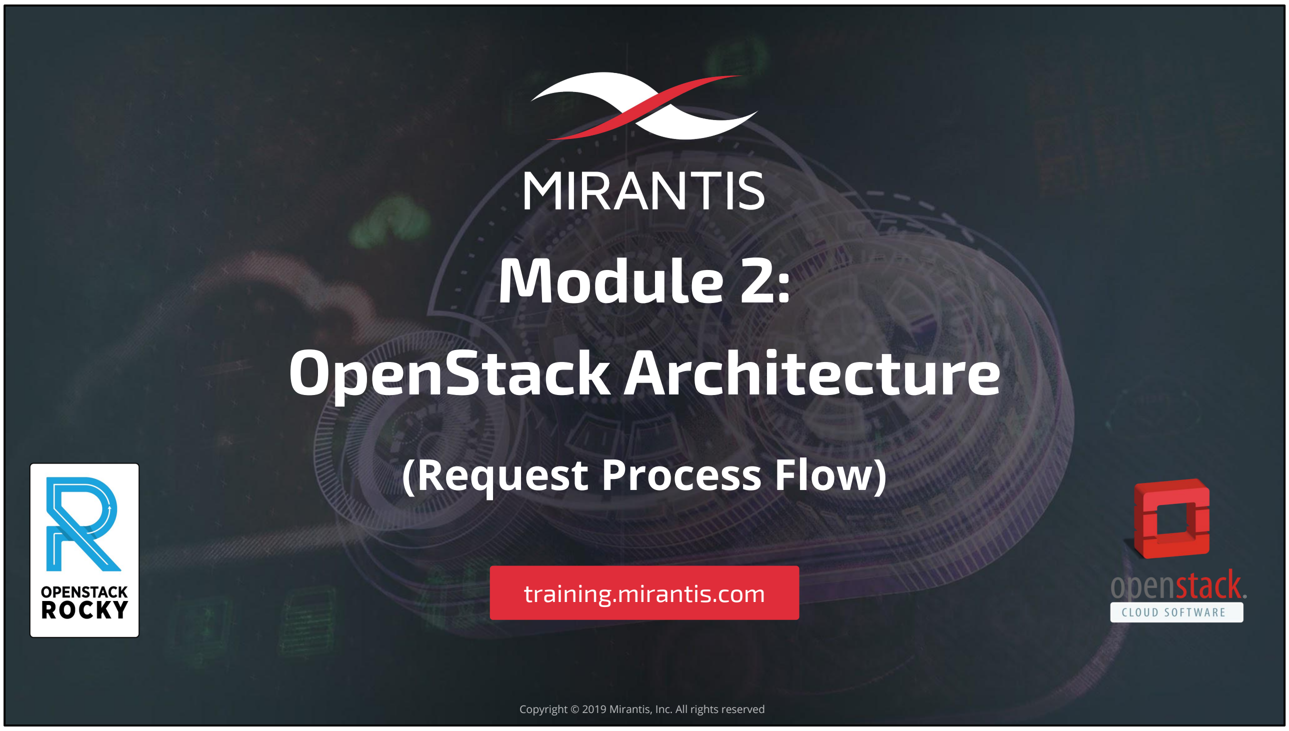


MIRANTIS

Module 2: OpenStack Architecture

(Request Process Flow)

training.mirantis.com

OPENSTACK ROCKY

openstack. CLOUD SOFTWARE

This module uses an example process flow (create a virtual machine instance) to introduce the most popular OpenStack components and their architecture.

Objectives

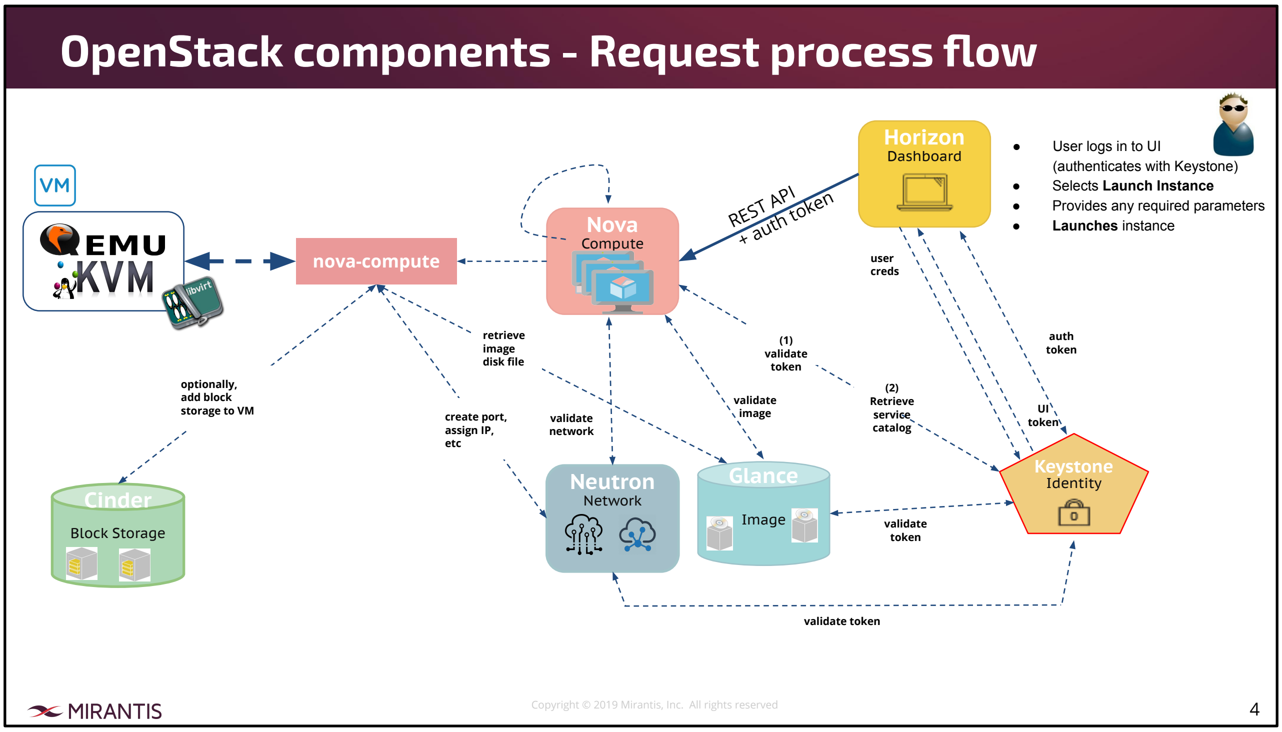
At the end of this presentation, you should be able to:

- Explain the flow to deploy an instance and the role of each component
  - Horizon (Dashboard UI)
  - Keystone (Identity service)
  - Nova (Compute service)
  - Neutron (Network service)
  - Glance (Image service)
  - Cinder (Block Storage service)

Request Process Flow

Deploy a VM instance

MIRANTIS



OpenStack components - Request process flow

- User logs in to UI (authenticates with Keystone)
- Selects **Launch Instance**
- Provides any required parameters
- **Launches** instance

MIRANTIS

4

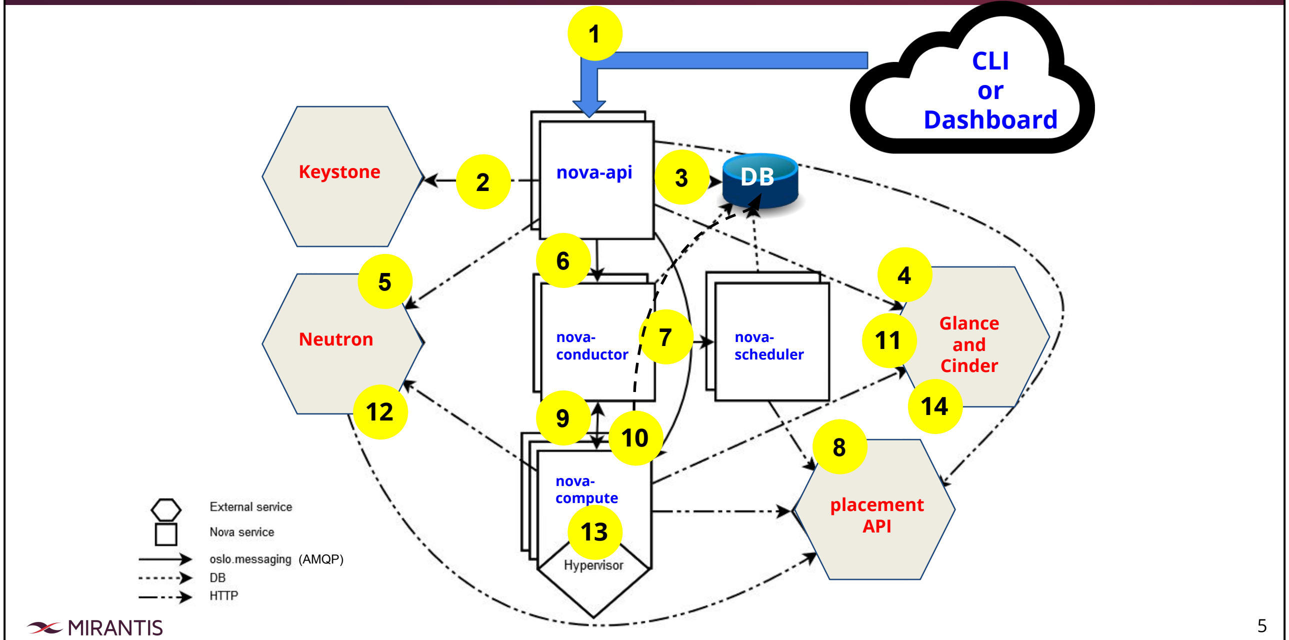
OpenStack components are designed to work together. Suppose you request to deploy a virtual machine instance could come from the OpenStack CLI client (**OpenStack server create**) or the Horizon Dashboard UI.

- The request is sent to Nova (Compute service). Nova will interface with other OpenStack components to fulfill the request, using hypervisor-specific APIs to deploy the instance.
- Nova uses images stored by Glance (Image service).
- The images might be stored in Swift, for example. Swift is an object store, using an http server that replicates s3 api and allows storage and retrieval of objects (images, in this case)
- Neutron (Network service) provides the virtual networking, such as an IP address, to deploy the instance.
- Optionally, your might include additional (non-root) disks (volumes) to attach to the instance using Cinder (Volume service).
- Keystone (Identity service) is used throughout the process, providing authentication and authorization functions.
- Additionally, Ceilometer might be used to capture metrics, such as CPU utilization, by polling Nova, Glance, Neutron, or subscribing to rpc events.
- Optionally, Heat can be used to orchestrate (automate) the entire lifecycle of the VM

Each component utilizes

- Database: sql database for data storage, such as MySQL
- Message queue service: For communications between various OpenStack processes, such as RabbitMQ

Request process flow from Nova perspective



MIRANTIS

When the user logs in to the dashboard (or CLI), their user credentials are authenticated with the Identity Service via REST API, receiving an **auth-token** which is *used for the UI session*. When the user submits a request, such as launch instance, **a new auth-token** *is assigned for the request* and used for the lifecycle of the request, including inter-component communications.

1. The dashboard or CLI converts the **launch instance** request to a REST API POST request, obtains an **auth-token**, and sends the request to nova-api The **auth-token** is part of the request and will be used throughout the lifecycle of the request.
2. nova-api receives the request and sends a request to the Identity Service for validation of the **auth-token** and access permission. The Identity Service validates the **auth-token** and sends updated authentication headers with roles and permissions.
3. nova-api checks for conflicts with nova-database and then creates initial database entry for the new instance.
    - nova-api also validates the flavor and the compute resource quotas
4. nova-api sends request to Glance (glance-api) to validate the image
5. nova-api sends request to Neutron (neutron-server) to validate network, port, and network quotas.
6. nova-api sends the rpc.call request to nova-conductor expecting to get updated instance entry with host ID specified.
7. nova-conductor receives and processes the message, sending it to nova-scheduler. nova-scheduler picks up the request from the queue.
8. nova-scheduler interacts with the placement API to find an appropriate host via filtering and weighing. nova-scheduler returns the updated instance entry with the appropriate host ID (of chosen hypervisor) after filtering and weighing.
9. nova-conductor sends the rpc.cast request to nova-compute for launching an instance on the appropriate host.

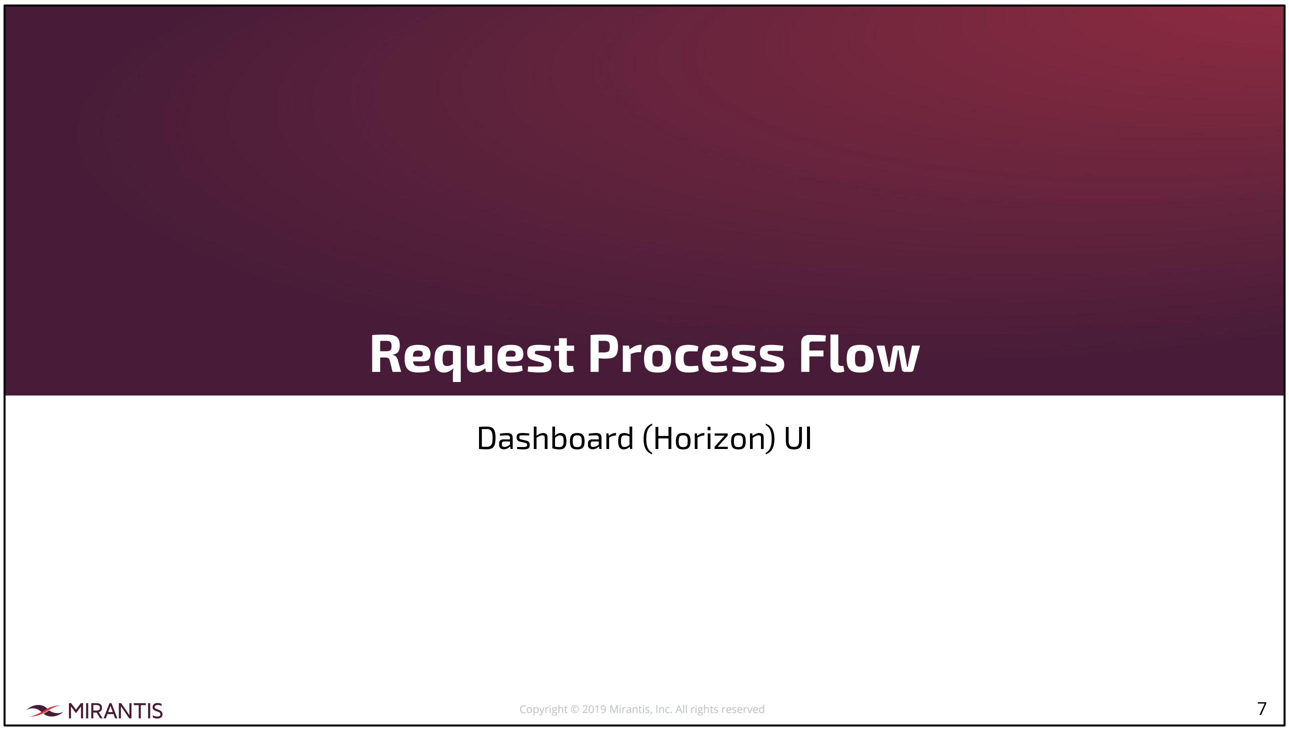
10. nova-compute picks up the request from the queue. nova-compute sends the rpc.call request to nova-conductor to fetch the instance information such as host ID and flavor (RAM, CPU, Disk) from the database.
11. nova-compute sends a REST call to glance-api, using the Image ID, to retrieve the Image URI from the Image Service, and loads the image from the image storage.
    - nova-compute also retrieves the image metadata.
12. nova-compute performs a REST call to the Neutron API (neutron-server) to allocate and configure the network so that the instance gets a port with an IP address.
    - neutron-server sends request to DHCP server for IP address, for example.
    - nova-compute then retrieves the network info.
13. nova-compute generates data for the hypervisor driver and executes the request on the hypervisor (via libvirt or specific hypervisor API) ro create an instance.
14. If a volume is to be attached, nova-compute performs a REST call to the cinder-api to attach the volume to the instance. nova-compute then retrieves the block storage info.

Important Note: As each component, such as glance-api for Glance, receives a request, it validates the **auth-token** with Keystone before taking any action on the request. If the token is not valid, the request is rejected. The token validation occurs at each step above and is not shown or discussed in the flow.

VM provisioning request

- Provisioning a VM instance is the most common request in an IaaS environment
- Also, one of the most complex
- Involves interaction between many OpenStack components

- This presentation will use this flow to discuss Horizon, Keystone, Nova, Glance, Neutron, and Cinder

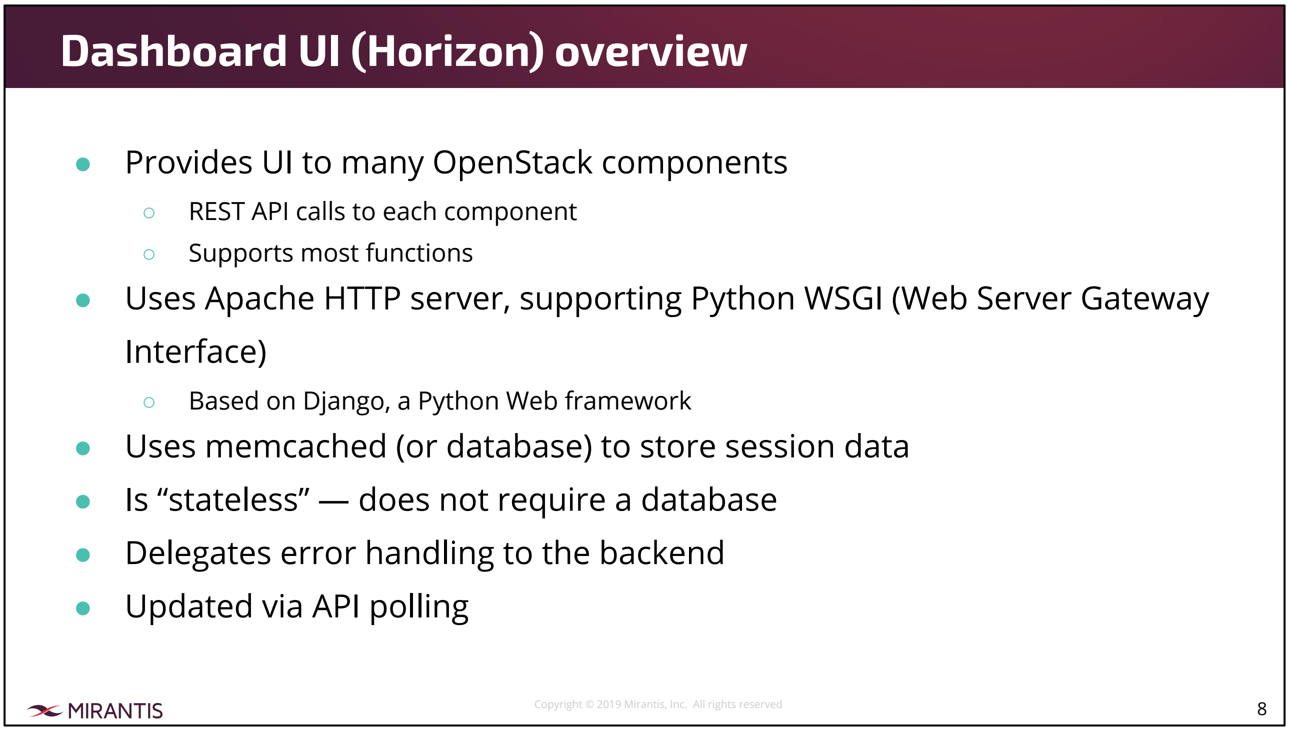


Request Process Flow

Dashboard (Horizon) UI

MIRANTIS

The Horizon dashboard provides an administrative user interface to functions for the various OpenStack components: Keystone, Nova, Glance, Swift, Neutron, Cinder, and so on. As a result, users can use the Dashboard to create volumes, images, users, virtual machine instances, networks, and so on.
Since it is easier to use than the CLI, most customers are running the Horizon Dashboard UI.



Dashboard UI (Horizon) overview

- Provides UI to many OpenStack components
  - REST API calls to each component
  - Supports most functions
- Uses Apache HTTP server, supporting Python WSGI (Web Server Gateway Interface)
  - Based on Django, a Python Web framework
- Uses memcached (or database) to store session data
- Is “stateless” — does not require a database
- Delegates error handling to the backend
- Updated via API polling

MIRANTIS

Very simple web application, "Stateless“, no DB, gets state from polling the Nova API, for example.

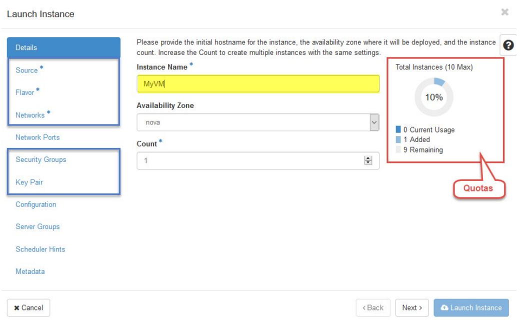
- Doesn’t inherently know whether or not an action can be performed – error handling is delegated to backend
- Can use memcached (or database) to store sessions. Using memcached is not recommended for production use or serious development work.
- Gets updated via api polling, updating every 5 seconds

Horizon uses standard Django Cache Framework:
https://docs.djangoproject.com/en/dev/topics/cache/?from=olddocs#the-per-site-cache
Here is an article on Django sessions storing and the differences between cookie, memcache and FS (DB):
https://docs.djangoproject.com/en/dev/topics/http/sessions/
Also, as a comparison, AWS also used Django Cache Framework. here is something that makes me personally understand a little better.
http://www.scribd.com/doc/54883641/Scaling-Django-Apps-With-Amazon-AWS

Dashboard UI (Horizon)

- Login to the Horizon Dashboard UI
  - **Domain + user name + password**
  - User is authenticated by Keystone and assigned an **auth-token** for the UI session
- Specify parameters for VM
  - Name
  - Flavor (CPU, RAM, disk)
  - Image (OS type)
  - Network
  - Optionally:
    - SSH keys
    - Cinder volumes to attach
    - Security groups
- Click **Launch Instance**
- Creates HTTP POST request to backend
  - Keystone assigns new **auth-token** for the request
  - Request routed to **nova-api** with **auth-token**
  - Request process flow begins on next slide

1



Please provide the initial hostname for the instance, the availability zone where it will be deployed, and the instance count. Increase the Count to create multiple instances with the same settings.

