are critical for the interaction between nova and neutron

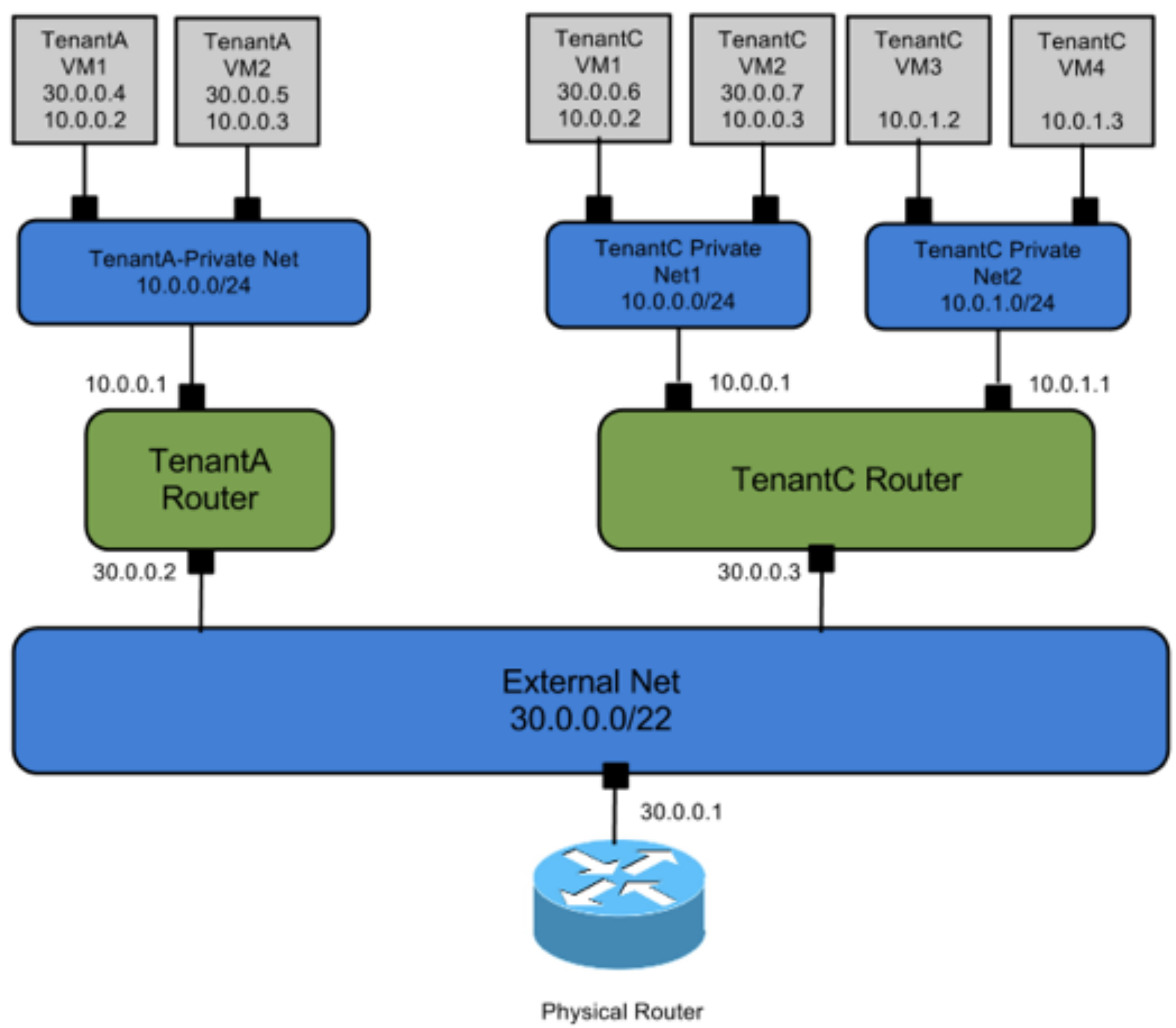
Interaction to boot VM



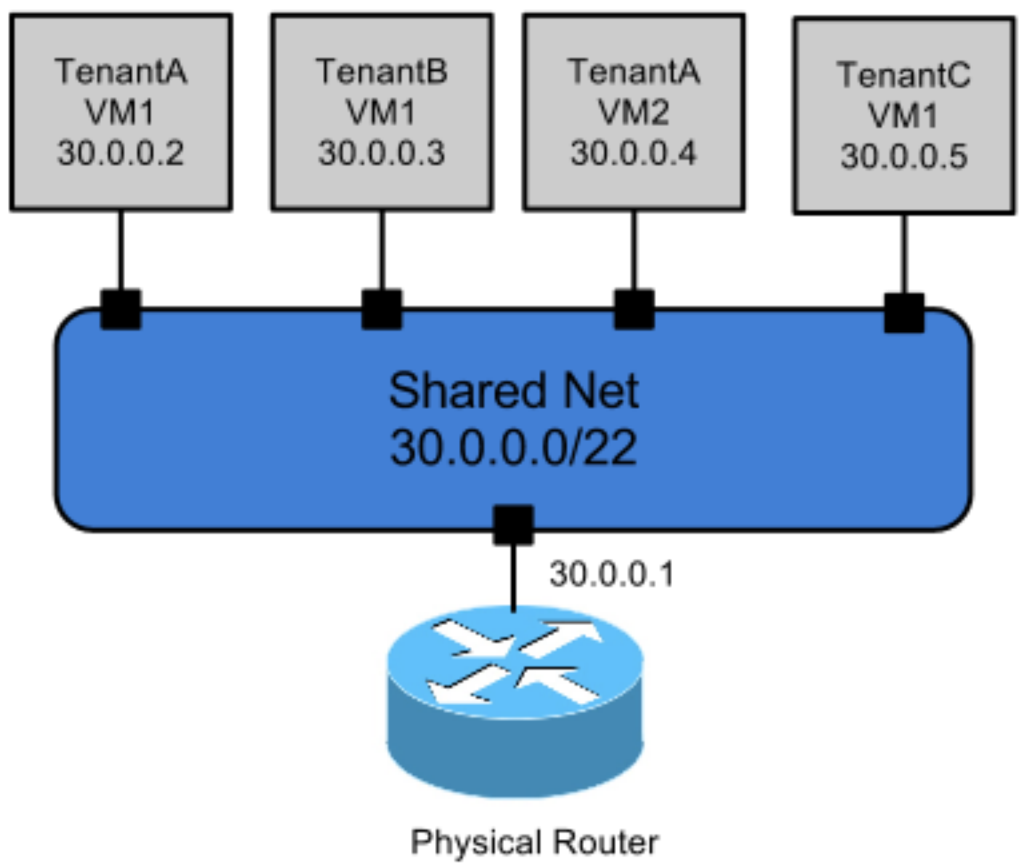
Scenarios

1. Single Flat Network
2. Multiple Flat Networks
3. Mixed Flat and Private Networks
4. Provider Router
5. Per-Tenant Router
6. Multiple External Networks
7. Routed Networks without NAT