Instance Name *: MyVM

Availability Zone: nova

Count *: 1

■ 0 Current Usage
■ 1 Added
■ 9 Remaining

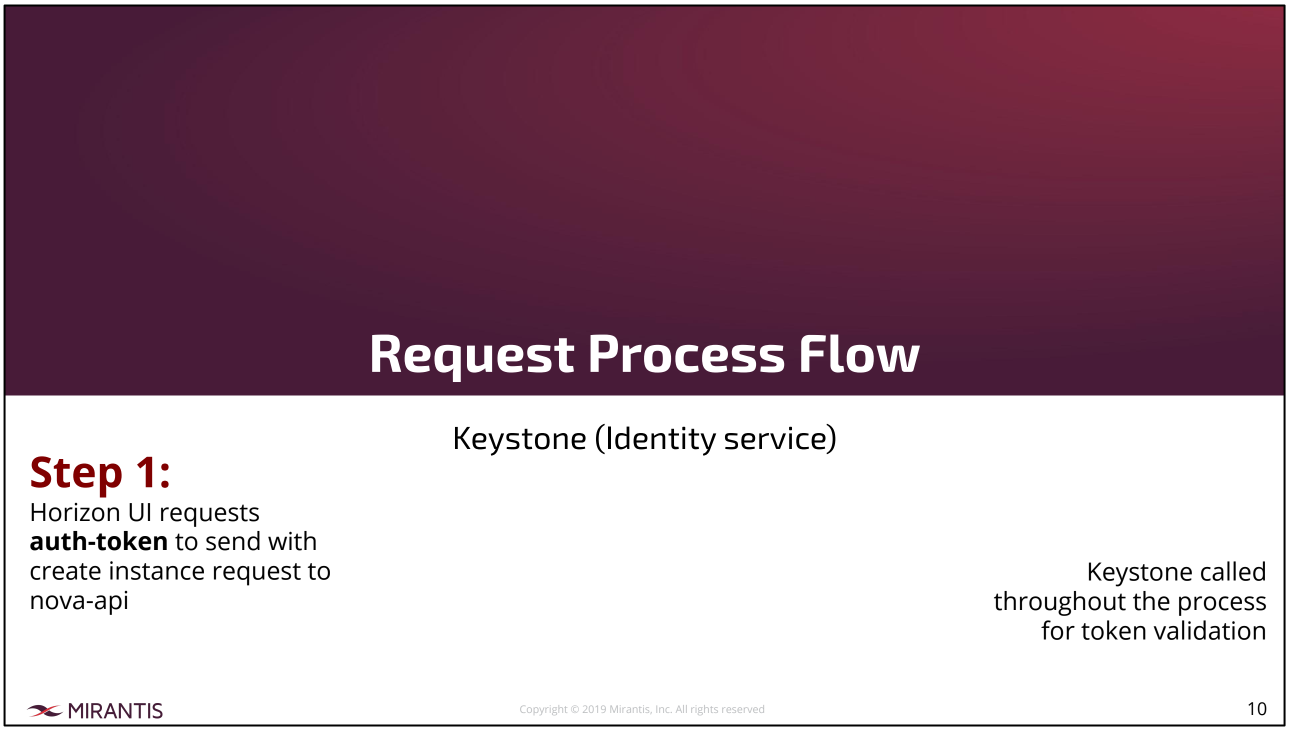
MIRANTIS

When a user logs in to the Dashboard UI, they must provide a domain name, user name, and password. Horizon sends a REST request to Keystone to authenticate the user credentials. Once authenticated, the user is assigned an **auth-token** that is used for the UI session. Another **auth-token** is created by Keystone for every UI request: create image, launch instance, stop server, add network, and so on.

Each user is logged in to their default project (if they are a member of multiple projects). To change the project, you can use the pull-down menu located in the navigation bar.

User access to functions is controlled by their role. By default there are two roles defined in Keystone: admin and member. The admin role is intended to be the cloud administrator. Therefore, you might choose to create a domain admin role and use it instead of the cloud admin role. In that case, you might also choose to assign the cloud admin role to a very limited group of users.

Users, roles, projects, and domains are discussed later in the Keystone lecture.



Request Process Flow

Keystone (Identity service)

Step 1:
Horizon UI requests **auth-token** to send with create instance request to nova-api

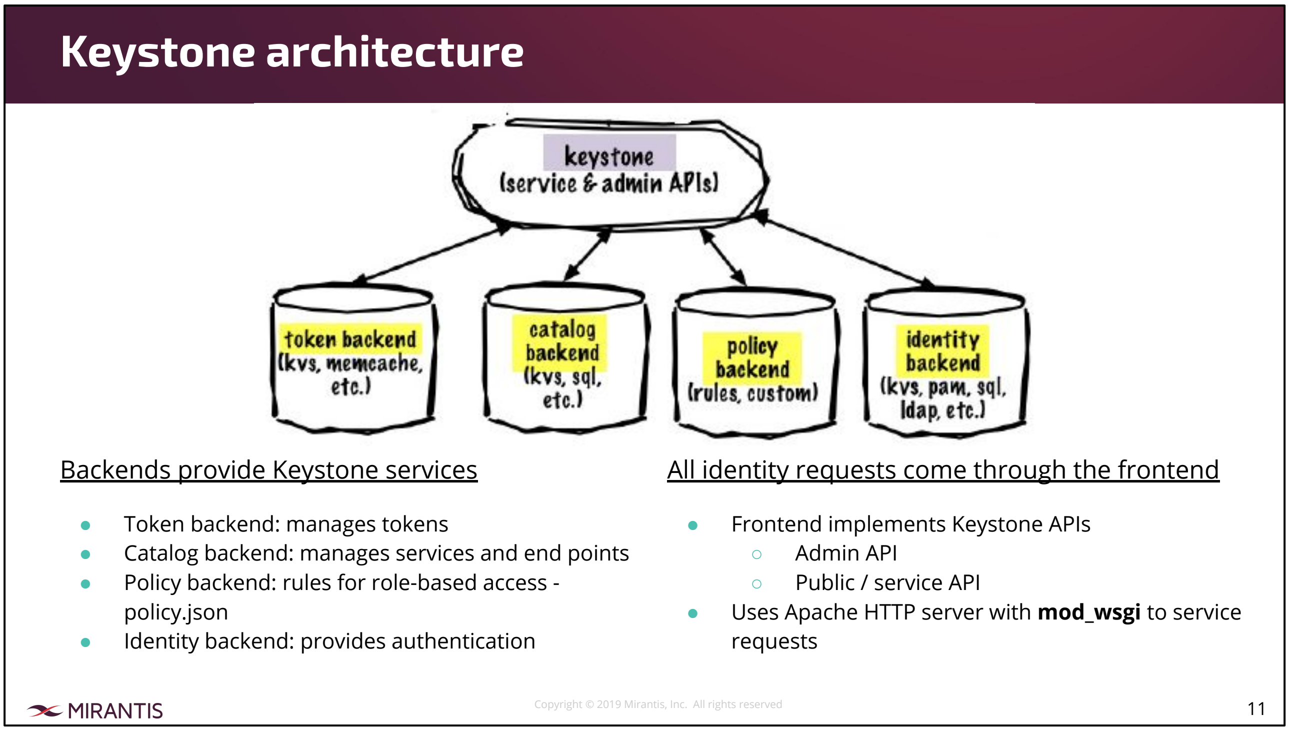
Keystone called throughout the process for token validation

MIRANTIS

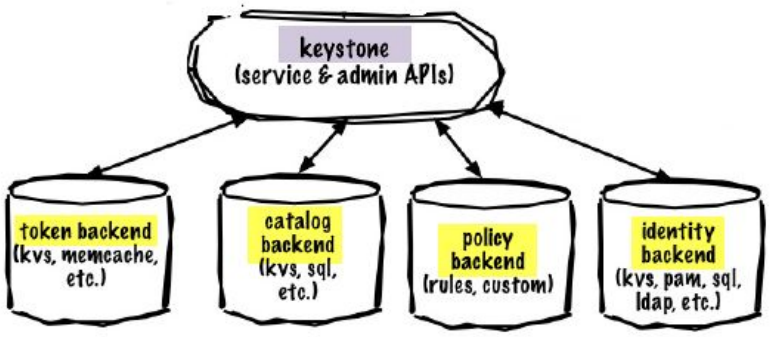
Keystone provides 4 primary functions:

- User authentication - Standard user name and password credential validation.
- Token management - Once the user is authenticated, they are assigned a token
- User authorization – Access to functions based on a user role.
- Service catalog - Contains *endpoint definitions* for each OpenStack service (Keystone, Nova, Glance, and so on). The endpoint defines URLs to access each specific component.

Keystone is used throughout the process flow to validate the auth-token at each step and to provide the service catalog. For example, when nova-api needs to send a request to the glance-api, it uses the entry for Glance in the service catalog as the target of the request.



Keystone architecture


keystone
(service & admin APIs)
token backend (kvs, memcache, etc.)
catalog backend (kvs, sql, etc.)
policy backend (rules, custom)
identity backend (kvs, pam, sql, ldap, etc.)

Backends provide Keystone services

- Token backend: manages tokens
- Catalog backend: manages services and end points
- Policy backend: rules for role-based access - policy.json
- Identity backend: provides authentication

All identity requests come through the frontend

- Frontend implements Keystone APIs
    - Admin API
    - Public / service API
- Uses Apache HTTP server with **mod_wsgi** to service requests

MIRANTIS

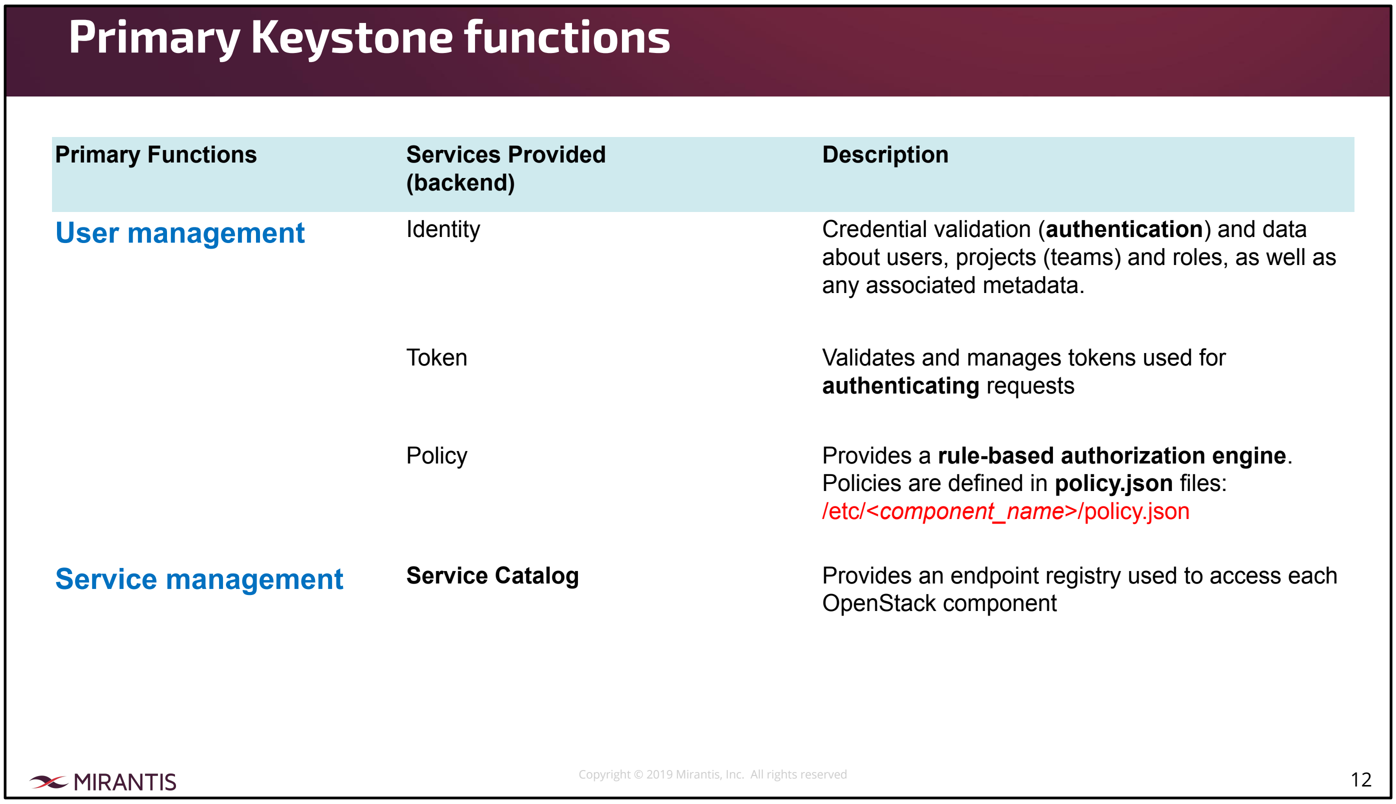
The Keystone **token backend** validates and manages tokens used for authenticating requests once a user credentials have been verified.

The Keystone **catalog backend** provides access to the service catalog. Each OpenStack component is registered as one or more services, each with a series of end point definitions (URLs) that control access to the service. For example, the nova-conductor service will query the service catalog for the URL of nova-compute to route a request to the nova-compute service.

The Keystone **identity backend** (database) can be implemented as one of the following:

- kvs: key-value store
- pam: pluggable authentication module
- ldap: lightweight directory access protocol. For example, might be Microsoft Active Directory (MSAD).
- sql: database (for example, MySQL). This is the most common (and default) choice.

The Keystone **policy backend** provides a rule-based authorization engine with policies defined for each component in ***/etc/component_name/*policy.json**



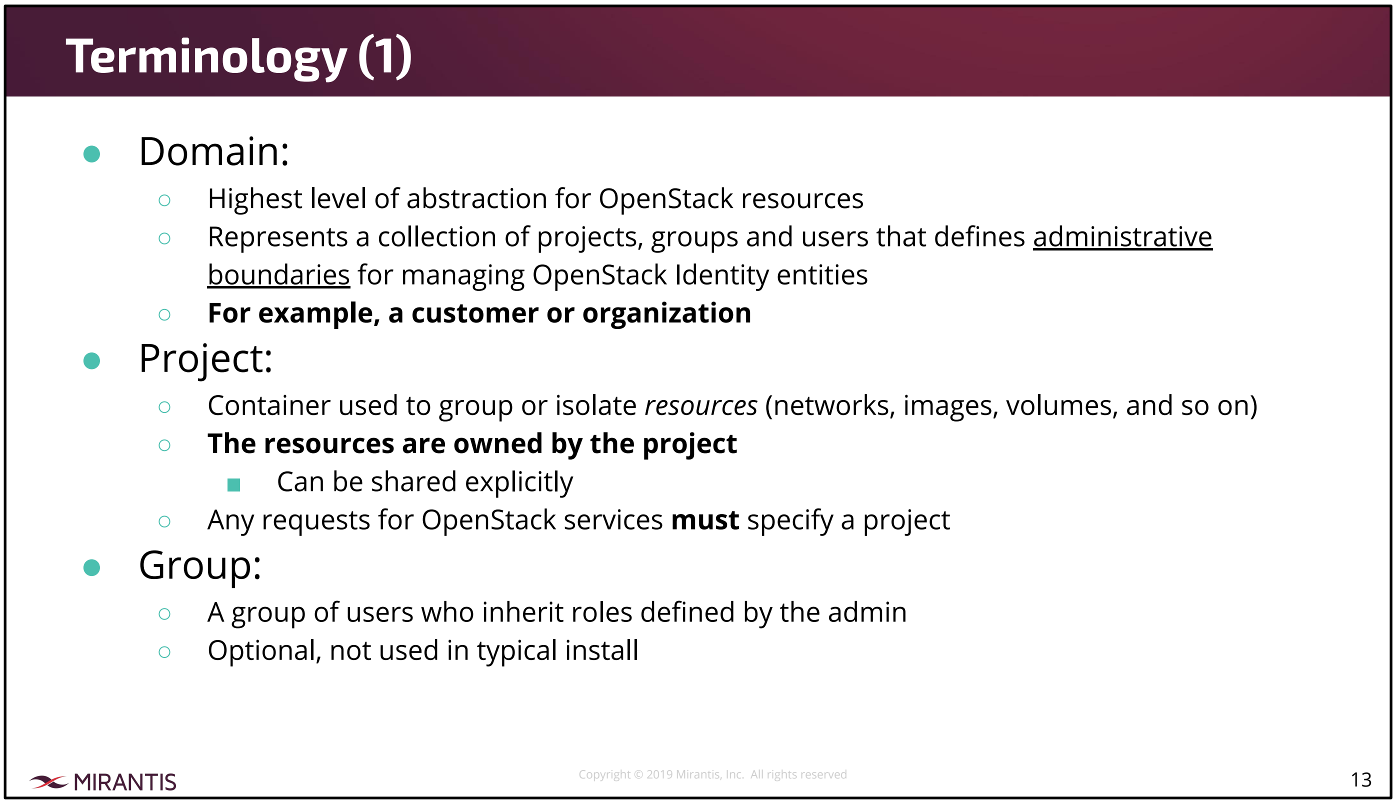
Primary Keystone functions

Primary Functions	Services Provided (backend)	Description
User management	Identity	Credential validation (**authentication**) and data about users, projects (teams) and roles, as well as any associated metadata.
	Token	Validates and manages tokens used for **authenticating** requests
	Policy	Provides a **rule-based authorization engine**. Policies are defined in **policy.json** files: /etc/<*component_name*>/policy.json
Service management	**Service Catalog**	Provides an endpoint registry used to access each OpenStack component

MIRANTIS

Keystone provides functions for authentication (credentials and tokens), authorization (rule-based access), and a service catalog:

- Identity: Traditional credential validation (user ID and password), also known as authentication.
  - By default, the MySQL database is used to store user credentials for authentication.
  - Keystone supports non-Keystone Lightweight Directory Access Protocol (LDAP) products. Many larger enterprises prefer to use an LDAP product, such as Microsoft Active Directory (MSAD).
- Token: Verifies and administers tokens used for authenticating requests. Once verified, users are assigned an internal token (X-AUTH-TOKEN) for their requests. Keystone manages these tokens,
- Policy: Exposes a *rule-based authorization engine*, supporting rules to access components and commands based on the *role of the user*. By default, Keystone supports 2 roles: **admin** and **member**. Custom roles can also be created, such as a team administrator role. Access to components, commands, and functions is based on a **policy.json** file provided by each component, under ***/etc/component_name***.
- (Service) Catalog: When you configure each component you define a **service** and **multiple endpoints**. Each endpoint defines a URL to access the component. This service and endpoint information is stored in the service catalog. The service catalog is sometimes called a *service registry*.

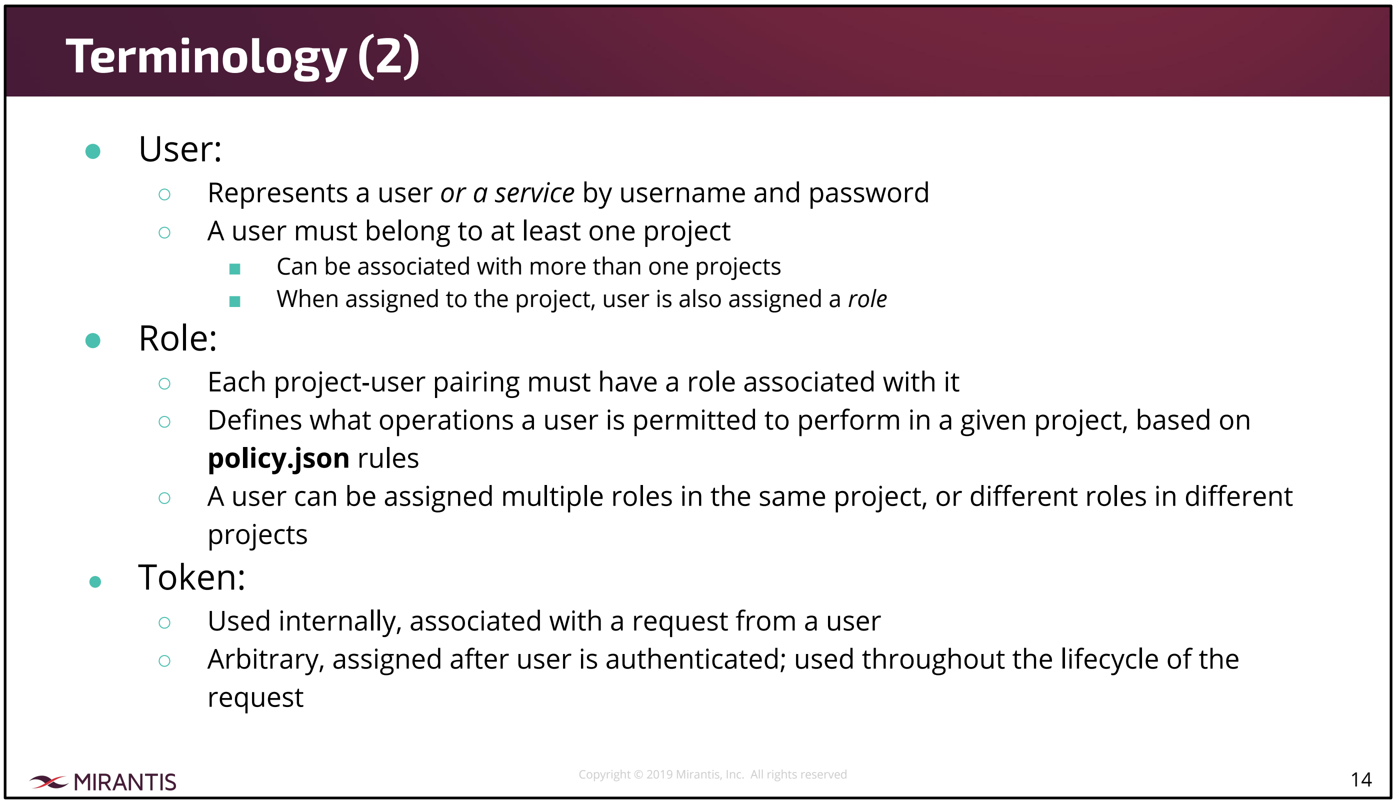


Terminology (1)

- Domain:
  - Highest level of abstraction for OpenStack resources
  - Represents a collection of projects, groups and users that defines administrative boundaries for managing OpenStack Identity entities
  - **For example, a customer or organization**
- Project:
  - Container used to group or isolate *resources* (networks, images, volumes, and so on)
  - **The resources are owned by the project**
    - Can be shared explicitly
  - Any requests for OpenStack services **must** specify a project
- Group:
  - A group of users who inherit roles defined by the admin
  - Optional, not used in typical install

MIRANTIS

Domain:
Domains are only available with Keystone v3, or later.
A collection of projects (and their users). Typically used in a multi-tenancy environment where there are multiple customers. Each customer has a unique domain, setting boundaries on resources (such as networks) they can use.
For example, CUSTOMER A can only deploy VMs to NETA. Network NETA is owned by a project in the CUSTOMER A domain. Any other customer, such as CUSTOMER B, does not have access to NETA or the VMs deployed in NETA.
There must be at least one domain. The default domain is called **default**.
Project (team):
A project is a container that is used to isolate resources and their ownership.
Projects represent the base unit of ownership in OpenStack, in that all resources in OpenStack are owned by a specific project. For example, when you create a network/subnetwork, it belongs to the team for the user who creates the resources. This might get confusing if the user is a member of multiple teams, especially when using the Horizon Dashboard UI.

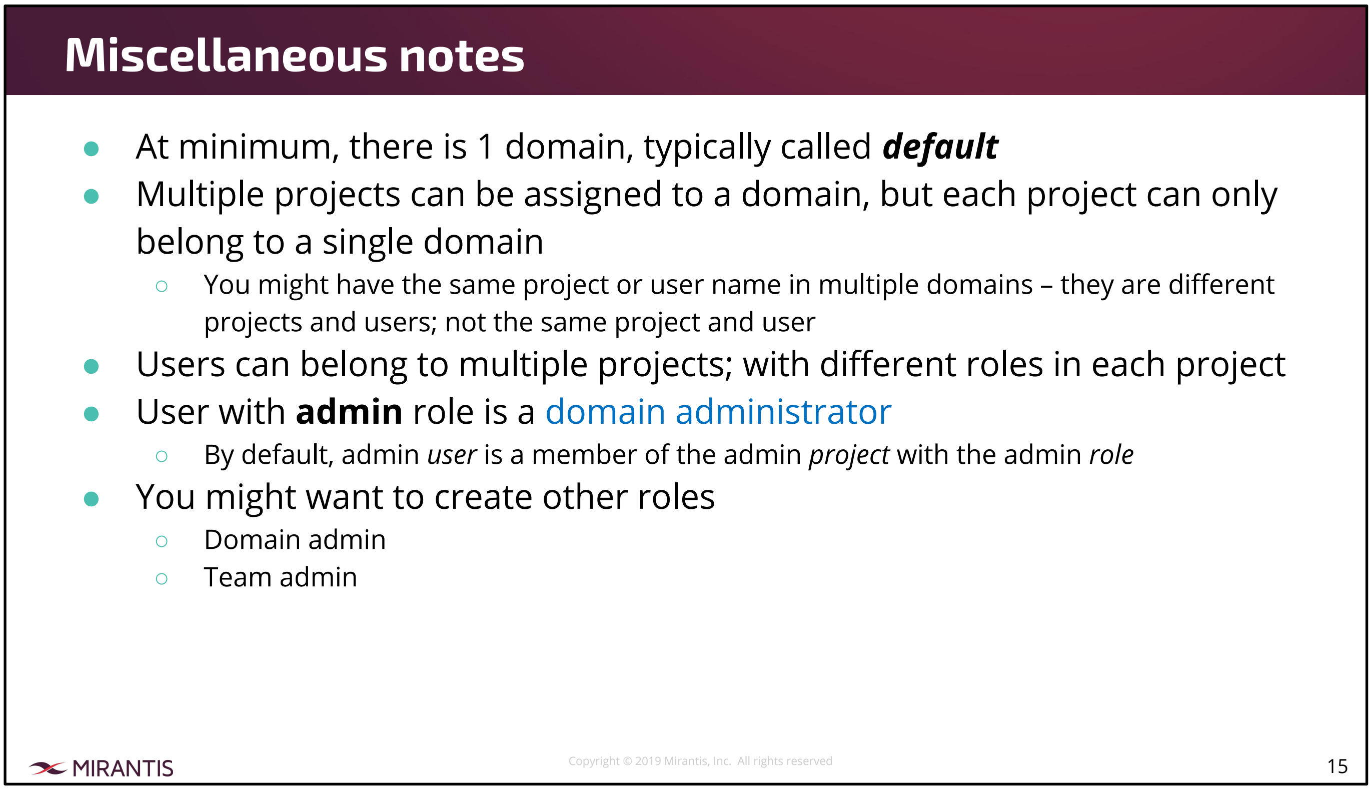


Terminology (2)

- User:
  - Represents a user *or a service* by username and password
  - A user must belong to at least one project
    - Can be associated with more than one projects
    - When assigned to the project, user is also assigned a *role*
- Role:
  - Each project-user pairing must have a role associated with it
  - Defines what operations a user is permitted to perform in a given project, based on **policy.json** rules
  - A user can be assigned multiple roles in the same project, or different roles in different projects
- Token:
  - Used internally, associated with a request from a user
  - Arbitrary, assigned after user is authenticated; used throughout the lifecycle of the request

MIRANTIS

User:
Users are exactly what you expect them to be – users who can log in to OpenStack and submit requests. Users can be members of 1 or more projects (teams). They can also have multiple roles within the same team or in different teams.
Role:
Each user is assigned at least 1 role when they are assigned to a project (team).
By default, OpenStack provides 2 roles: admin and user. You can create custom roles, such as a domain administrator, team administrator, or cloud administrator. If you create a custom role, you need to modify the rules in the various **policy.json** files for that custom role.



Miscellaneous notes

- At minimum, there is 1 domain, typically called ***default***
- Multiple projects can be assigned to a domain, but each project can only belong to a single domain
  - You might have the same project or user name in multiple domains – they are different projects and users; not the same project and user
- Users can belong to multiple projects; with different roles in each project
- User with **admin** role is a domain administrator
  - By default, admin *user* is a member of the admin *project* with the admin *role*
- You might want to create other roles
  - Domain admin
  - Team admin

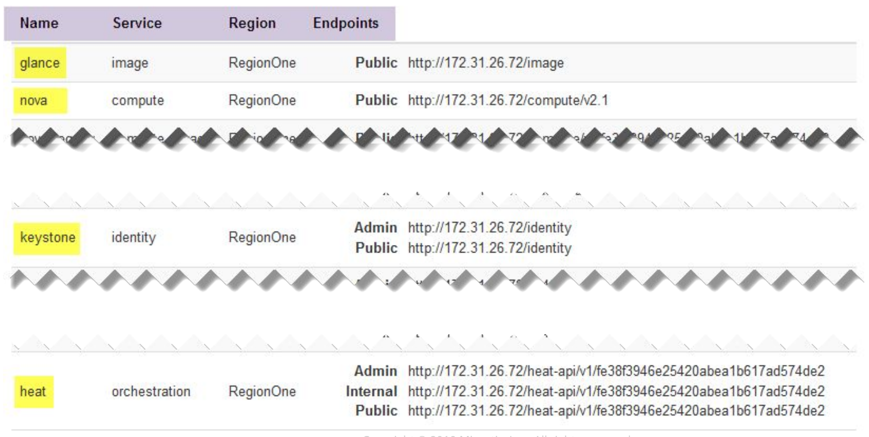
MIRANTIS

The ***default*** domain is created during the Keystone installation and configuration tasks. You do that in the lab exercises.
The ***default*** domain includes the *special* ***service*** *project* that contains all OpenStack services (Nova, Glance, and so on) that are installed in the environment.

Remember, the intent of having multiple domains is to provide segregation between the domains. For example, cloud service providers often provide cloud services to multiple customers. Each customer is assigned to a specific domain, eliminating the sharing of resources between the domains.

Service catalog

- OpenStack services query the Keystone **service catalog** to determine how to communicate with other OpenStack services
  - For example, during VM deployment *nova-api* retrieves the URL of Glance from the service catalog to send a request to the *glance-api* to validate the specified image
- You can view the service catalog from the CLI or UI



Name	Service	Region	Endpoints
glance	image	RegionOne	**Public** http://172.31.26.72/image
nova	compute	RegionOne	**Public** http://172.31.26.72/compute/v2.1
keystone	identity	RegionOne	**Admin** http://172.31.26.72/identity **Public** http://172.31.26.72/identity
heat	orchestration	RegionOne	**Admin** http://172.31.26.72/heat-api/v1/fe38f3946e25420abea1b617ad574de2 **Internal** http://172.31.26.72/heat-api/v1/fe38f3946e25420abea1b617ad574de2 **Public** http://172.31.26.72/heat-api/v1/fe38f3946e25420abea1b617ad574de2

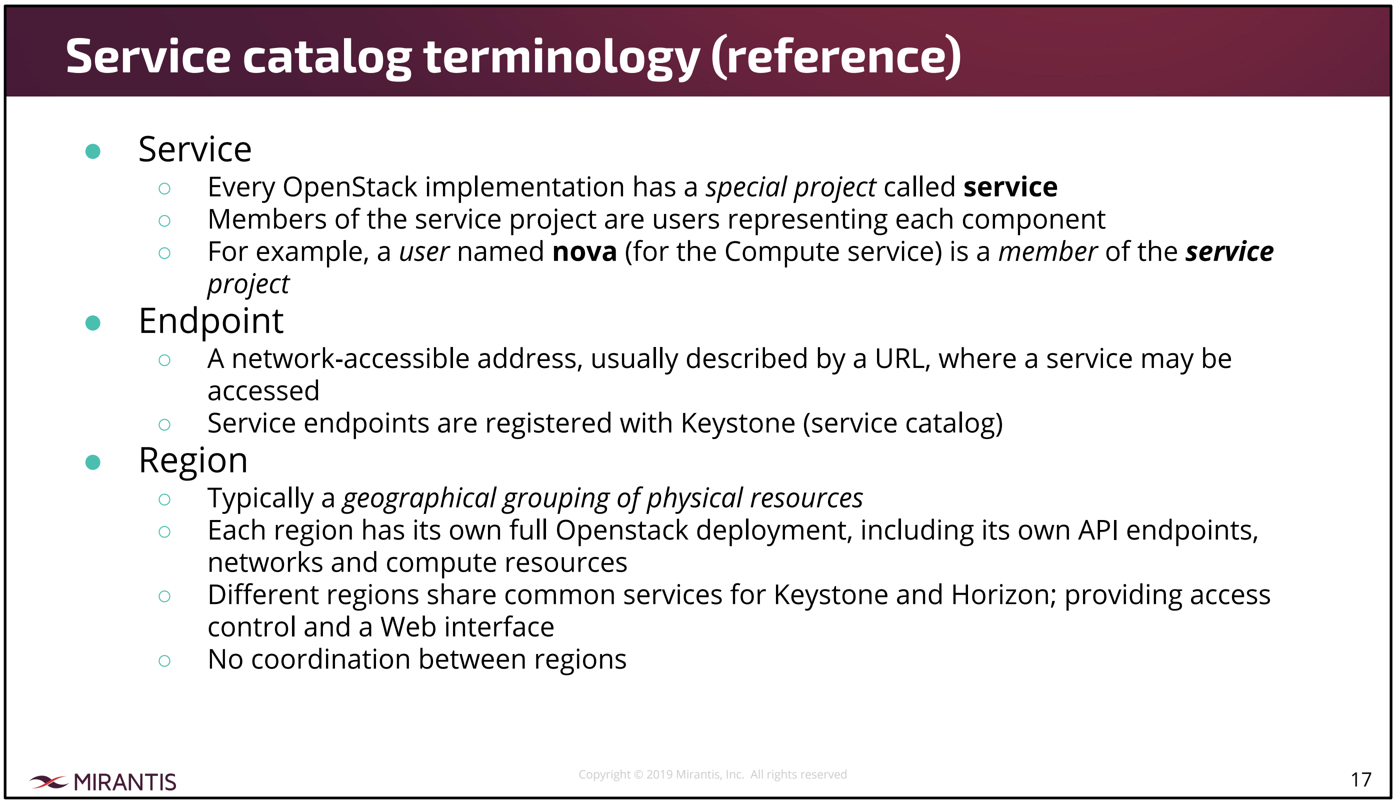
16

When you deploy a virtual machine instance, for example, Nova needs to communicate with Glance to verify the selected image is valid. Nova will query the service catalog to retrieve the URL to access Glance and then send a request to Glance (using that URL) to validate the image. This occurs during **step 4** of the *request process flow* discussed in this lecture.

Each service in the service catalog contains at least 1 endpoint. Some have 3:

- 3 endpoints (URLs):
  - **public url:** the public facing endpoint for the service
  - **internal url:** the internal facing endpoint. Usually the same as the public url. This endpoint is used when components need to communicate, such as Nova validating an image with Glance.
  - **admin url:** the endpoint for service administration functions. Not applicable to all components.
- Each endpoint is associated with a region
  - *region: the name given to a collection of cloud services (default: RegionOne)*
- Service id: the service the endpoint is associated with (Keystone, Nova, Glance, and so on)
- Service name: keystone, nova, glance, and so on
- Service type: identity, compute, image, and so on

Note: services with less than 3 endpoints have been migrated to use of Apache Web server with mod_wsgi.



Service catalog terminology (reference)

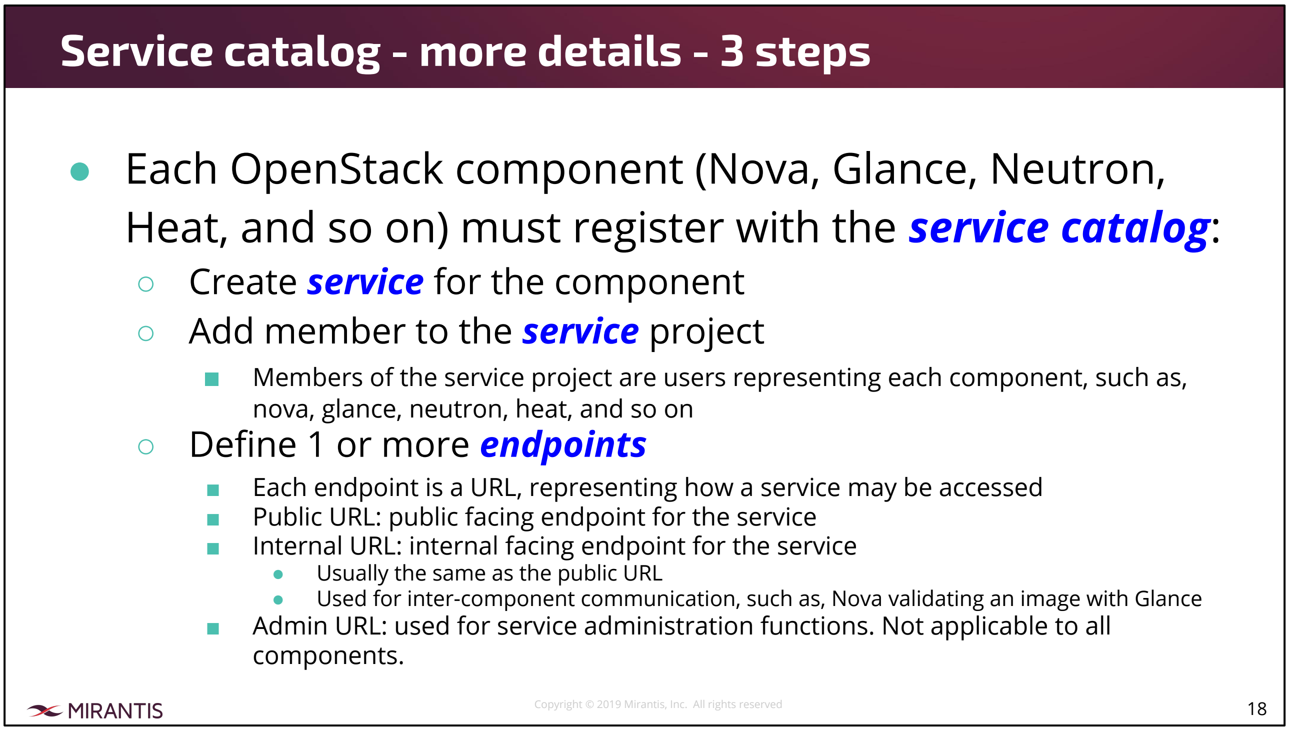
- Service
  - Every OpenStack implementation has a *special project* called **service**
  - Members of the service project are users representing each component
  - For example, a *user* named **nova** (for the Compute service) is a *member* of the ***service** project*
- Endpoint
  - A network-accessible address, usually described by a URL, where a service may be accessed
  - Service endpoints are registered with Keystone (service catalog)
- Region
  - Typically a *geographical grouping of physical resources*
  - Each region has its own full Openstack deployment, including its own API endpoints, networks and compute resources
  - Different regions share common services for Keystone and Horizon; providing access control and a Web interface
  - No coordination between regions

MIRANTIS

When you define your Keystone environment, one of the tasks is to create a special project called **service**. Each OpenStack component has a user named for the component (nova for the Compute service, glance for the Image service, neutron for the Network service, and so on). Those users are defined as members of the special service project.

Each OpenStack component must register **endpoints** with the service catalog. The endpoints define how to address the component. Typically this is a URL. Each components has 3 endpoints defined: public, admin, and internal.

Regions are typically used to define multiple instances of a service. For example, if you have multiple hypervisors, you have multiple instances of the nova-compute process (compute node) to interact with each hypervisor. Regions are then used to distinguish the nova-compute instances. Regions can also be used to represent a geographical grouping of resources. By default, there is 1 region, RegionOne. Regions are discussed in more detail in the Nova (Compute service) lecture, along with *availability zones*.



Service catalog - more details - 3 steps

- Each OpenStack component (Nova, Glance, Neutron, Heat, and so on) must register with the ***service catalog***:
  - Create ***service*** for the component
  - Add member to the ***service*** project
    - Members of the service project are users representing each component, such as, nova, glance, neutron, heat, and so on
  - Define 1 or more ***endpoints***
    - Each endpoint is a URL, representing how a service may be accessed
    - Public URL: public facing endpoint for the service
    - Internal URL: internal facing endpoint for the service
      - Usually the same as the public URL
      - Used for inter-component communication, such as, Nova validating an image with Glance
    - Admin URL: used for service administration functions. Not applicable to all components.

MIRANTIS

You execute these steps for each component (Nova, Glance, Neutron, and so on) during the labs on day 4 of the OS250 course.

Service catalog - more details (steps 1-2)

- To display members of the ***service*** project:

```
openstack user list --project service
+----------------------------------+---------------+
| ID                               | Name          |
+----------------------------------+---------------+
| 42a625d472f04cd483ef5482014207f7 | gnocchi       |
| bdcfa996280a484bb98023a3041f1dfc | ceilometer    |
| 627a12a790c448b9a5274a8a9f407211 | nova          |
| d23e57aa1982477e88a519adc472ac29 | glance-swift  |
| c972b3c60fc3443e80c8514a6f4663eb | cinder        |
| cfdfa309046e490e8577fdcedaf99ba5 | glance        |
| cfb5d65ab5a94f959ed3b4fc87217111 | octavia       |
| 581f25eec4bd47099bb6d9d53406ec70 | placement     |
| 54675ec5fc4a494c82cd0ef1aa788101 | heat          |
| 4d18f1d684cc4e938e083954f20f7947 | aodh          |
| 601fb410f3794b9b8fdb2eba2abc0ed9 | swift         |
| ea2c0c3f26b241c3ada4298cfdea6272 | neutron       |
+----------------------------------+---------------+
```

- To create the ***nova*** service:

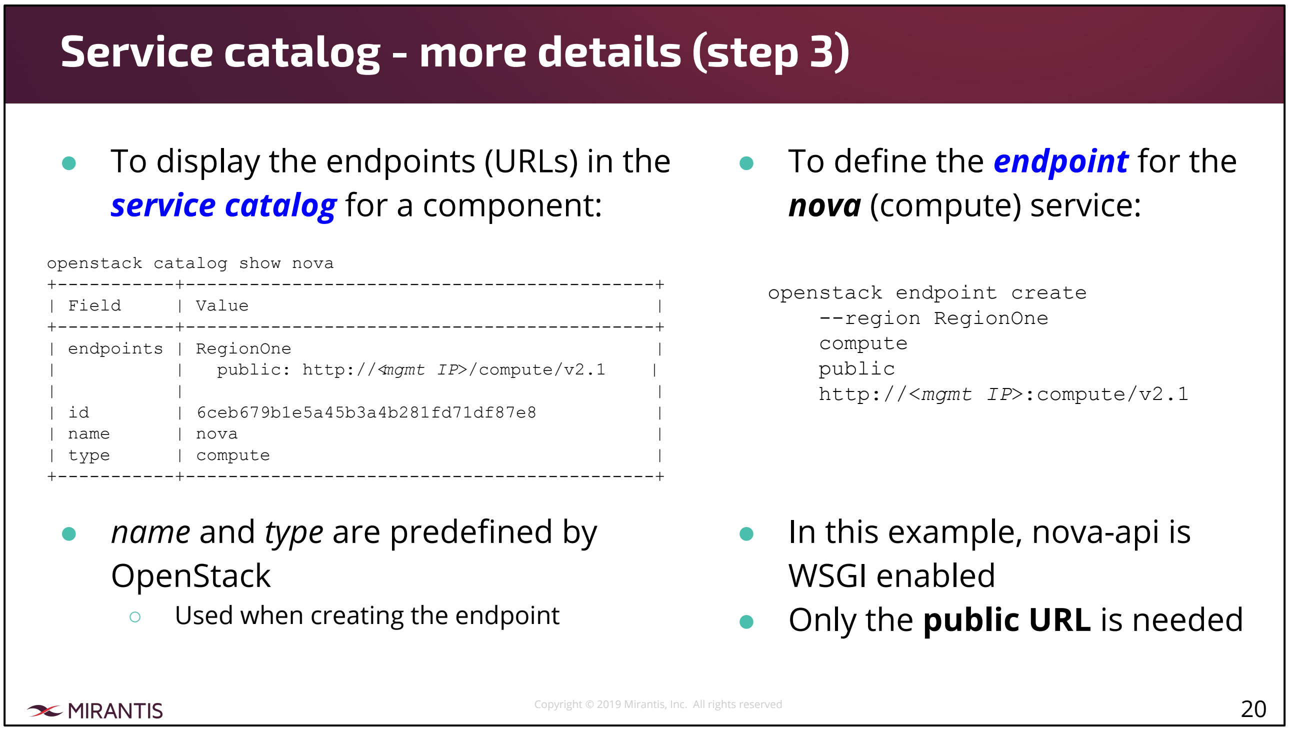
```
openstack service create --name nova
    --description "OpenStack Compute"
    compute
```

- To add the ***nova*** user to the ***service*** project:

```
openstack user create nova
    --domain default --password nova

openstack role add admin
    --project service --user nova
```

MIRANTIS



Service catalog - more details (step 3)

- To display the endpoints (URLs) in the ***service catalog*** for a component:

```
openstack catalog show nova
+-----------+--------------------------------------------+
| Field     | Value                                      |
+-----------+--------------------------------------------+
| endpoints | RegionOne                                  |
|           |   public: http://<mgmt IP>/compute/v2.1    |
|           |                                            |
| id        | 6ceb679b1e5a45b3a4b281fd71df87e8           |
| name      | nova                                       |
| type      | compute                                    |
+-----------+--------------------------------------------+
```

- *name* and *type* are predefined by OpenStack
  - Used when creating the endpoint

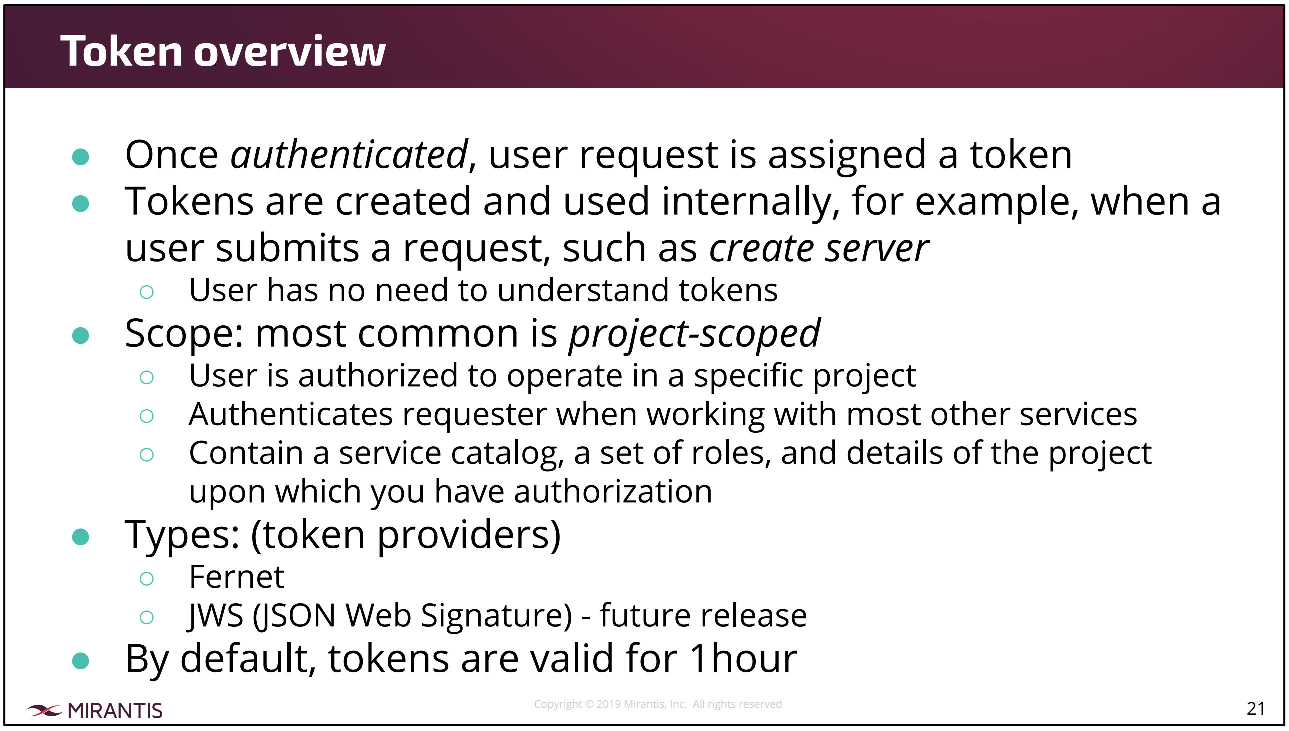
- To define the ***endpoint*** for the ***nova*** (compute) service:

```
openstack endpoint create
    --region RegionOne
    compute
    public
    http://<mgmt IP>:compute/v2.1
```

- In this example, nova-api is WSGI enabled
- Only the **public URL** is needed

MIRANTIS

The supported OpenStack service types are documented here:
https://service-types.openstack.org/service-types.json.2019-07-23T14:12:22.205489



Token overview

- Once *authenticated*, user request is assigned a token
- Tokens are created and used internally, for example, when a user submits a request, such as *create server*
  - User has no need to understand tokens
- Scope: most common is *project-scoped*
  - User is authorized to operate in a specific project
  - Authenticates requester when working with most other services
  - Contain a service catalog, a set of roles, and details of the project upon which you have authorization
- Types: (token providers)
  - Fernet
  - JWS (JSON Web Signature) - future release
- By default, tokens are valid for 1hour

MIRANTIS

The scope of a token can be:

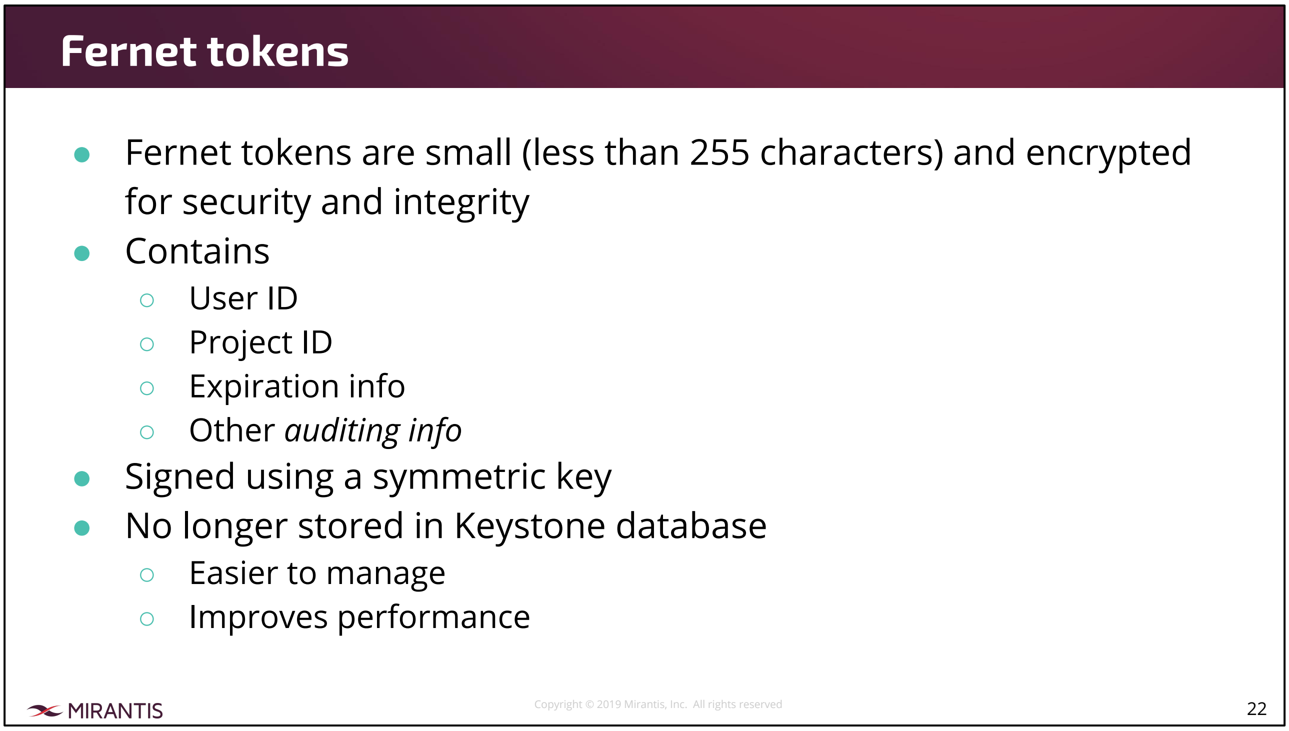
- Un-scoped
- Project-scoped (most common use)
- Domain-scoped
- System-scoped

Supported token providers are:

- Fernet (the only supported token provider in Rocky release)
- JWS (not supported in Rocky, but will be in a future release)

By default, a token is valid for an hour. Most requests will complete within less time. Some longer running requests might need more time. For example, uploading large images to the Glance image repository, or deploying Heat stacks with multiple instances/larger images/installing and configuring software In those cases, a token re-validation process is handled internally.