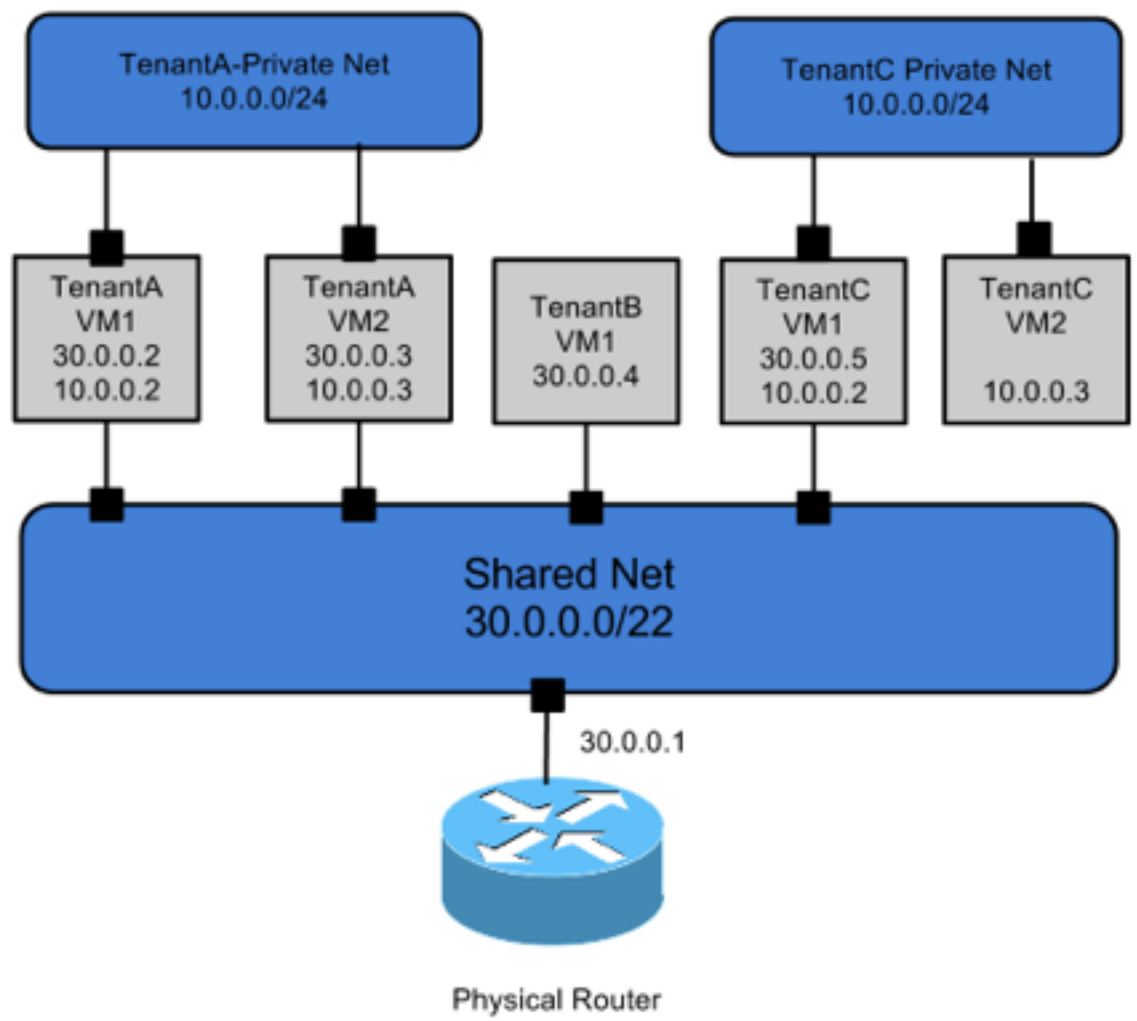
Per tenant-routers with private networks



Single Flat Network



Mixed flat & private network



Created VMs can have NICs on any of the shared or private networks that the tenant owns. This enables the creation of multi-tier topologies that use VMs with multiple NICs. It also enables a VM to act as a gateway so that it can provide services such as routing, NAT, and load balancing.

Neutron extension: allowed-address-pairs

- iptables rules are programmed by Neutron on the compute node that hosts the instance that permits traffic from the instance matching the IP and MAC address of the associated neutron port (to prevent IP and MAC spoofing)
- This is a problem if the instance is used to route traffic between multiple networks.
- Solution:

```
neutron port-update <port_id> --allowed-
address-pairs type=dict list=true
ip_address=<ip_address/CIDR>
```

Dashboard

Openstack dashboard: Horizon

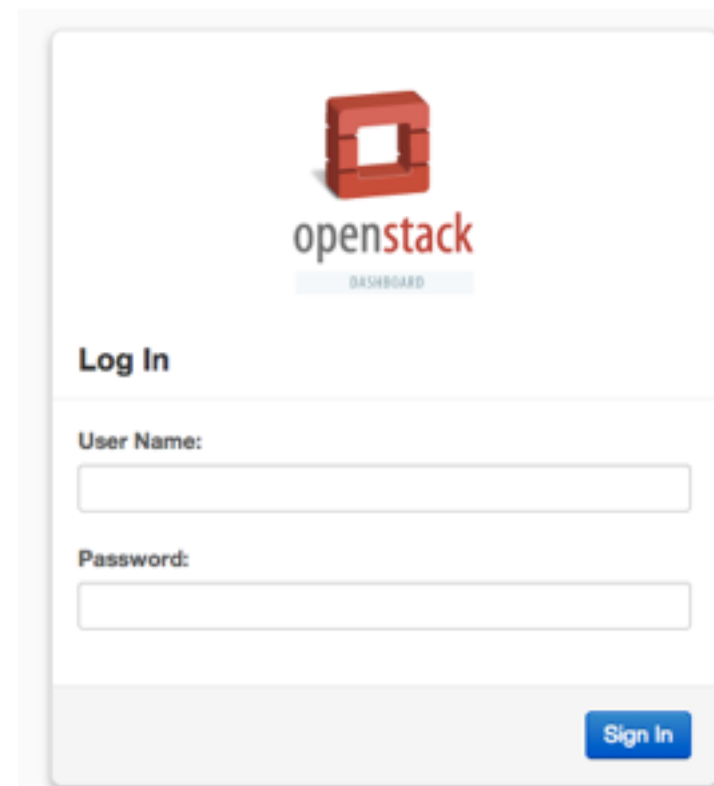
- Horizon provides a web based user interface to OpenStack services including Nova, Swift, Keystone, Neutron, etc.

- The Service Catalog returned by the Identity Service after a user has successfully authenticated determines the dashboards and panels that will be available within the OpenStack Dashboard.

- Horizon ships with three central dashboards, “Project”, “Admin”, and “Settings”.

Horizon installation

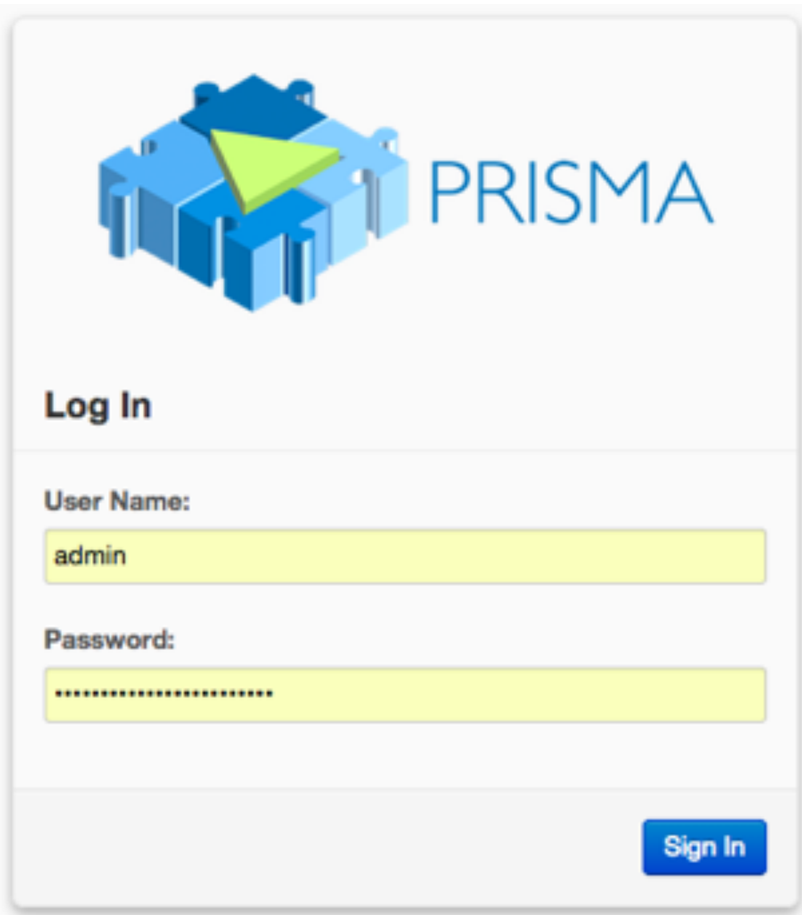
- Install the required packages
- Edit the configuration file /etc/openstack-dashboard/local_settings.py
    - Configure Your Identity Service Host
        - OPENSTACK_HOST
        - OPENSTACK_KEYSTONE_URL
    - Configure Your Session Storage
        - memcached (in our tutorial)
- Restart apache2 and memcached