For more details:
https://docs.openstack.org/keystone/latest/admin/tokens-overview.html



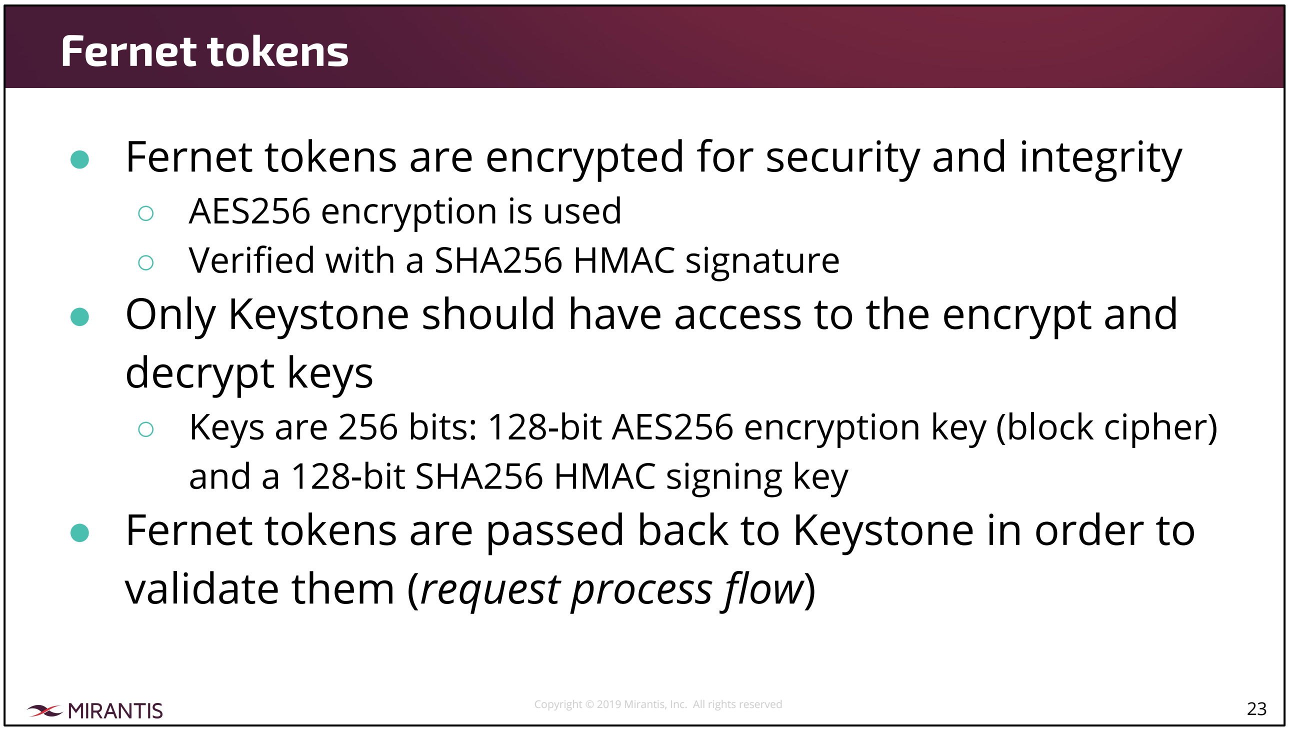
Fernet tokens

- Fernet tokens are small (less than 255 characters) and encrypted for security and integrity
- Contains
  - User ID
  - Project ID
  - Expiration info
  - Other *auditing info*
- Signed using a symmetric key
- No longer stored in Keystone database
  - Easier to manage
  - Improves performance

MIRANTIS

This slide introduces fernet tokens. There is a lot to learn about fernet tokens:
https://docs.openstack.org/keystone/rocky/admin/identity-fernet-token-faq.html

Fernet tokens are encrypted using AES256 (Advanced Encryption Standard) encryption and verified with a SHA256 HMAC signature. For more information, start here:
https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
HMAC (hash-based message authentication code) is a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key.



Fernet tokens

- Fernet tokens are encrypted for security and integrity
  - AES256 encryption is used
  - Verified with a SHA256 HMAC signature
- Only Keystone should have access to the encrypt and decrypt keys
  - Keys are 256 bits: 128-bit AES256 encryption key (block cipher) and a 128-bit SHA256 HMAC signing key
- Fernet tokens are passed back to Keystone in order to validate them (*request process flow*)

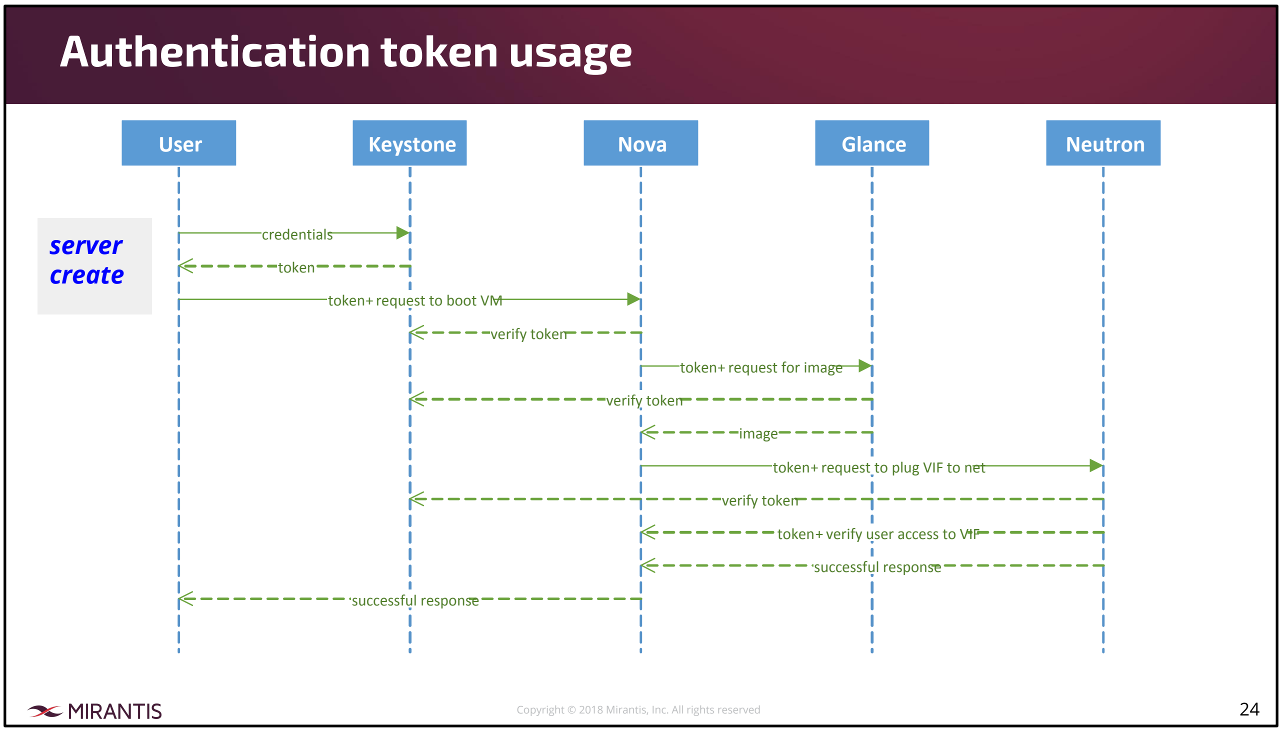
MIRANTIS

This slide introduces fernet tokens. There is a lot to learn about fernet tokens: https://docs.openstack.org/keystone/rocky/admin/identity-fernet-token-faq.html

Fernet tokens use Advanced Encryption Standard (AES) encryption. For more information, start here:
https://en.wikipedia.org/wiki/Advanced_Encryption_Standard

HMAC (hash-based message authentication code) is a specific type of message authentication code (MAC) involving a cryptographic hash function and a secret cryptographic key.

By default, a token is valid for an hour. Most requests will complete within less time. Some longer running requests might need more time. For example, uploading large images to the Glance image repository, or deploying Heat stacks with multiple instances/larger images/installing and configuring software In those cases, a token re-validation process is handled internally..



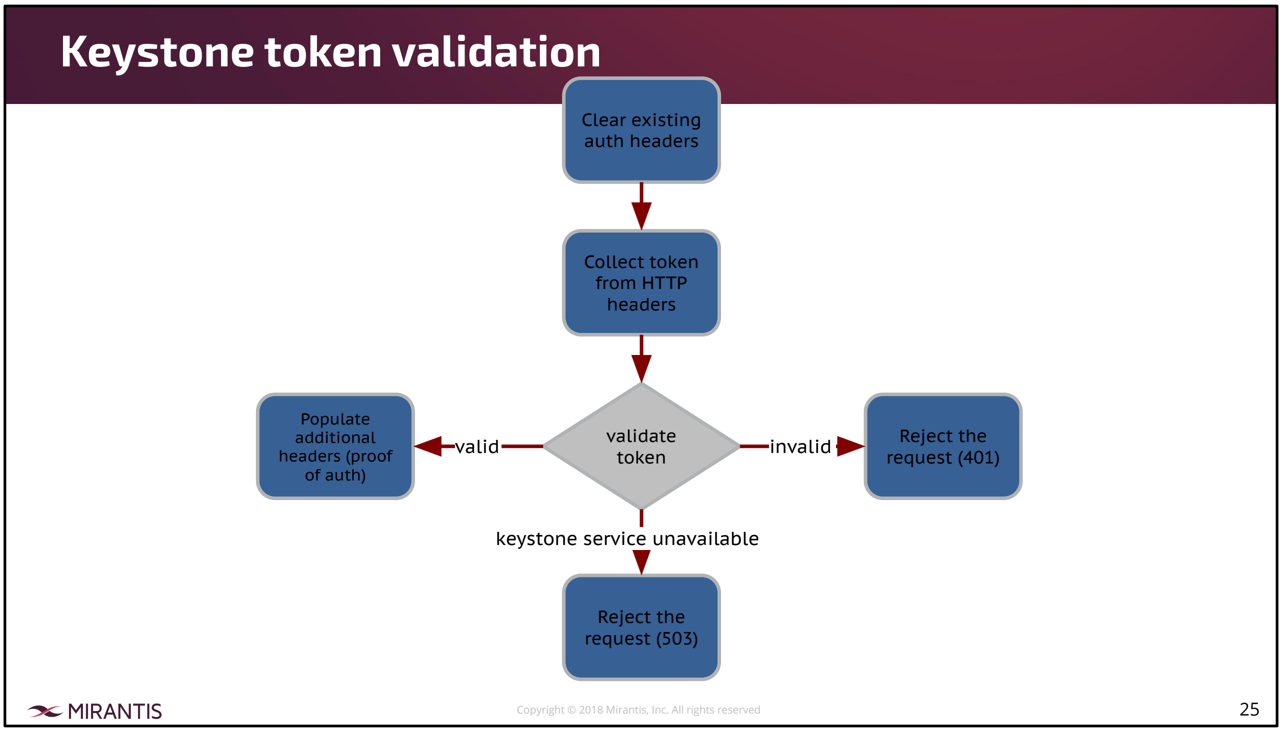
Authentication token usage

24

When the user submits a request (for example, deploy instance), their credentials are authenticated with Keystone and the request is assigned an **auth-token** to be used for the life of the request. Example credentials are defined in the CLI script, such as ***credrc.sh***:

- OS_AUTH_TYPE=password: Use password authentication
- OS_AUTH_URL=http://172.31.26.72/identity/: URL where Keystone is running
- OS_IDENTITY_API_VERSION="3": Use Keystone v3 API
- OS_PROJECT_NAME="demo": project
- OS_USERNAME="admin": user
- OS_PASSWORD="nova": password
- OS_PROJECT_DOMAIN_ID="default"
- OS_USER_DOMAIN_ID="default"
- OS_REGION_NAME="RegionOne": (default value) identifies region where Keystone is running

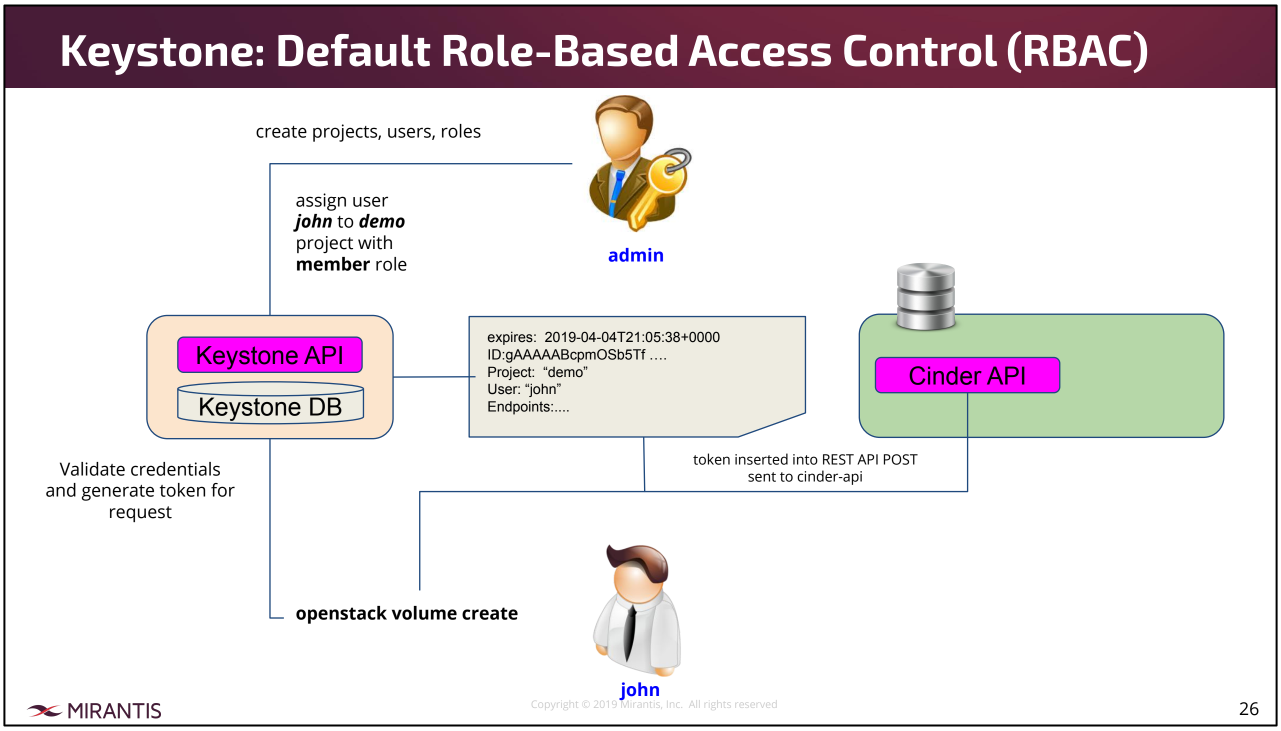
The token is passed between components (such as, nova-compute to glance-api). Each component re-validates the token with Keystone. This ensures the request is valid. For example, it reduces the possibility that a hacker can submit a request because they will not have a valid token.



Keystone token validation

Clear existing auth headers
Collect token from HTTP headers
validate token
valid
Populate additional headers (proof of auth)
invalid
Reject the request (401)
keystone service unavailable
Reject the request (503)

401 = Not authorized
503 = Service not available


Keystone: Default Role-Based Access Control (RBAC)

create projects, users, roles
assign user *john* to *demo* project with **member** role
admin
Keystone API
Keystone DB
Cinder API
Validate credentials and generate token for request
token inserted into REST API POST sent to cinder-api
openstack volume create
john

```
expires:  2019-04-04T21:05:38+0000
ID:gAAAAABcpmOSb5Tf ….
Project:  "demo"
User: "john"
Endpoints:....
```

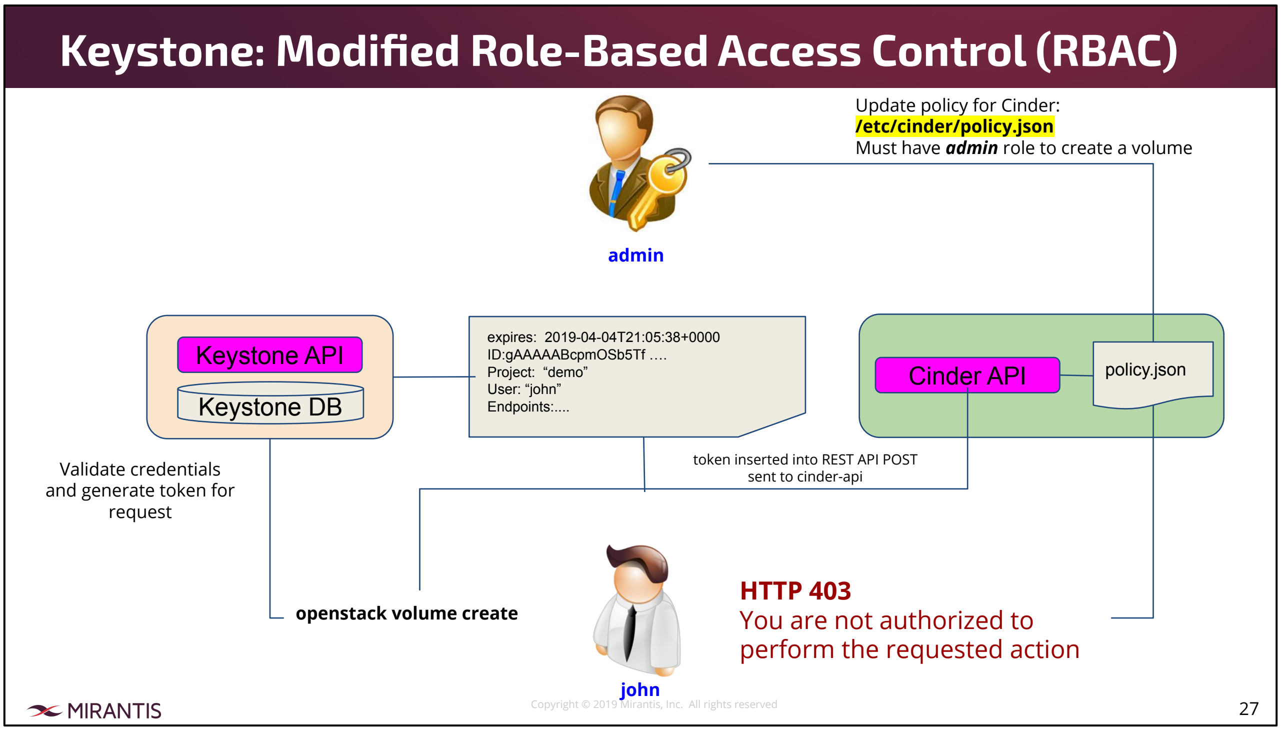
The administrator (role=admin) has the responsibility to configure users, roles, and projects using the Dashboard UI or CLI. This sends requests to the Keystone API.

The process of creating a user includes assigning a role to the user, when the user is added as a member of a project. Users might have multiple roles in a project. They might also be members of more than one project, including having different roles in each project.

The most common roles are admin and member. You can create other roles, as needed. For example, team_leader or dept_manager.
Keystone provides role-based access to ***operations*** on OpenStack resources. For example, by default:

- Any user with the admin role can perform most, if not all,operations on OpenStack resources.
- Any user with the member role is restricted from listing users or performing updates for other users. They are, however, allowed to perform normal operations in the Cloud, such as creating Cinder volumes.

The user (John in this example), who is a member of the demo team (role = member), issues a CLI request to create a Cinder volume. Keystone validates the user credentials as well as their ***operational access*** (role-based policy access). When authenticated, the auth-token is generated and inserted into the REST API POST request sent to cinder-api.



Keystone: Modified Role-Based Access Control (RBAC)

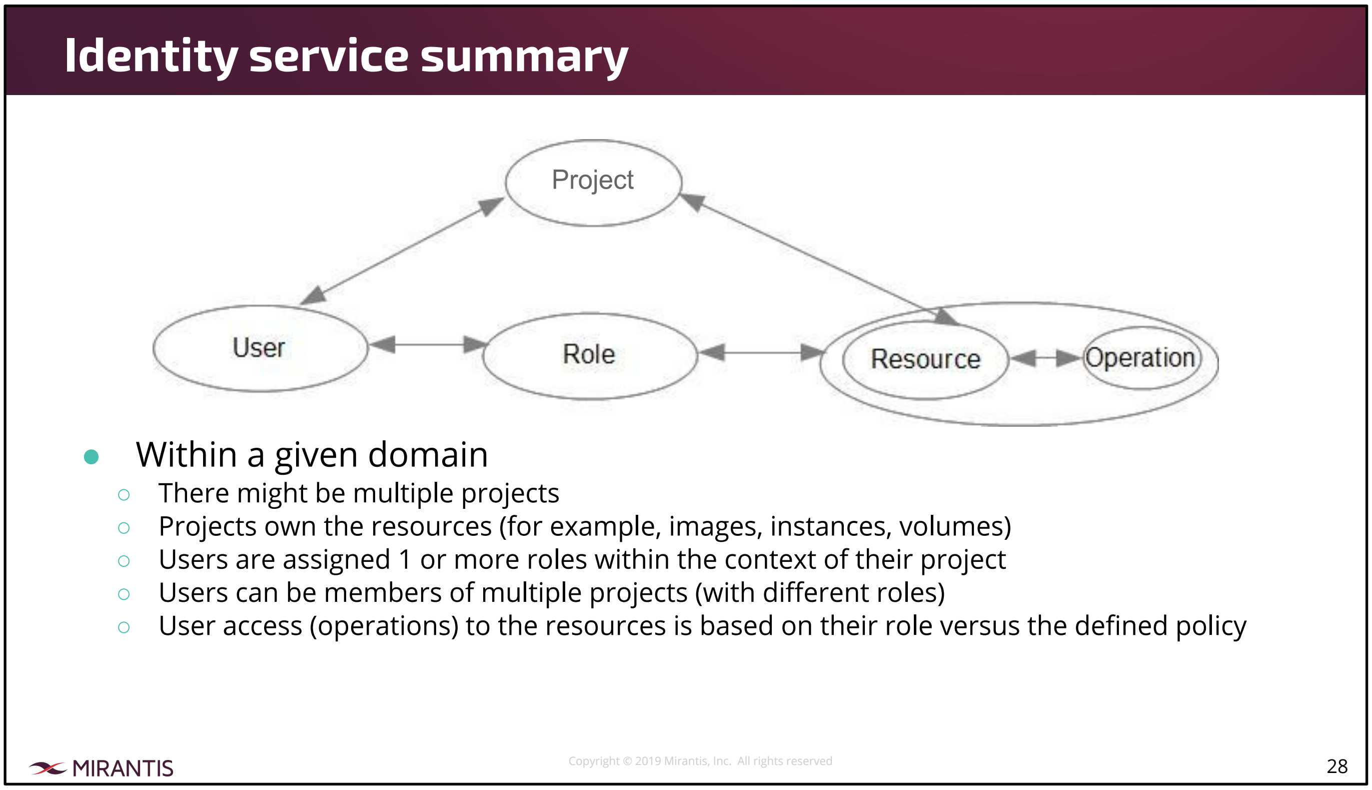
Update policy for Cinder:
/etc/cinder/policy.json
Must have *admin* role to create a volume
admin
Keystone API
Keystone DB
expires: 2019-04-04T21:05:38+0000
ID:gAAAAABcpmOSb5Tf ….
Project: "demo"
User: "john"
Endpoints:....
Cinder API
policy.json
Validate credentials and generate token for request
token inserted into REST API POST sent to cinder-api
openstack volume create
HTTP 403
You are not authorized to perform the requested action
john
MIRANTIS
Copyright © 2019 Mirantis, Inc. All rights reserved
27

Suppose John's company only wants ***admin operational access*** to create Cinder volumes.

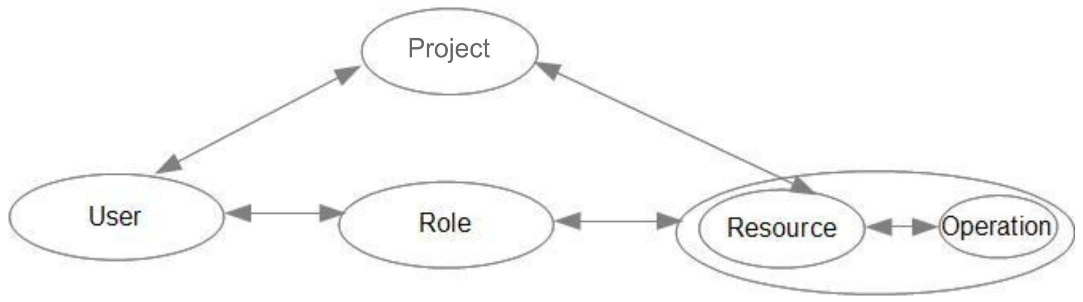
Each OpenStack component supports configuring its policy files located in the ***/etc/<project>/*** directory. The administrator can edit the policies in the **policy.json** or **policy.yaml** files based on the requirements of the organization.

Since John has the **member** role, his request will not be allowed, with an HTTP 403 error message.

Note: In some cases, a policy file might not exist for the OpenStack component. See the discussion on the next slide.



Identity service summary



- Within a given domain
  - There might be multiple projects
  - Projects own the resources (for example, images, instances, volumes)
  - Users are assigned 1 or more roles within the context of their project
  - Users can be members of multiple projects (with different roles)
  - User access (operations) to the resources is based on their role versus the defined policy

MIRANTIS

There are more complex diagrams of the OpenStack Access Control (OSAC) model. To keep it simple, let's eliminate groups and use the knowledge that projects must belong to a domain. Roles can not be assigned to projects, but they can be assigned to groups.

To summarize:

- There can be multiple projects in a domain
- Users can be members of 1 or more project, with 1 or more roles in each project
- When a user is added to a project, they are assigned a role (admin, user, and so on)
- The role correlates to rules defined in the various policy.json files
- The user can perform operations on the resources owned by their project
- Based on the rules in the policy.json files, users have access to perform specific operations (list users, boot a VM, and so on)
- The user is authenticated and provided with an auth-token to use for their requested operation

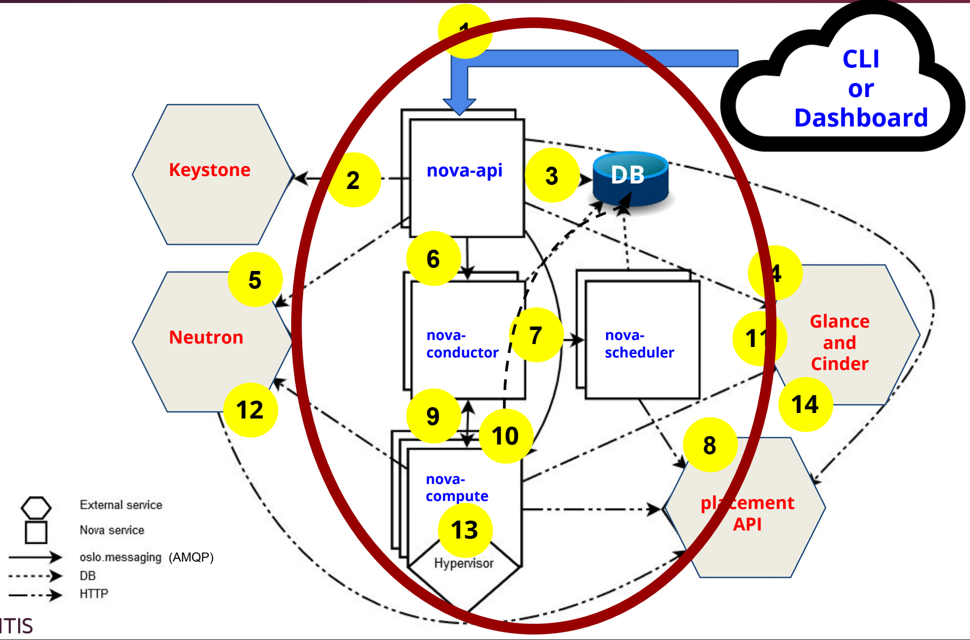
Request Process Flow

Steps 2-14:
nova-api >
nova-conductor >
nova-scheduler >
nova-placement >
nova-conductor >
nova-compute >
hypervisor

Nova (Compute service)

Plus, calls to Glance,
Neutron,
and Cinder

Nova (Compute service)



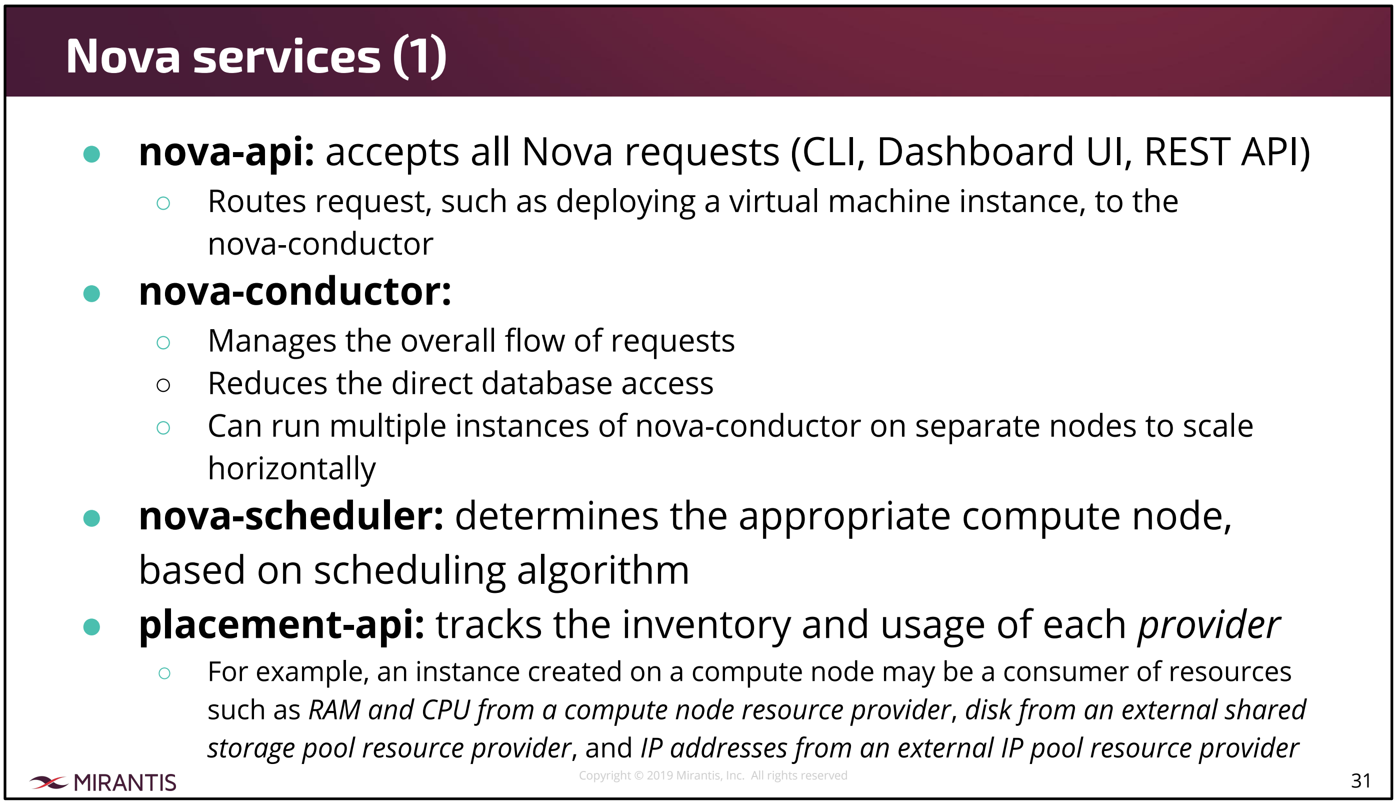
MIRANTIS

When the user logs in to the dashboard (or CLI), their user credentials are authenticated with the Identity Service via REST API, receiving an **auth-token** which is *used for the UI session*. When the user submits a request, such as launch instance, **a new auth-token** *is assigned for the request* and used for the lifecycle of the request, including inter-component communications.

1. The dashboard or CLI converts the **launch instance** request to a REST API POST request, obtains an **auth-token**, and sends the request to nova-api The **auth-token** is part of the request and will be used throughout the lifecycle of the request.
2. nova-api receives the request and sends a request to the Identity Service for validation of the **auth-token** and access permission. The Identity Service validates the **auth-token** and sends updated authentication headers with roles and permissions.
3. nova-api checks for conflicts with nova-database and then creates initial database entry for the new instance.
    - nova-api also validates the flavor and the compute resource quotas
4. nova-api sends request to Glance (glance-api) to validate the image
5. nova-api sends request to Neutron (neutron-server) to validate network, port, and network quotas.
6. nova-api sends the rpc.call request to nova-conductor expecting to get updated instance entry with host ID specified.
7. nova-conductor receives and processes the message, sending it to nova-scheduler. nova-scheduler picks up the request from the queue.
8. nova-scheduler interacts with the placement API to find an appropriate host via filtering and weighing. nova-scheduler returns the updated instance entry with the appropriate host ID (of chosen hypervisor) after filtering and weighing.
9. nova-conductor sends the rpc.cast request to nova-compute for launching an instance on the appropriate host.

10. nova-compute picks up the request from the queue. nova-compute sends the rpc.call request to nova-conductor to fetch the instance information such as host ID and flavor (RAM, CPU, Disk) from the database.
11. nova-compute sends a REST call to glance-api, using the Image ID, to retrieve the Image URI from the Image Service, and loads the image from the image storage.
    - nova-compute also retrieves the image metadata.
12. nova-compute performs a REST call to the Neutron API (neutron-server) to allocate and configure the network so that the instance gets a port with an IP address.
    - neutron-server sends request to DHCP server for IP address, for example.
    - nova-compute then retrieves the network info.
13. nova-compute generates data for the hypervisor driver and executes the request on the hypervisor (via libvirt or specific hypervisor API) ro create an instance.
14. If a volume is to be attached, nova-compute performs a REST call to the cinder-api to attach the volume to the instance. nova-compute then retrieves the block storage info.

Important Note: As each component, such as glance-api for Glance, receives a request, it validates the **auth-token** with Keystone before taking any action on the request. If the token is not valid, the request is rejected. The token validation occurs at each step above and is not shown or discussed in the flow.



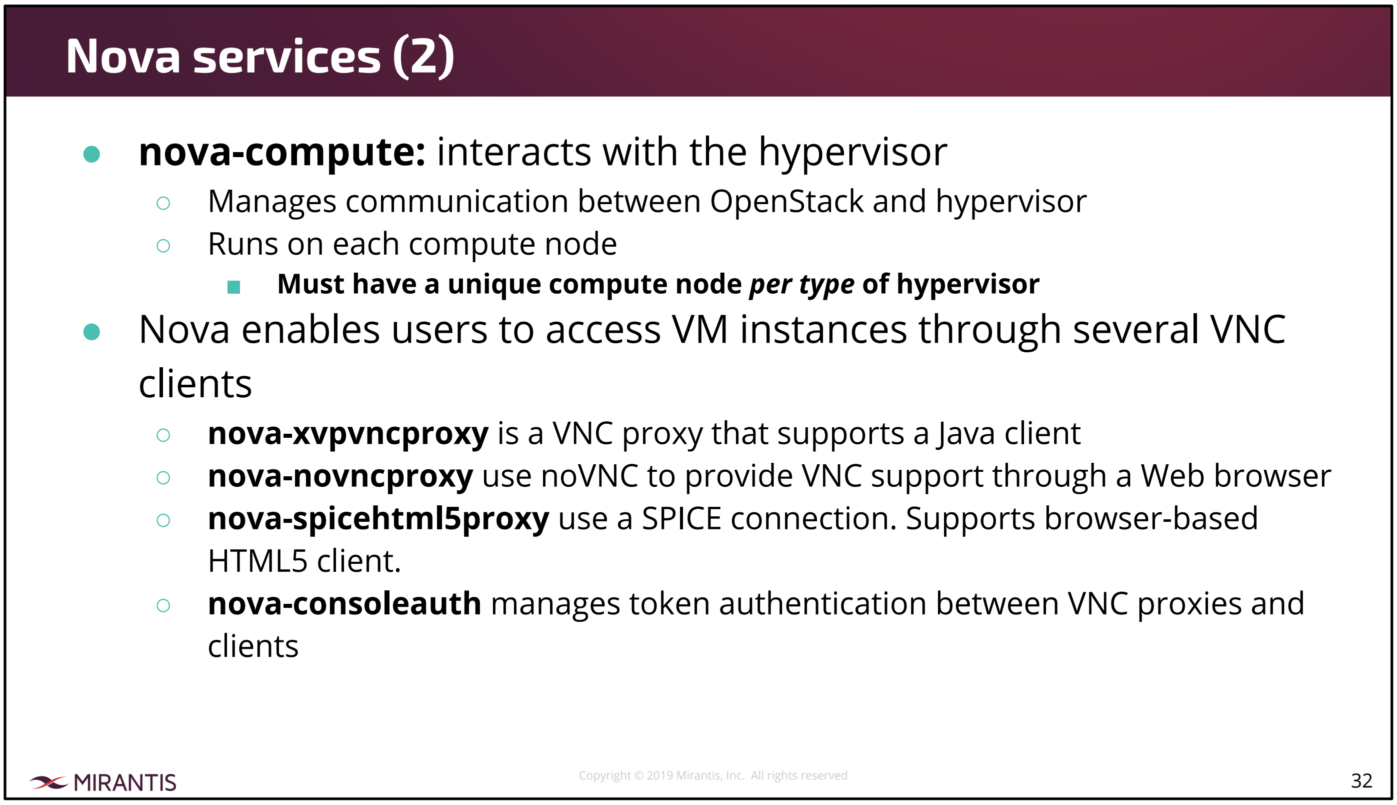
Nova services (1)

- **nova-api:** accepts all Nova requests (CLI, Dashboard UI, REST API)
  - Routes request, such as deploying a virtual machine instance, to the nova-conductor
- **nova-conductor:**
  - Manages the overall flow of requests
  - Reduces the direct database access
  - Can run multiple instances of nova-conductor on separate nodes to scale horizontally
- **nova-scheduler:** determines the appropriate compute node, based on scheduling algorithm
- **placement-api:** tracks the inventory and usage of each *provider*
  - For example, an instance created on a compute node may be a consumer of resources such as *RAM and CPU from a compute node resource provider*, *disk from an external shared storage pool resource provider*, and *IP addresses from an external IP pool resource provider*

MIRANTIS

nova-api supports APIs for both OpenStack and Amazon EC2 (compatible). It communicates with other OpenStack components through message queue (RabbitMQ) or HTTP (in the case of the Swift object store),

The placement-api is currently part of the Nova set of services. In a future release, it will be split out and become its own OpenStack component.



Nova services (2)

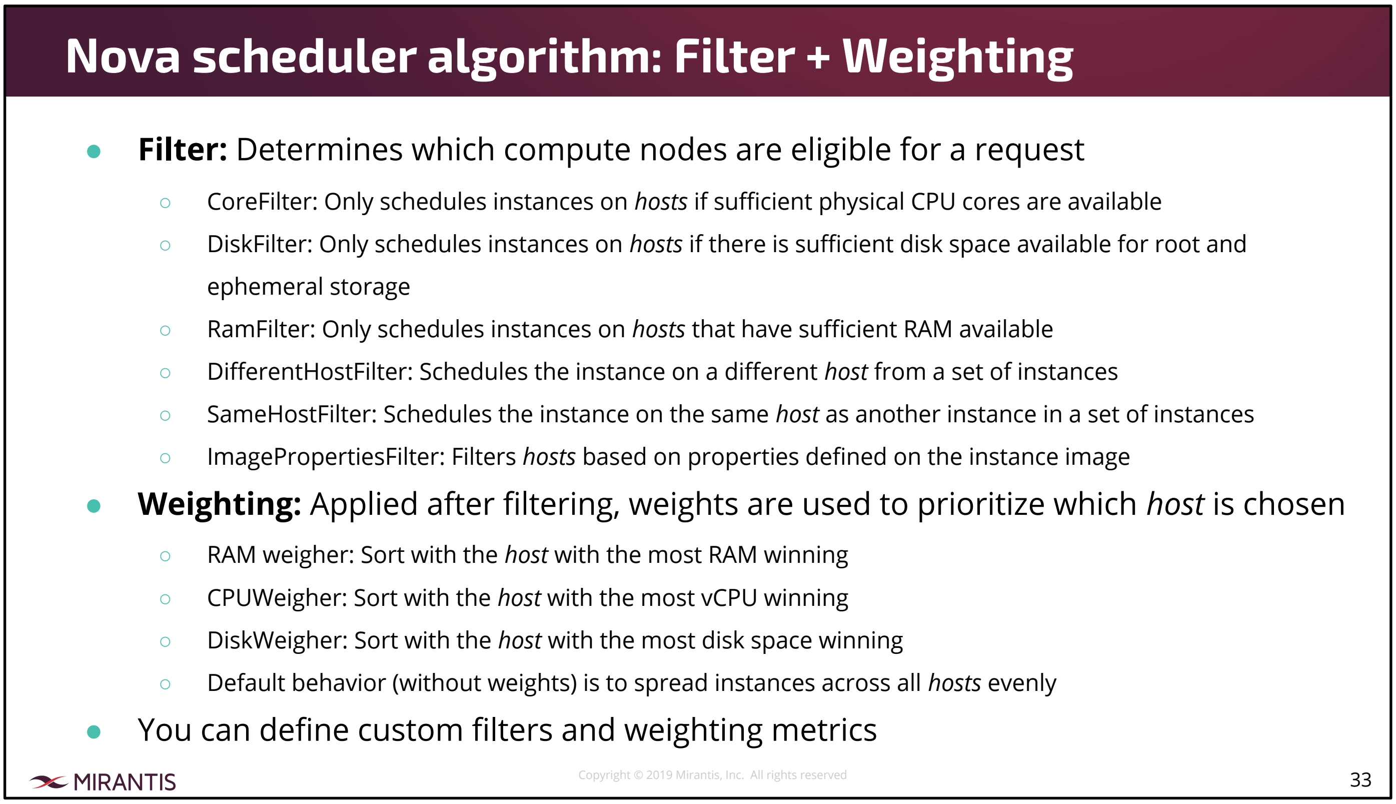
- **nova-compute:** interacts with the hypervisor
  - Manages communication between OpenStack and hypervisor
  - Runs on each compute node
    - **Must have a unique compute node *per type* of hypervisor**
- Nova enables users to access VM instances through several VNC clients
  - **nova-xvpvncproxy** is a VNC proxy that supports a Java client
  - **nova-novncproxy** use noVNC to provide VNC support through a Web browser
  - **nova-spicehtml5proxy** use a SPICE connection. Supports browser-based HTML5 client.
  - **nova-consoleauth** manages token authentication between VNC proxies and clients

MIRANTIS

Notes:

- In a production environment, where there are multiple machines running OpenStack services, **nova-compute** runs on the compute node.
- You can run more than one type of hypervisor. You need a separate instance of the compute node with nova-compute process for each type of hypervisor you are supporting. The most commonly used hypervisor is KVM (with QEMU and libvirtd).

For a full list of supported hypervisors:
https://wiki.openstack.org/wiki/HypervisorSupportMatrix



Nova scheduler algorithm: Filter + Weighting

- **Filter:** Determines which compute nodes are eligible for a request
  - CoreFilter: Only schedules instances on *hosts* if sufficient physical CPU cores are available
  - DiskFilter: Only schedules instances on *hosts* if there is sufficient disk space available for root and ephemeral storage
  - RamFilter: Only schedules instances on *hosts* that have sufficient RAM available
  - DifferentHostFilter: Schedules the instance on a different *host* from a set of instances
  - SameHostFilter: Schedules the instance on the same *host* as another instance in a set of instances
  - ImagePropertiesFilter: Filters *hosts* based on properties defined on the instance image
- **Weighting:** Applied after filtering, weights are used to prioritize which *host* is chosen
  - RAM weigher: Sort with the *host* with the most RAM winning
  - CPUWeigher: Sort with the *host* with the most vCPU winning
  - DiskWeigher: Sort with the *host* with the most disk space winning
  - Default behavior (without weights) is to spread instances across all *hosts* evenly
- You can define custom filters and weighting metrics

MIRANTIS

33

The Nova scheduler process (algorithm) is similar to that of the Cinder scheduler, discussed earlier in this class.
The Nova scheduling filters are defined in the **nova.conf** file on the controller node. By default, all filters (30 total) are loaded. You might not want to use all filters. In that case, you can code a **scheduler_default_filters** property in **nova.conf** and specify the filters you want to use.