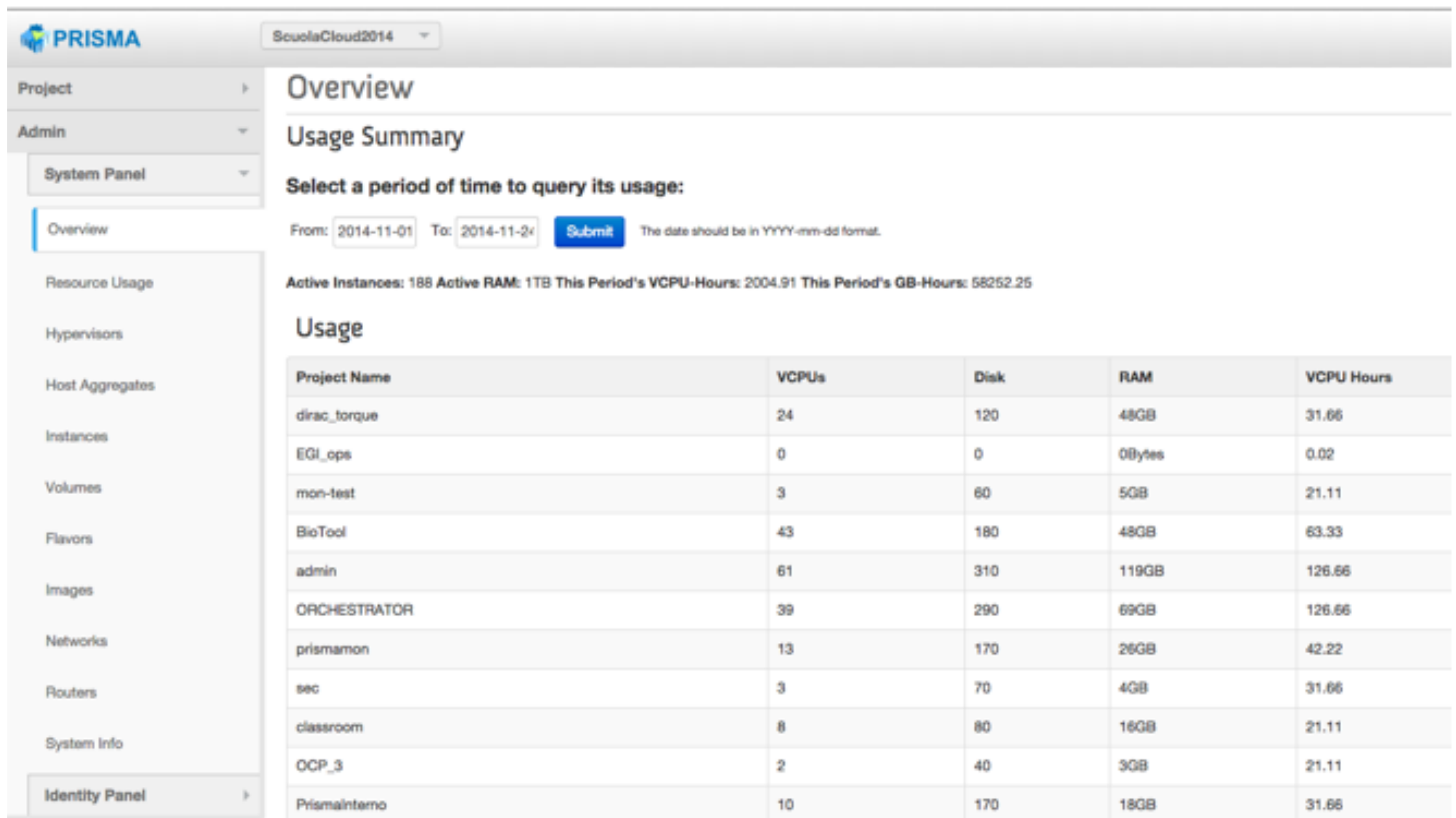


Note: You can secure your dashboard enabling the SSL support.

Moreover, you can customize your dashboard changing the logo, the site title, modifying the existing dashboards and panels.