The Nova filters and weighting options are documented here:
https://docs.openstack.org/nova/rocky/user/filter-scheduler.html

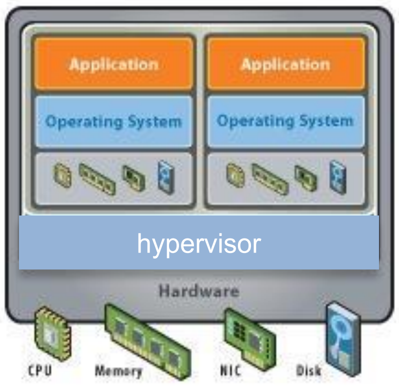
Terminology note- In this case, ***host*** means a compute node-hypervisor pair.

VM resources - flavors

- Each VM needs four types of resources
  - NIC (at least 1)
  - CPU
  - Memory
  - Hard disk

Nova flavor:
combination of CPU, memory, disk

- Plus an operating system based on Glance image



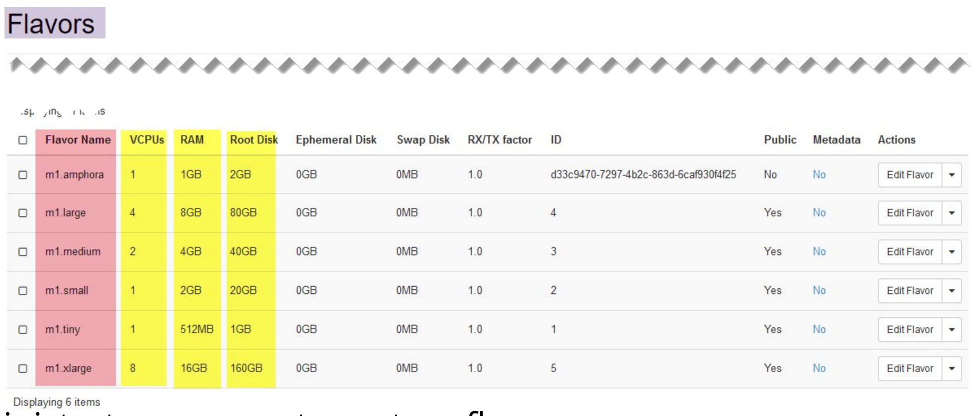
Notice that physical machine has same (real) resources

The amount of CPU, memory, and disk that is allocated to a VM is controlled with a ***Nova flavor***, such as m1.tiny or m1.small. You specify the flavor when you submit a request to deploy an instance.
Each VM that is deployed has at least 1 NIC associated with it. Some VMs will have more. The IP address of the NIC is configured from a pool of addresses assigned to your network. A MAC address is also assigned to the deployed VM, based on the **base_mac** (for example, fa:16:3e:4f:00:00) property in the **neutron.conf** file.
When the VM is deployed, the request also identifies the Glance image to use. The Glance image must contain, at minimum, a bootable operating system. The image might also contain software that is installed and configured.
In addition to the operating system you might choose to:

- automatically deploy software to the VM at boot time (for example, using the OpenStack Heat component) or manually after the VM is deployed.
- attach additional block storage (Cinder volumes) at boot time or after the VM is deployed.

Nova flavors

- Flavor is virtual hardware *template* for VM instance
- Defines sizes for memory, disk, number of cores, and so on



Flavors

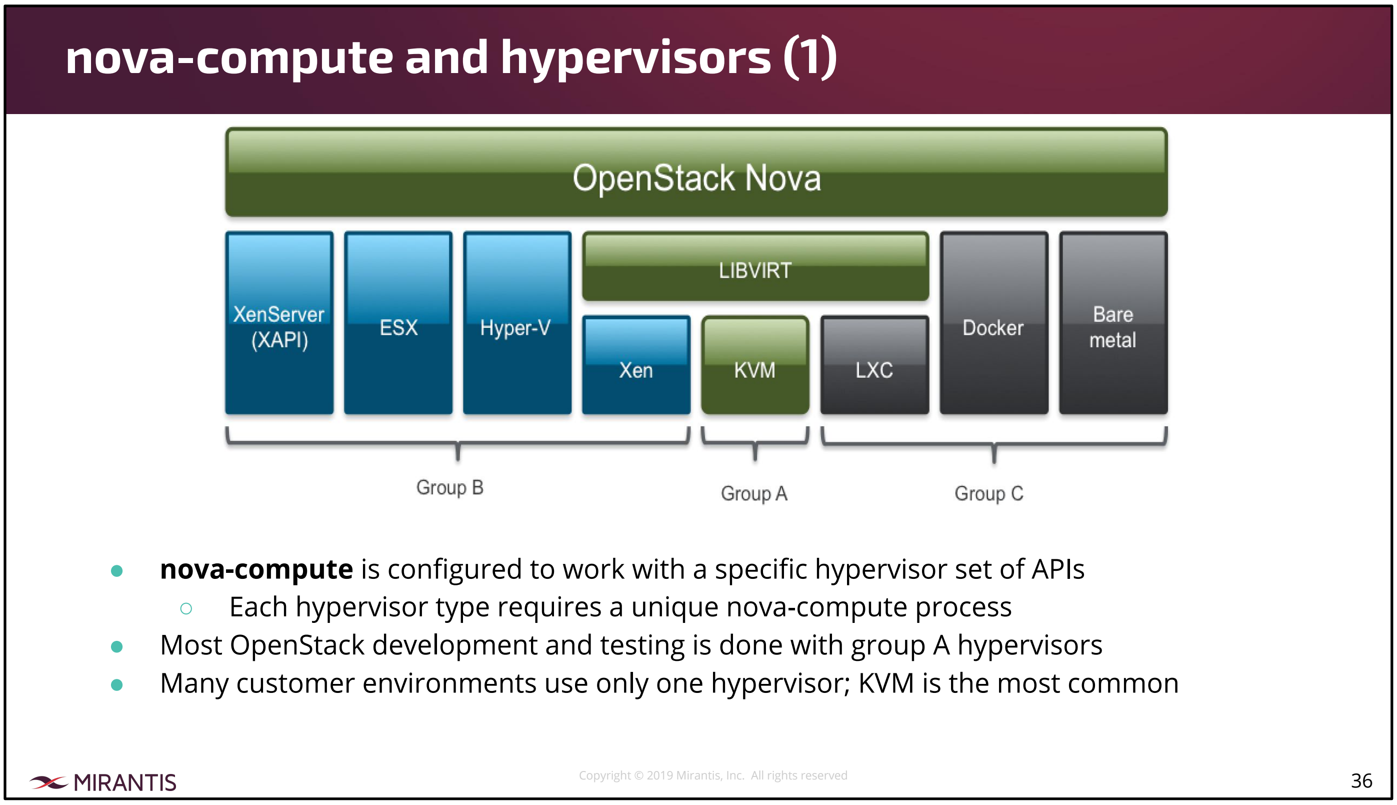
Flavor Name	VCPUs	RAM	Root Disk	Ephemeral Disk	Swap Disk	RX/TX factor	ID	Public	Metadata	Actions
m1.amphora	1	1GB	2GB	0GB	0MB	1.0	d33c9470-7297-4b2c-863d-6caf930f4f25	No	No	Edit Flavor
m1.large	4	8GB	80GB	0GB	0MB	1.0	4	Yes	No	Edit Flavor
m1.medium	2	4GB	40GB	0GB	0MB	1.0	3	Yes	No	Edit Flavor
m1.small	1	2GB	20GB	0GB	0MB	1.0	2	Yes	No	Edit Flavor
m1.tiny	1	512MB	1GB	0GB	0MB	1.0	1	Yes	No	Edit Flavor
m1.xlarge	8	16GB	160GB	0GB	0MB	1.0	5	Yes	No	Edit Flavor

Displaying 6 items

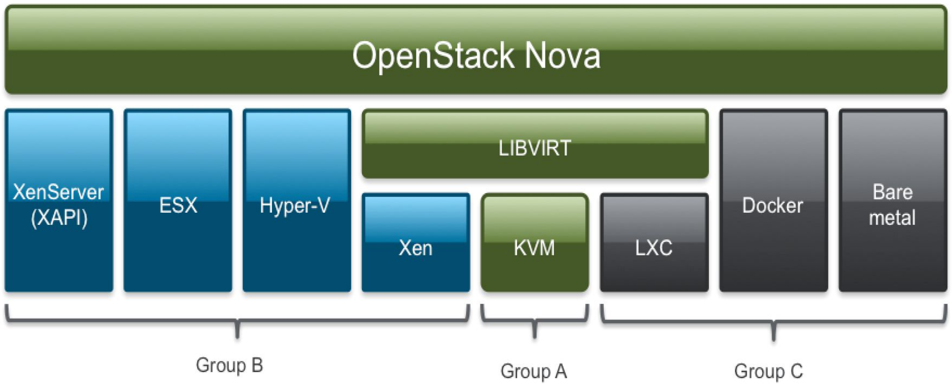
- Administrator can create custom flavors
  - For example, 1GB RAM + 10GB disk + 2 vCPUs

MIRANTIS

The flavor is a combination of characteristics (for example, memory, disk, vCPU) that are used when deploying a VM.
For example, if you specify the *m1.tiny* flavor on a create instance request, the VM will have 1 vCPU, 512MB memory, and a 1GB disk when it is deployed.
This slide shows several flavors that are defined for OpenStack. You can create new flavors if you have the admin role..
Most flavors are ***public***. That is, they can be used by all projects in all tenants. You can define ***private*** flavors that can only be used by specific projects and tenants. For example, the m1.amphora flavor is private (public =no).



nova-compute and hypervisors (1)



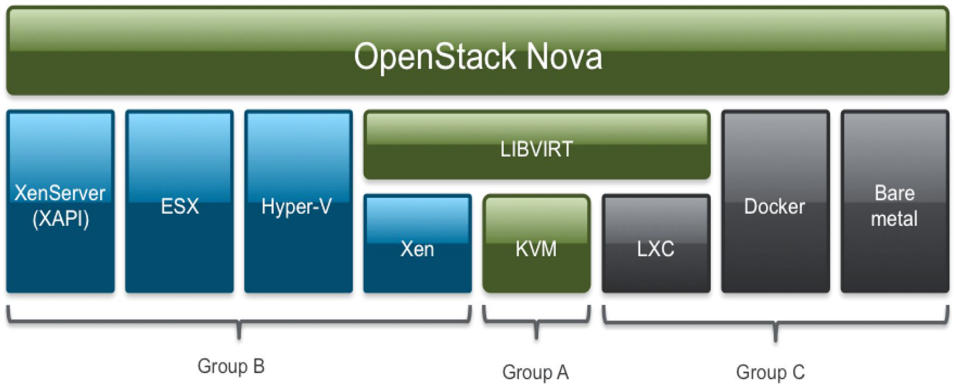
- **nova-compute** is configured to work with a specific hypervisor set of APIs
  - Each hypervisor type requires a unique nova-compute process
- Most OpenStack development and testing is done with group A hypervisors
- Many customer environments use only one hypervisor; KVM is the most common

MIRANTIS 36

Hypervisor: a piece of computer software or hardware that creates and run virtual machines. A system on which one or more virtual machines is defined is referred as host machine.
Definitions from Wikipedia:

- Kernel-based Virtual Machine (**KVM**) is a virtualization infrastructure for the Linux kernel that *turns it into a hypervisor*, using QEMU.
- **QEMU** (short for Quick Emulator) is a free and open-source hosted hypervisor that performs hardware virtualization (not to be confused with hardware-assisted virtualization).
- VMware ESXi (formerly **ESX**) is an enterprise-class, type-1 hypervisor developed by VMware for deploying and serving virtual computers. As a type-1 hypervisor, ESXi is not a software application that one installs in an operating system (OS); instead, it includes and integrates vital OS components, such as a kernel.
- **LXC** (Linux Containers, through libvirt) is an operating-system-level virtualization method for running multiple isolated Linux systems (containers) on a control host using a single Linux kernel; managed through libvirt.
- Microsoft **Hyper-V**, formerly known as Windows Server Virtualization, is a native hypervisor; it can create virtual machines on x86-64 systems running Windows.
- **XEN** is a hypervisor using a microkernel design, providing services that allow multiple computer operating systems to execute on the same computer hardware concurrently.
- Citirix **XenServer** is a hypervisor platform that enables the creation and management of virtualized server infrastructure. It is developed by Citrix Systems and is built over the Xen virtual machine hypervisor. **XenServer** provides server virtualization and monitoring services.
- **User-mode Linux** (UML) enables multiple virtual **Linux** kernel-based operating systems (known as guests) to run as an application within a normal **Linux** system (known as the host). Typically, this is only used in a development environment.
- **Ironic** is an OpenStack project which provisions bare metal (as opposed to virtual) machines. It uses technologies, such as Preboot Execution Environment (PXE), Network Bootstrap Program (NBP), Intelligent Platform Management Interface (IPMI), and others.

nova-compute and hypervisors (2)

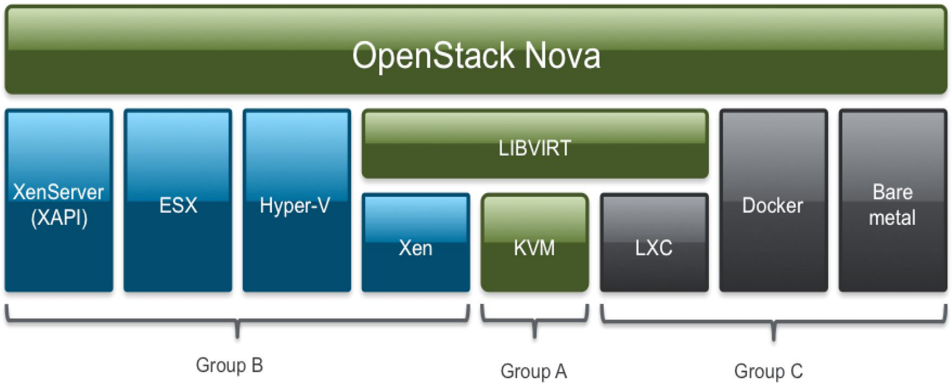


- **Group A: Fully tested and supported**
  - Most OpenStack development and testing is done with group A hypervisors
- **Group B:** *middle ground*
  - functional testing providing by an external system that does not gate commits, but advises patch authors and reviewers of results
- **Group C:** minimal testing; may or may not work at any given time. Use them at your own risk.

MIRANTIS

For more details:
https://wiki.openstack.org/wiki/HypervisorSupportMatrix#Driver_Testing_Status

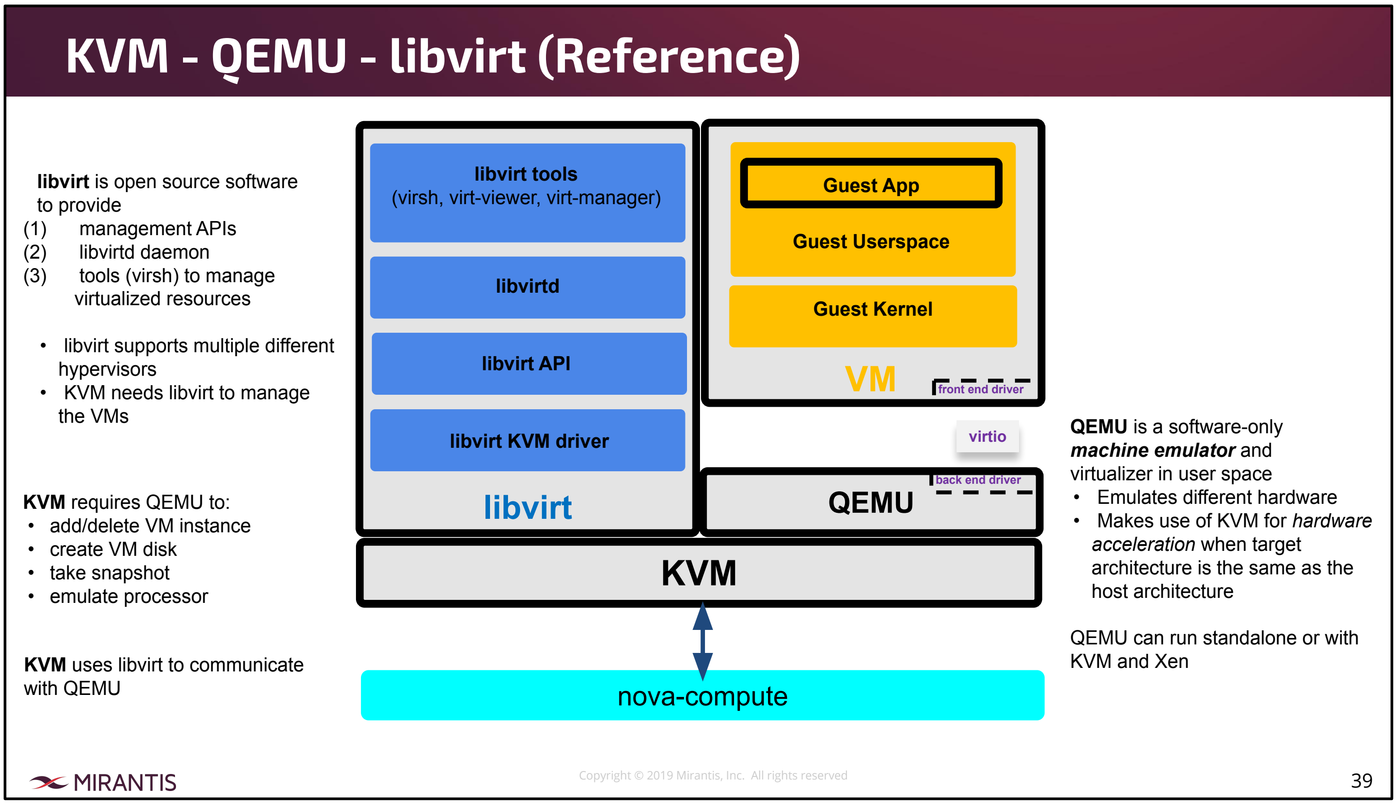
nova-compute and hypervisors (3)



- If you need to support multiple hypervisors:
  - Place each hypervisor in a different compute node
  - Optionally, use filters (ComputeFilter and ImagePropertiesFilter) or image metadata to select the hypervisor at deploy time

MIRANTIS

This slide is intended for reference only; for users requiring information running more than 1 hypervisor. Historically, most OpenStack development is done with the KVM hypervisor with which you are more likely to find community support for issues . All features that are currently supported in KVM are also supported in QEMU. To verify the proper list of supported hypervisors, check:
https://wiki.openstack.org/wiki/HypervisorSupportMatrix
https://docs.openstack.org/nova/rocky/admin/arch.html#hypervisors



KVM - QEMU - libvirt (Reference)

libvirt is open source software to provide
(1) management APIs
(2) libvirtd daemon
(3) tools (virsh) to manage virtualized resources

- libvirt supports multiple different hypervisors
- KVM needs libvirt to manage the VMs

KVM requires QEMU to:
- add/delete VM instance
- create VM disk
- take snapshot
- emulate processor

KVM uses libvirt to communicate with QEMU

QEMU is a software-only ***machine emulator*** and virtualizer in user space
- Emulates different hardware
- Makes use of KVM for *hardware acceleration* when target architecture is the same as the host architecture

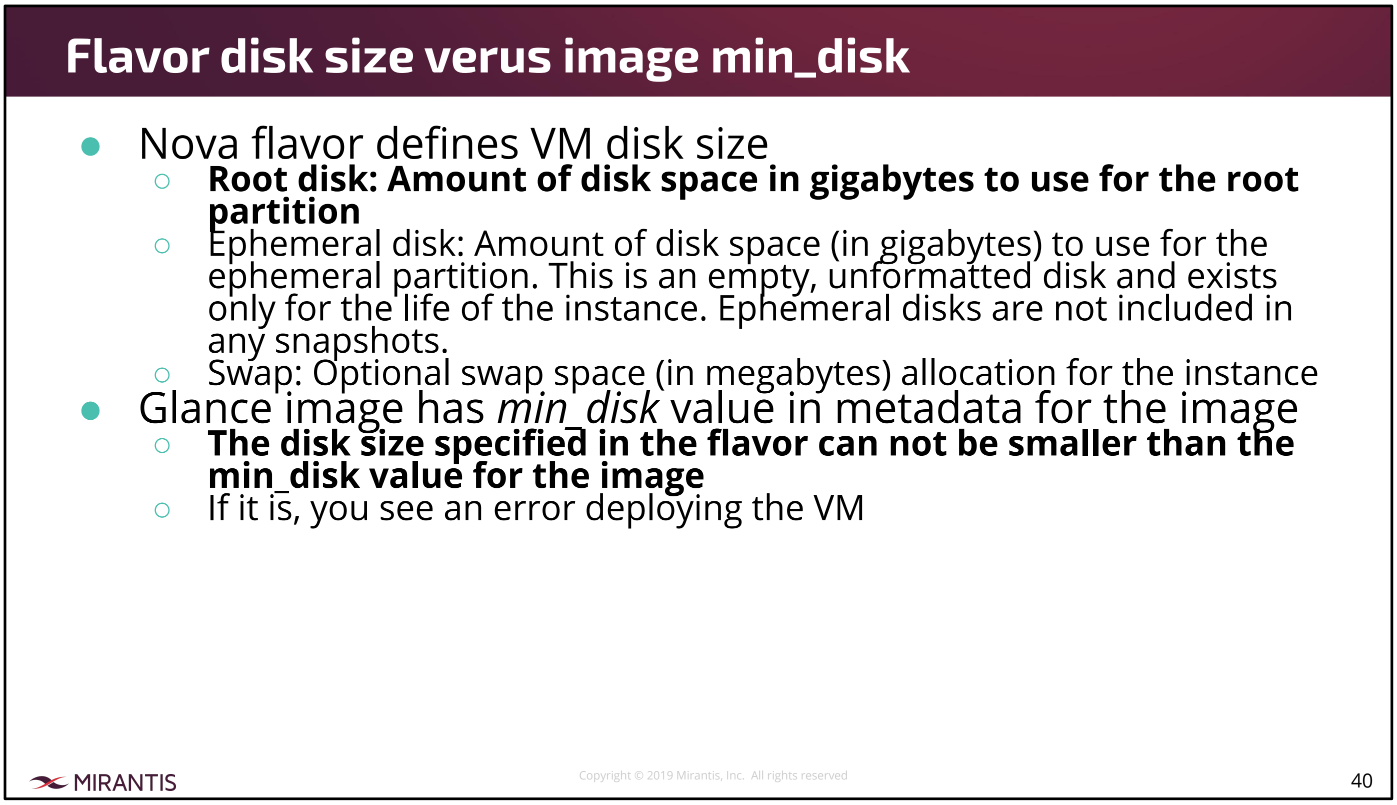
QEMU can run standalone or with KVM and Xen

MIRANTIS

Although it is considered a hypervisor, KVM is not. Instead, KVM provides hardware acceleration for virtualized environments.
When people say that KVM is their hypervisor, they mean that they are also running:
- Quick emulator (QEMU): Providing the actual hypervisor functions
- libvirt: Tools to deploy VMs to hypervisors, such as QEMU

In fact, KVM requires that QEMU and libvirt are also installed. The 3 products (together) provide the hypervisor environment.