PRISMA dashboard



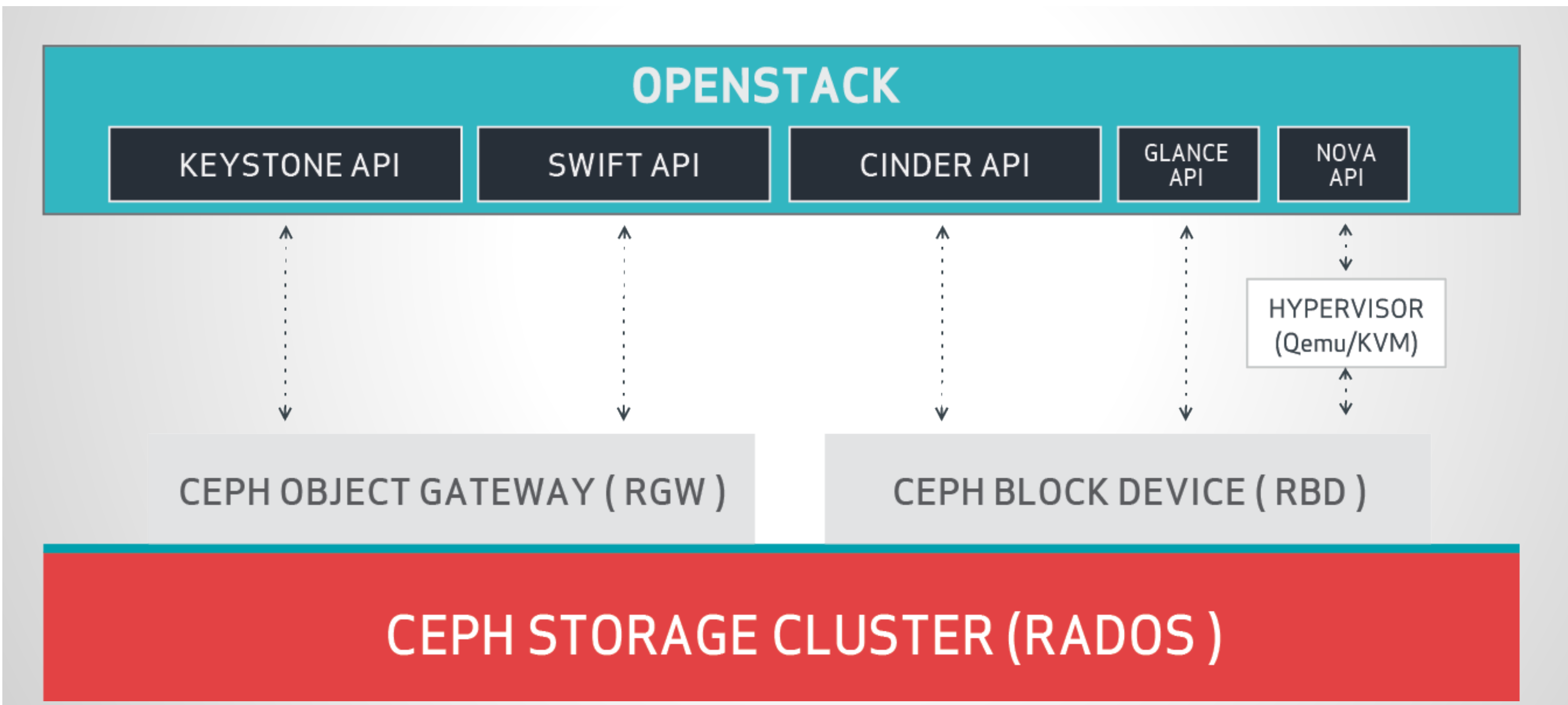


Additional services & Advanced configurations

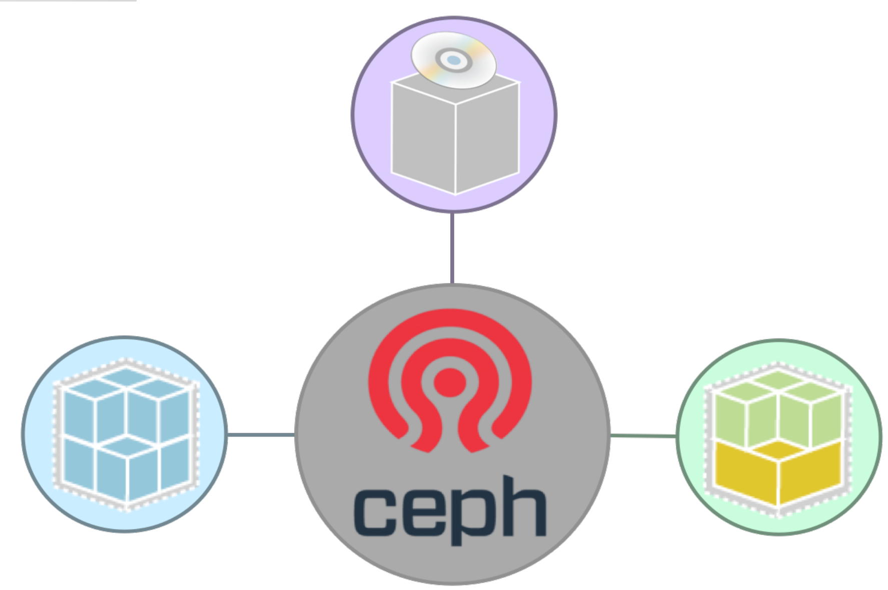
Advanced services

- Storage
  - Block storage: cinder
  - Object storage: swift
- Neutron Avanced configuration
  - VPNaaS
  - LBaaS
- Telemetry
- Zabbix
- Heat

Ceph: de-facto storage backend for Openstack



- Storage consolidation:
  - Glance, Cinder, Nova integration
  - Swift (RADOS GW)



Configure RBD backend for nova-compute

- Configure Libvirt

```
uuidgen
d38c68b3-53d3-4a4f-8f36-10d3b37ca4eb

cat > secret.xml <<EOF
<secret ephemeral='no' private='no'>
  <uuid> d38c68b3-53d3-4a4f-8f36-10d3b37ca4eb</uuid>
  <usage type='ceph'>
    <name>client.cinder secret</name>
  </usage>
</secret>
EOF
sudo virsh secret-define --file secret.xml
Secret d38c68b3-53d3-4a4f-8f36-10d3b37ca4eb created
sudo virsh secret-set-value --secret d38c68b3-53d3-4a4f-8f36-10d3b37ca4eb --base64 $(cat
client.cinder.key) && rm client.cinder.key secret.xml
```

- Edit /etc/nova/nova.conf, add:

```
[libvirt]
images_type = rbd
images_rbd_pool = vms
images_rbd_ceph_conf = /etc/ceph/ceph.conf
rbd_user = cinder
rbd_secret_uuid = d38c68b3-53d3-4a4f-8f36-10d3b37ca4eb
```

Cinder: rbd driver

```
[ceph]
volume_driver = cinder.volume.drivers.rbd.RBDDriver
rbd_pool = volumes
glance_api_version = 2
rbd_user = cinder
rbd_secret_uuid = 925560f4-ae0d-40a6-805f-dc628d63cef8
volume_backend_name=CEPH
rbd_ceph_conf=/etc/ceph/ceph.conf
rbd_flatten_volume_from_snapshot=false
rbd_max_clone_depth=5
```

cinder.conf

```
    <disk type='network' device='disk'>
      <driver name='qemu' type='raw' cache='none'/>
      <auth username='cinder'>
        <secret type='ceph' uuid='925560f4-ae0d-40a6-805f-dc628d63cef8'/>
      </auth>
      <source protocol='rbd' name='volumes/volume-66ae13fe-2d3d-414a-809e-3ae295697497'>
        <host name='90.147.75.247' port='6789'/>
        <host name='90.147.75.248' port='6789'/>
        <host name='90.147.75.249' port='6789'/>
      </source>
      <target dev='vdb' bus='virtio'/>
      <serial>66ae13fe-2d3d-414a-809e-3ae295697497</serial>
      <alias name='virtio-disk1'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x06' function='0x0'/>
    </disk>
```

VM definition file (virsh dumpxml <domain>)