Summary:
- KVM provides the hardware acceleration
- virtio provides a better performing I/O framework
- QEMU provides the actual hypervisor functions
- libvirt provides the management tools



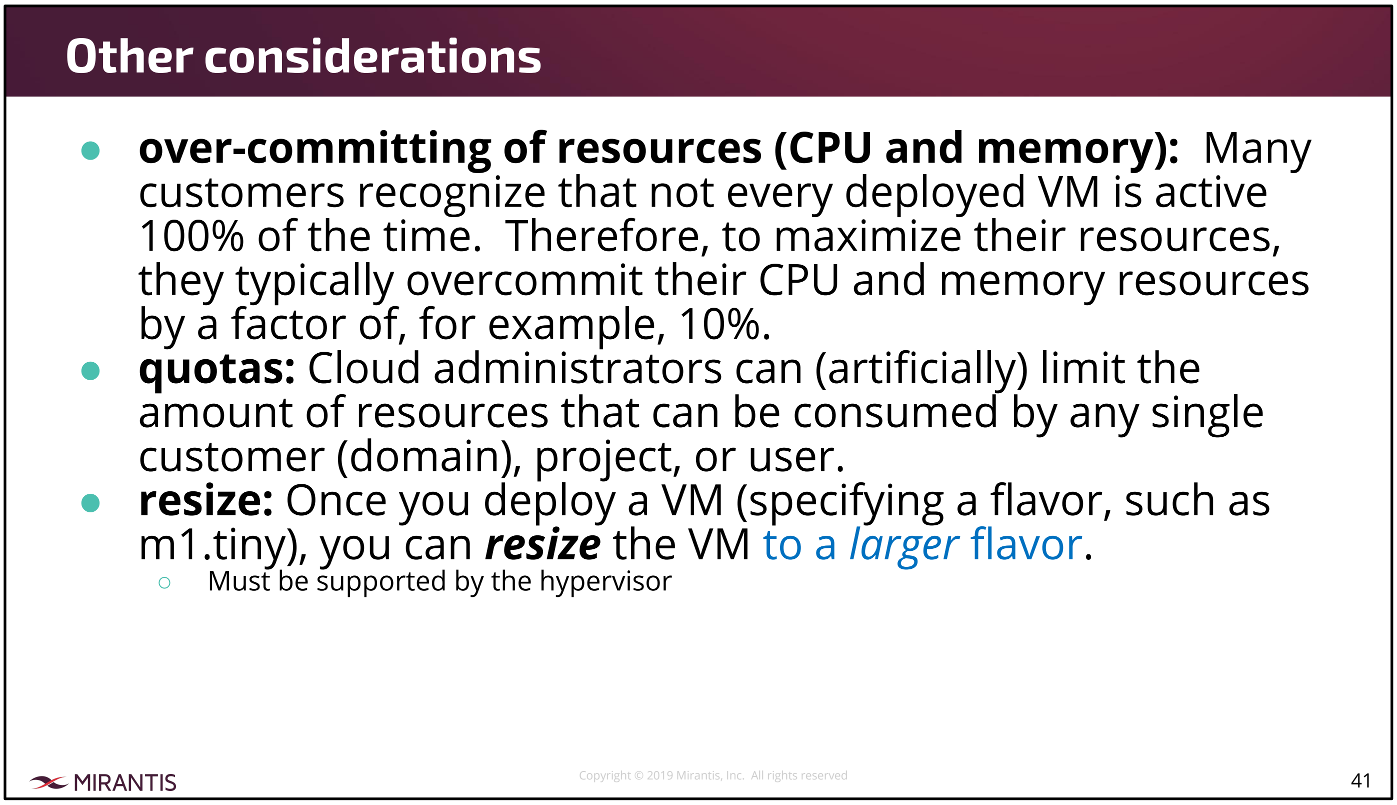
Flavor disk size verus image min_disk

- Nova flavor defines VM disk size
  - **Root disk: Amount of disk space in gigabytes to use for the root partition**
  - Ephemeral disk: Amount of disk space (in gigabytes) to use for the ephemeral partition. This is an empty, unformatted disk and exists only for the life of the instance. Ephemeral disks are not included in any snapshots.
  - Swap: Optional swap space (in megabytes) allocation for the instance
- Glance image has *min_disk* value in metadata for the image
  - **The disk size specified in the flavor can not be smaller than the min_disk value for the image**
  - If it is, you see an error deploying the VM

MIRANTIS

When you deploy a virtual machine you specify the flavor as a parameter on the nova boot command. If the disk size for the flavor is smaller than the min_disk for the image, you see an error. You must choose a flavor with a disk size that is larger than the min_disk for the image.
If you are deploying the instance from the Horizon dashboard UI, you will see an indicator that you cannot select a flavor if its disk size is smaller than the min_disk value for the image.

Consider this: Suppose you have an image with a min_disk value of 5GB. You cannot use the m1.tiny flavor to deploy a VM with the image. The disk size specified in m1.tiny is only 1GB. You need to use the m1.small flavor (20GB disk). However, you are deploying a VM with too much disk space! Therefore, you can consume disk space more quickly. This example shows the need for a custom flavor with a smaller disk allocation. For example, 10GB (or even less). The difference (10GB) is not huge, but when you multiply it over thousands of VMs, it adds up quickly.


Other considerations

- **over-committing of resources (CPU and memory):** Many customers recognize that not every deployed VM is active 100% of the time. Therefore, to maximize their resources, they typically overcommit their CPU and memory resources by a factor of, for example, 10%.
- **quotas:** Cloud administrators can (artificially) limit the amount of resources that can be consumed by any single customer (domain), project, or user.
- **resize:** Once you deploy a VM (specifying a flavor, such as m1.tiny), you can ***resize*** the VM to a *larger* flavor.
  - Must be supported by the hypervisor

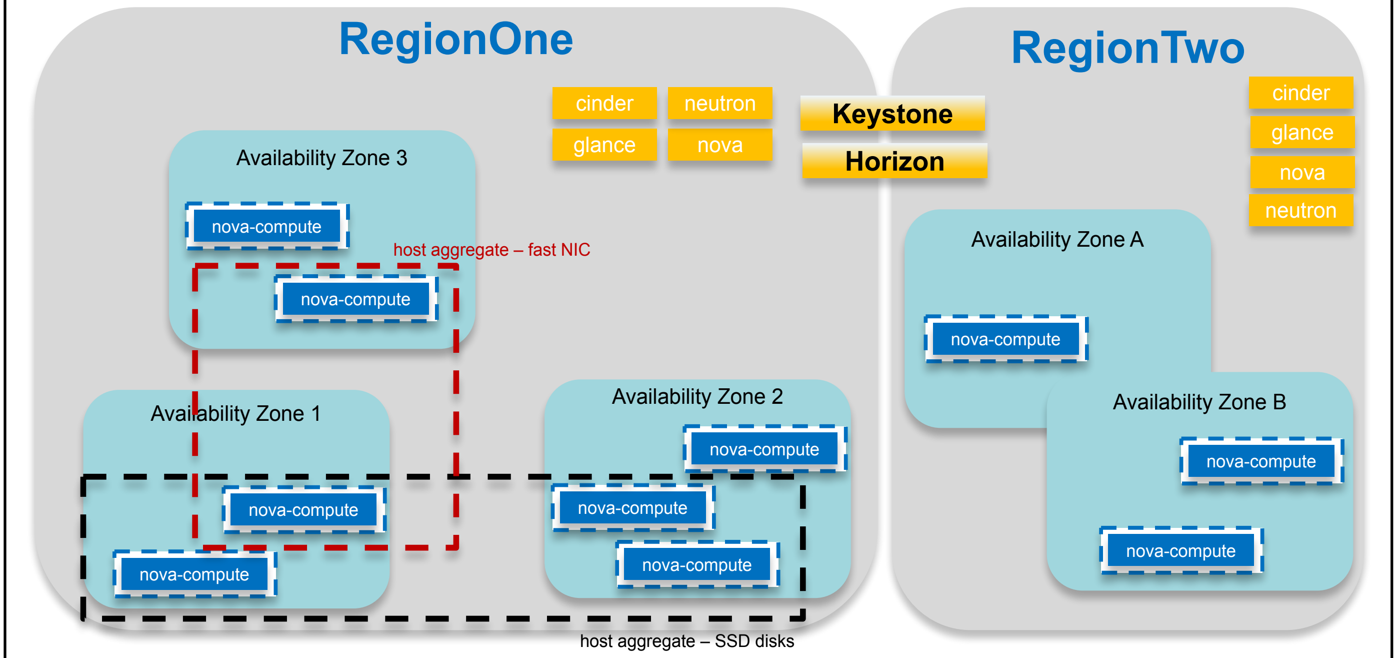
MIRANTIS

41

Other important topics to consider:

- **over-committing of resources (CPU and memory):** Many customers recognize that not every deployed VM is active 100% of the time. Therefore, to maximize their resources, they typically overcommit their CPU and memory resources by a factor of, for example, 10%. This approach is especially true when the customers are involved in a charge-back mechanism where they are paying for the cloud resources. In that case, they do not want to pay for resources that are not *running*. Typically, they are charged for a smaller fee for the resources allocated and another fee for the amount of time the resource is active. There are 2 properties that can be added to the **nova.conf** file to support this:
  - Cpu_allocation_ratio
  - ram_allocation_ratio
- **quotas:** Cloud administrators can (artificially) limit the amount of resources that can be consumed by any single tenant, project (team) or user; preventing any of the available resources being consumed; thus limiting the amount of resources available for the remaining tenants, projects, or users.
- **resize:** After deploying a VM (specifying a flavor, such as m1.tiny), you might need to make it larger. You can ***resize*** the VM using a bigger flavor.
  - You might need to update **nova.conf** with: **allow_resize_to_same_host = True**

Availability zones and regions



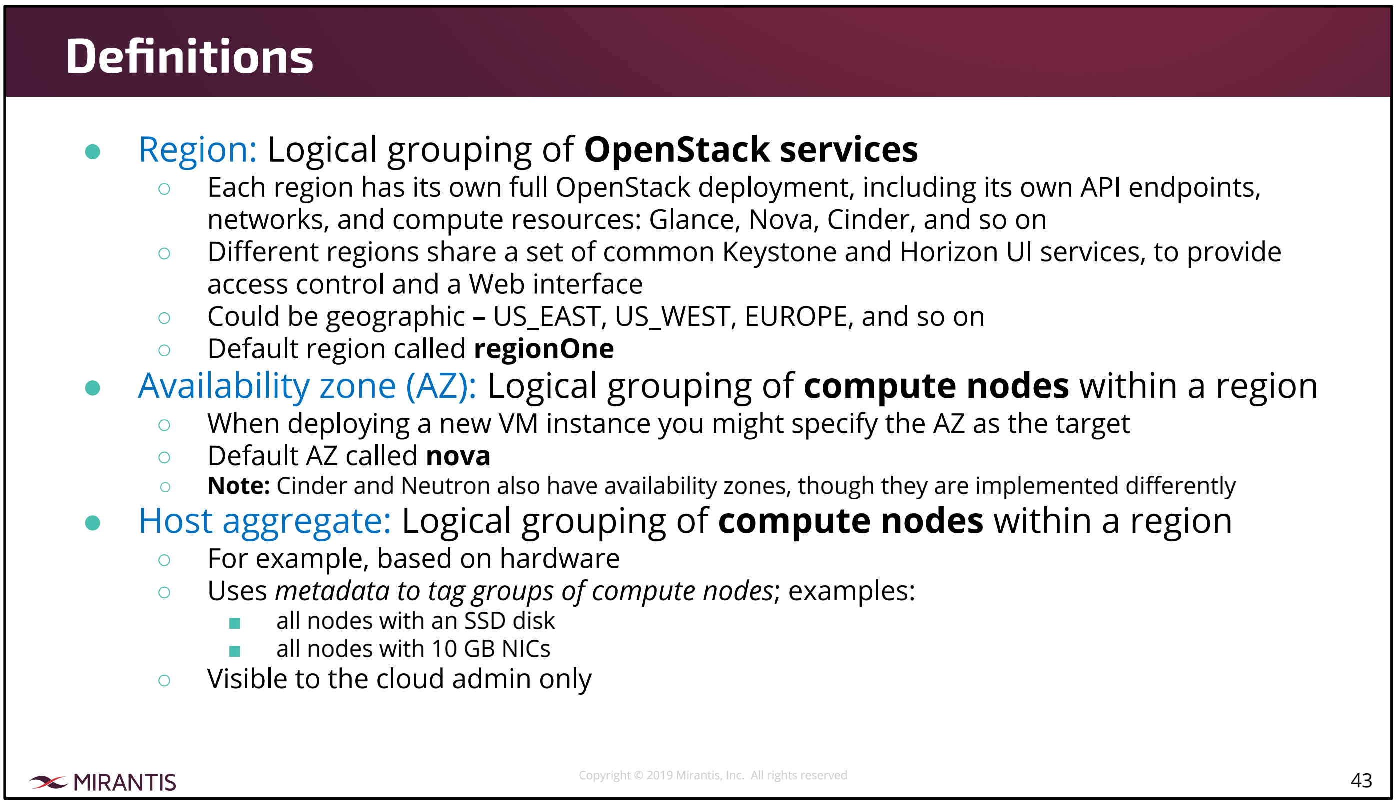
MIRANTIS

OpenStack clouds can be divided (segregated) in to three main hierarchical groupings: Regions, Availability Zones and Host Aggregates.
The segregation of OpenStack services enables OpenStack to support massive horizontal scaling. Most OpenStack implementations do not require such a massive scale, although the use of regions, availability zones, and host aggregates is common in smaller implementations.
A region is a grouping of OpenStack services, typically grouped geographically. Without considering HA, there can only be 1 Keystone and 1 Dashboard UI, regardless of the number of regions.
An AZ is a grouping of nova-compute nodes. For example, AZ3 might be KVM, whereas AZ1 might be VMware.
When you deploy an instance, you have the option of specifying which AZ to use.

This slide illustrates OpenStack regions, availability zones, and host aggregates. By default, there is 1 availability zone (nova) and 1 region (RegionOne).
In this example, there are two regions: RegionOne and RegionTwo, with a total of 10 hosts (nova-compute nodes).
RegionOne has three availability zones: AZ 1 (2 hosts), AZ 2 (3 hosts), AZ 3 (2 hosts)
RegionTwo has two availability zones: AZ A (1 host), AZ B (2 hosts)

Host Aggregates: (beyond the scope of this class) RegionOne also has 2 host aggregates defined:
- One is for SSD disks (black dashed box)
- One for fast NICs (red dashed box)

Host aggregates are logical groupings of **compute nodes** within a region. The hosts in the host aggregate might span availability zones within the region. Notice, for example, there is one nova-compute (host) that intersects both of the host aggregates.



Definitions

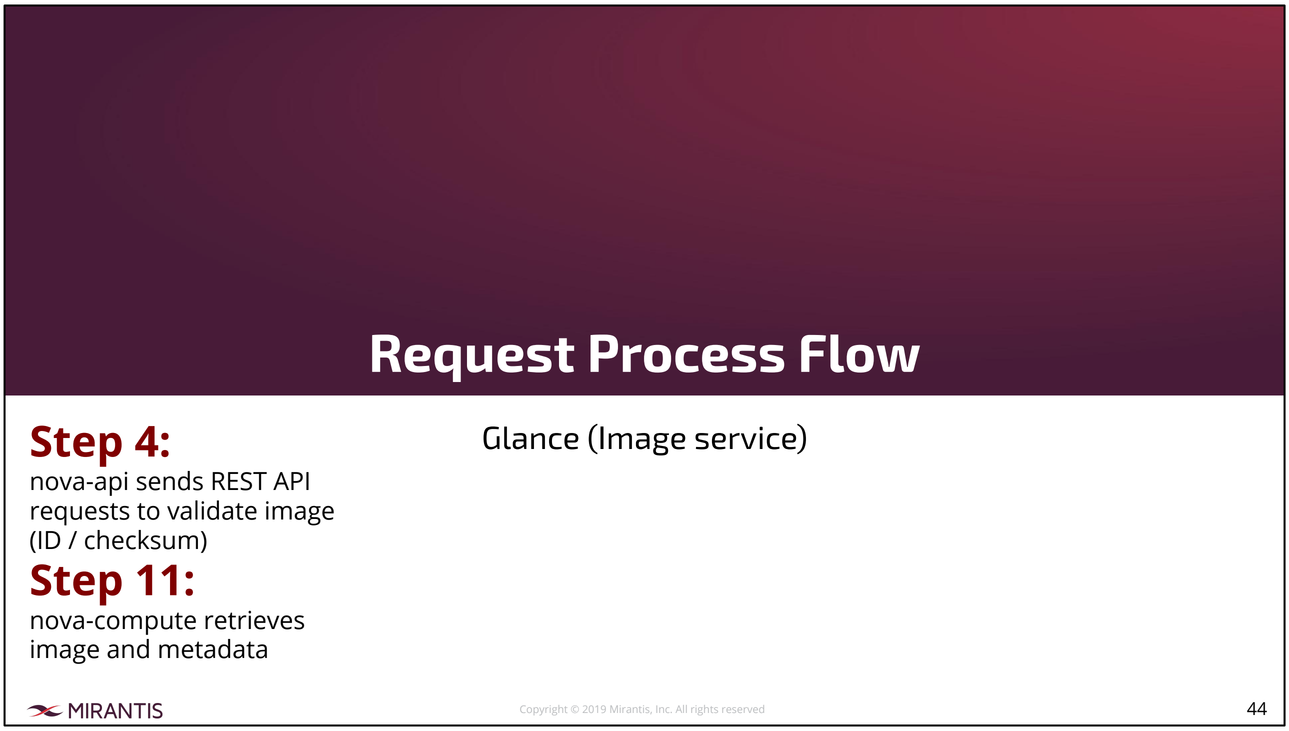
- Region: Logical grouping of **OpenStack services**
  - Each region has its own full OpenStack deployment, including its own API endpoints, networks, and compute resources: Glance, Nova, Cinder, and so on
  - Different regions share a set of common Keystone and Horizon UI services, to provide access control and a Web interface
  - Could be geographic – US_EAST, US_WEST, EUROPE, and so on
  - Default region called **regionOne**
- Availability zone (AZ): Logical grouping of **compute nodes** within a region
  - When deploying a new VM instance you might specify the AZ as the target
  - Default AZ called **nova**
  - **Note:** Cinder and Neutron also have availability zones, though they are implemented differently
- Host aggregate: Logical grouping of **compute nodes** within a region
  - For example, based on hardware
  - Uses *metadata to tag groups of compute nodes*; examples:
    - all nodes with an SSD disk
    - all nodes with 10 GB NICs
  - Visible to the cloud admin only

MIRANTIS

Regions segregate completely independent installations linked by a single Identity (Keystone) and Dashboard (Horizon, optional) installation. Regions allow you to group a set of OpenStack services. Put another way, *regions are autonomous OpenStack clouds joined together through a shared Keystone identity server and managed through a common interface (Dashboard UI)*. A shared identity server provides a consolidated authentication engine across all regions. A common UI, such as the Horizon Dashboard, provides a single UI to manage your cloud, regardless of the number of regions.

Region examples:
Regions might be geographically organized (not required) for ease of operations.
Regions might be built for development, test, and production clouds.
Regions might be used for high availability of resources (not common).
Regions might be used to support multiple releases of OpenStack.

A compute node might belong to both one or more Host Aggregates and an Availability Zone at the same time. You might use host aggregates to further partition an availability zone, for example.



Request Process Flow

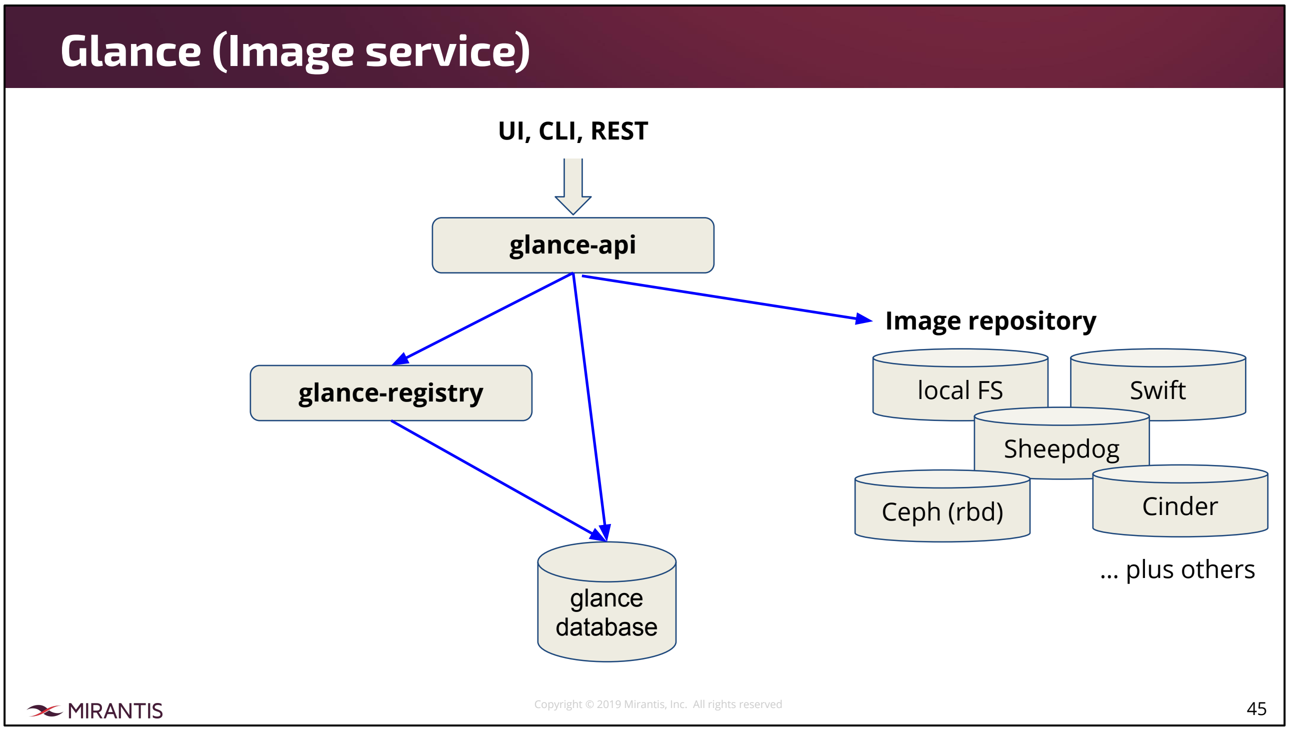
Step 4:
nova-api sends REST API requests to validate image (ID / checksum)

Step 11:
nova-compute retrieves image and metadata

Glance (Image service)

MIRANTIS

The Image service (Glance) enables users to discover, register, and retrieve virtual machine images. It offers a REST API that enables you to query virtual machine image metadata as well as retrieval of the images. Glance provides the images necessary to deploy virtual machine instances. You can store virtual machine images in a variety of locations, from simple file systems to object-storage systems like OpenStack Object Storage (Swift).



Glance (Image service)

MIRANTIS

Glance has several main components:

- glance-api:accepts image API calls for image creation (and storage), update, and retrieval
- glance-registry:stores, processes, and retrieves ***image metadata*** (size, type, hypervisor, and so on)
- database: stores image details, as well as, the image metadata
- image repository:storage repository, stores the actual image files

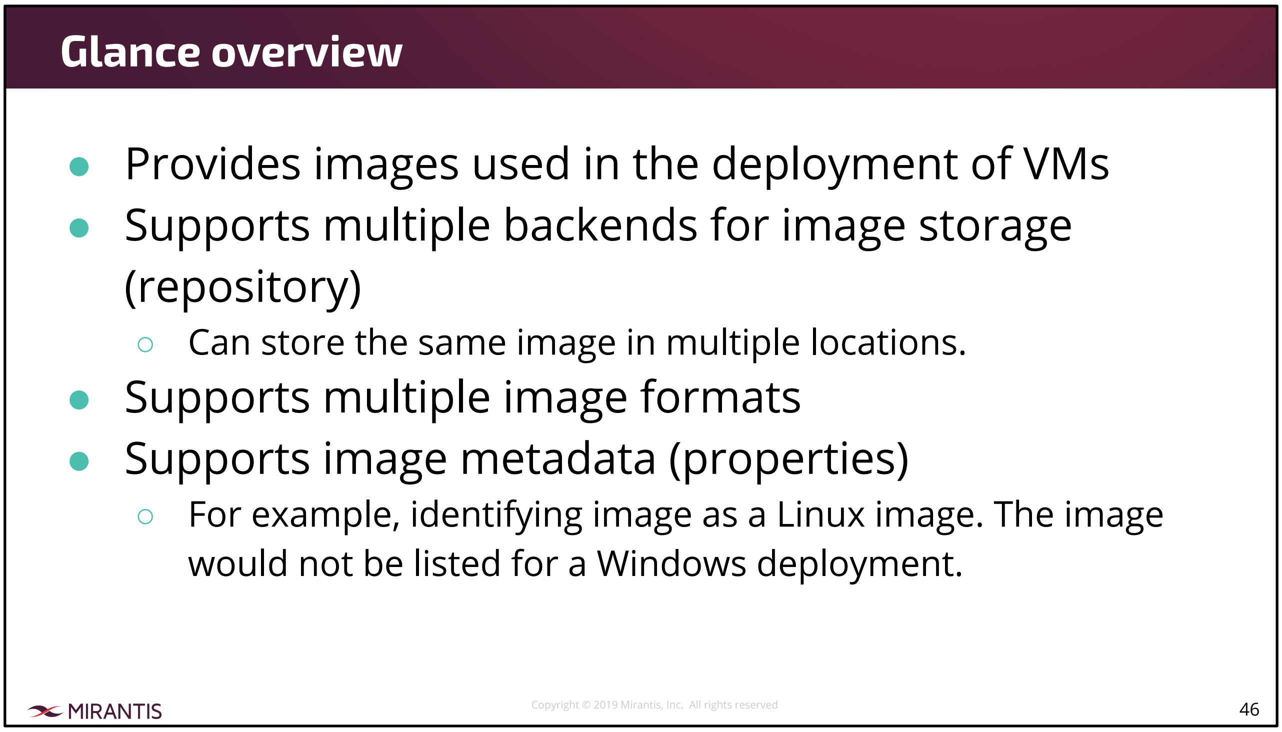
Step 4:

- nova-api performs a REST call to validate the image ID, passing the **auth-token** to glance-api.
- glance-api validates the **auth-token** with keystone.
- glance-api validates the chksum for the image (in the database); responds: *image valid*

Step 11:

- Later in the process, nova-compute uses the Image ID to retrieve the image URI from the Image Service, and loads the image from the image storage.
- nova-compute then retrieves the image metadata.

There are also several periodic processes running on Glance to support (internal) caching



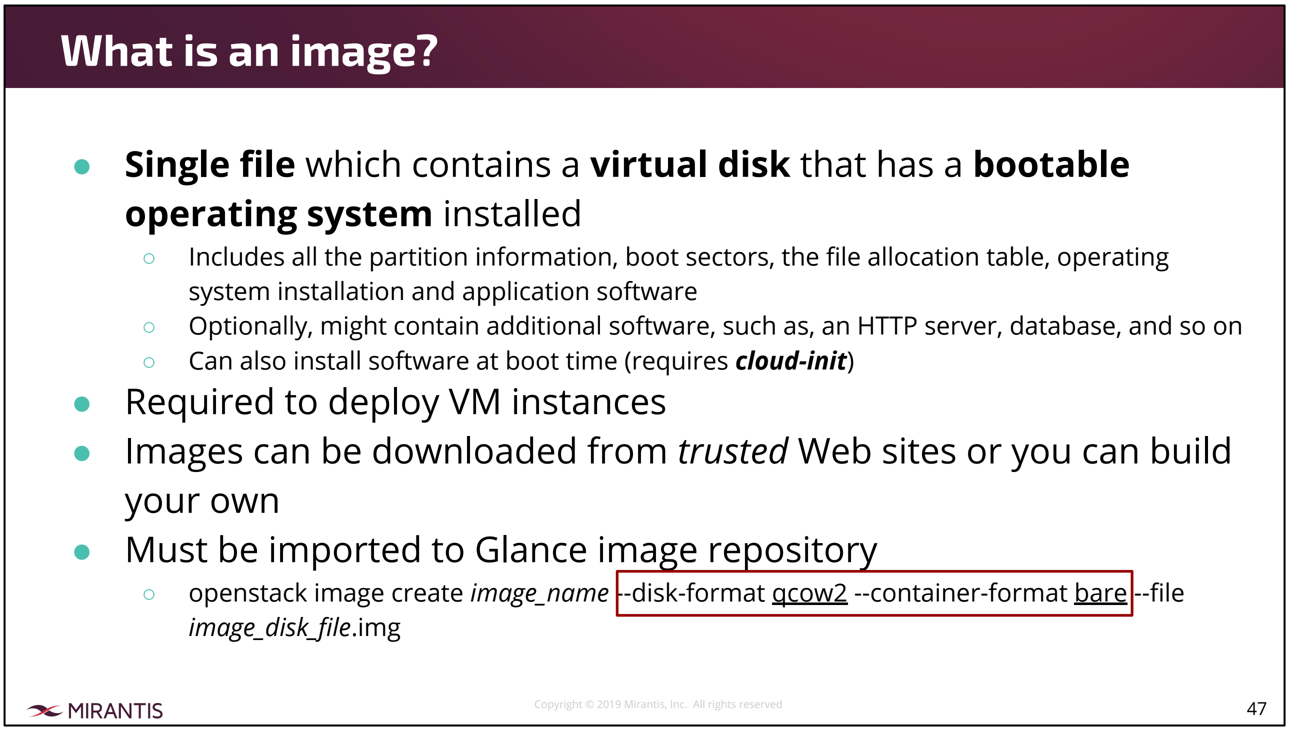
Glance overview

- Provides images used in the deployment of VMs
- Supports multiple backends for image storage (repository)
  - Can store the same image in multiple locations.
- Supports multiple image formats
- Supports image metadata (properties)
  - For example, identifying image as a Linux image. The image would not be listed for a Windows deployment.

MIRANTIS

Notes:

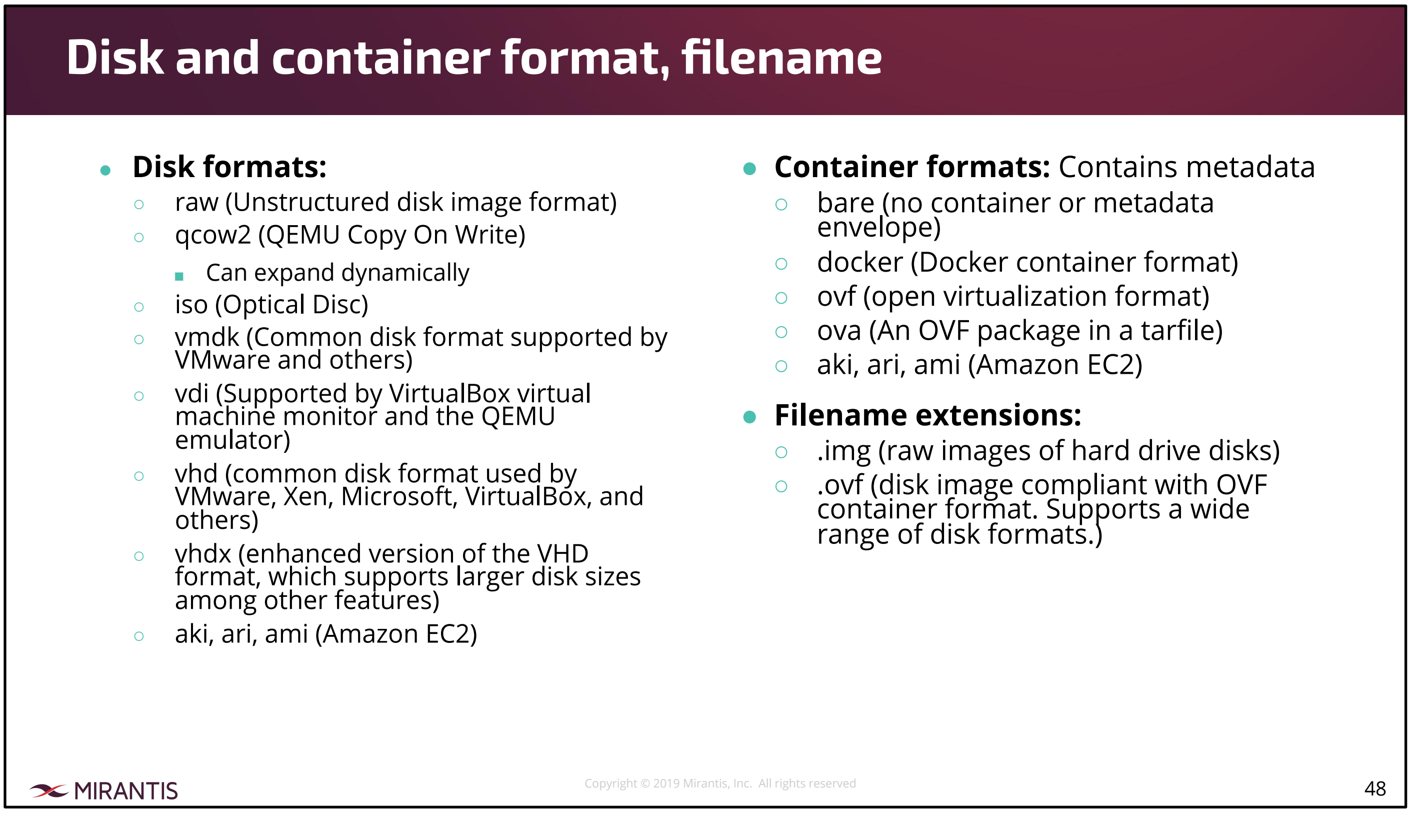
- Functionality provided by backends is not 100% similar.
- Exact Glance capabilities available depend on the backend and Hypervisor used.
- Most of the features are developed and tested with the file system or Swift.
- OpenStack does not use image metadata. However, you can define additional properties on an image that are used, such as, hypervisor type, OS type, and more.



What is an image?

- **Single file** which contains a **virtual disk** that has a **bootable operating system** installed
  - Includes all the partition information, boot sectors, the file allocation table, operating system installation and application software
  - Optionally, might contain additional software, such as, an HTTP server, database, and so on
  - Can also install software at boot time (requires ***cloud-init***)
- Required to deploy VM instances
- Images can be downloaded from *trusted* Web sites or you can build your own
- Must be imported to Glance image repository
  - openstack image create *image_name* --disk-format qcow2 --container-format bare --file *image_disk_file*.img

A Glance image is a single file that contains a disk with a bootable operating system. Images are used in the deployment of virtual machine instances.
The image might also contain additional software. If it does, the software is *baked in* to the image.
You can create your own images or you can download images from several vendor-provided web sites. In either case, you must import the image to the Glance image repository using the openstack image create command or its equivalent from the UI. When the image is imported, its image metadata is stored in the OpenStack (for example, MySQL) database.



Disk and container format, filename

- **Disk formats:**
  - raw (Unstructured disk image format)
  - qcow2 (QEMU Copy On Write)
    - Can expand dynamically
  - iso (Optical Disc)
  - vmdk (Common disk format supported by VMware and others)
  - vdi (Supported by VirtualBox virtual machine monitor and the QEMU emulator)
  - vhd (common disk format used by VMware, Xen, Microsoft, VirtualBox, and others)
  - vhdx (enhanced version of the VHD format, which supports larger disk sizes among other features)
  - aki, ari, ami (Amazon EC2)
- **Container formats:** Contains metadata
  - bare (no container or metadata envelope)
  - docker (Docker container format)
  - ovf (open virtualization format)
  - ova (An OVF package in a tarfile)
  - aki, ari, ami (Amazon EC2)
- **Filename extensions:**
  - .img (raw images of hard drive disks)
  - .ovf (disk image compliant with OVF container format. Supports a wide range of disk formats.)

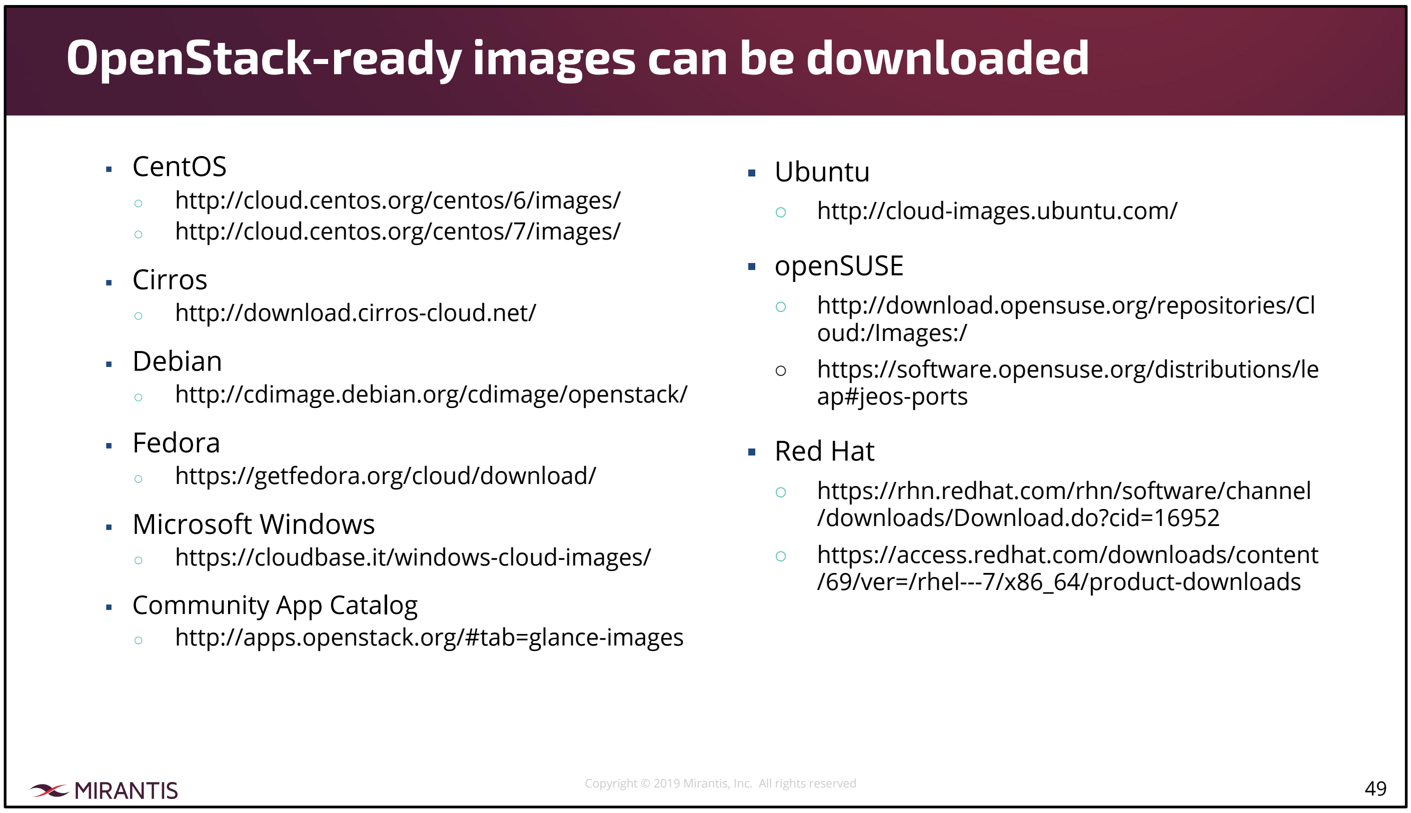
MIRANTIS

When you create an image, Glance:

- Imports the image file to the image repository (for example, Swift or the local file system)
- Creates the image metadata in the metadata repository (database). Image metadata comes from the **openstack image create** command parameters. If you specify a **container format** other than *bare*, additional metadata will be retrieved from the ovf, docker, aki, ari, or ami files.

The qcow2 (QEMU copy on write) disk image format uses the copy-on-write technique to reduce disk image size. The original storage is never modified. When a write request is made, it is redirected away from the original data into a new storage area.

<u>Acronyms:</u>
vhd: virtual hard disk
vdi: virtual disk image – None of the OpenStack Compute hypervisors support VDI directly, so you will need to convert these files to a different format to use them with OpenStack.
vmdk: virtual machine disk
Amazon EC2 image = aki (Amazon kernel image) + ari (Amazon ramdisk image) + ami (Amazon machine image); sometimes packaged as a Ubuntu Enterprise Cloud (UEC) tarball (support is deprecated in later releases).
You can also use tools, such as **VBoxManage** or **qemu-image convert**, to convert between the supported formats.
Note: raw images do not support snapshots.



OpenStack-ready images can be downloaded

- CentOS
  - http://cloud.centos.org/centos/6/images/
  - http://cloud.centos.org/centos/7/images/
- Cirros
  - http://download.cirros-cloud.net/
- Debian
  - http://cdimage.debian.org/cdimage/openstack/
- Fedora
  - https://getfedora.org/cloud/download/
- Microsoft Windows
  - https://cloudbase.it/windows-cloud-images/
- Community App Catalog
  - http://apps.openstack.org/#tab=glance-images
- Ubuntu
  - http://cloud-images.ubuntu.com/
- openSUSE
  - http://download.opensuse.org/repositories/Cloud:/Images:/
  - https://software.opensuse.org/distributions/leap#jeos-ports
- Red Hat
  - https://rhn.redhat.com/rhn/software/channel/downloads/Download.do?cid=16952
  - https://access.redhat.com/downloads/content/69/ver=/rhel---7/x86_64/product-downloads

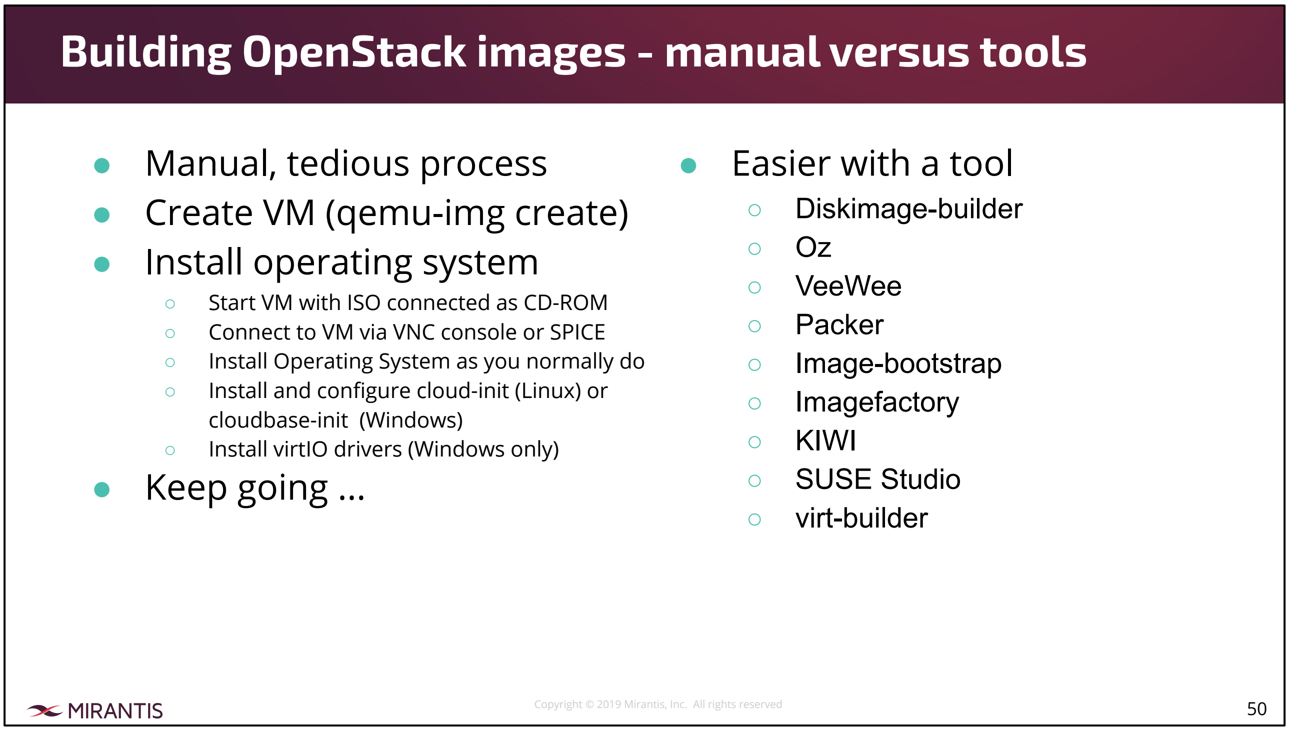
MIRANTIS

49

The simplest way to obtain a virtual machine image that works with OpenStack is to download one from a vendor-provided web site. Most of the images contain the *cloud-init* package to support the SSH key pair and user data injection.

Many vendors provide websites where you can download images. This slide lists the most common web pages to use for image downloads. For more details, see the *OpenStack image guide*: **http://docs.openstack.org/image-guide/obtain-images.html**

Caution: Not all of the images downloadable from these web sites contain **cloud-init**. For example, the Cirros images do not contain cloud-init.



Building OpenStack images - manual versus tools

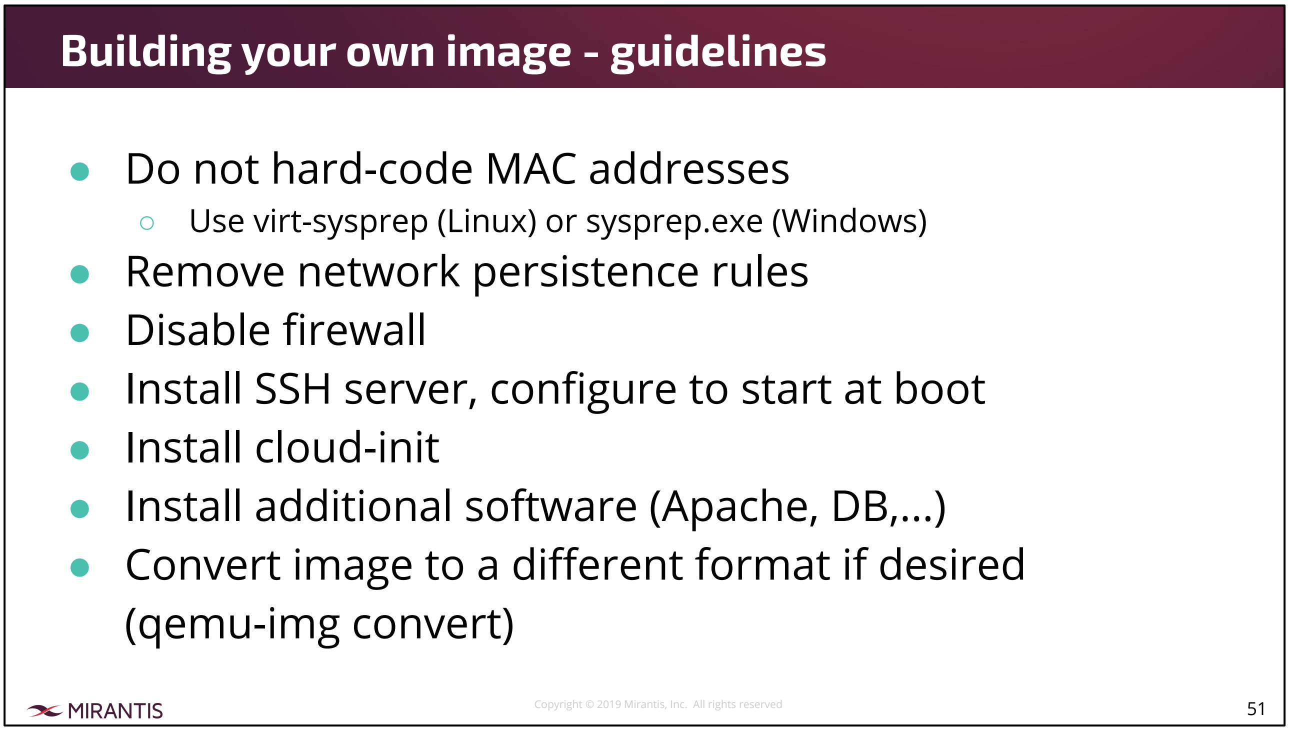
- Manual, tedious process
- Create VM (qemu-img create)
- Install operating system
  - Start VM with ISO connected as CD-ROM
  - Connect to VM via VNC console or SPICE
  - Install Operating System as you normally do
  - Install and configure cloud-init (Linux) or cloudbase-init (Windows)
  - Install virtIO drivers (Windows only)
- Keep going ...

- Easier with a tool
  - Diskimage-builder
  - Oz
  - VeeWee
  - Packer
  - Image-bootstrap
  - Imagefactory
  - KIWI
  - SUSE Studio
  - virt-builder

MIRANTIS

If you need to create a custom image, you can do it yourself (DIY) or you can use a tool, such as the ones listed on the slide. Tools automate the disk image creation process; supporting a variety of distributions and architectures. Check each tool for details on what they support.

To create a new image manually, you will need the installation (CD or DVD) ISO file for the guest operating system. You will also need access to a virtualization tool. You can use KVM for this. Or, if you have a GUI desktop virtualization tool (such as, VMware Fusion or VirtualBox), you can use that instead. The next slide contains more details as a reference.



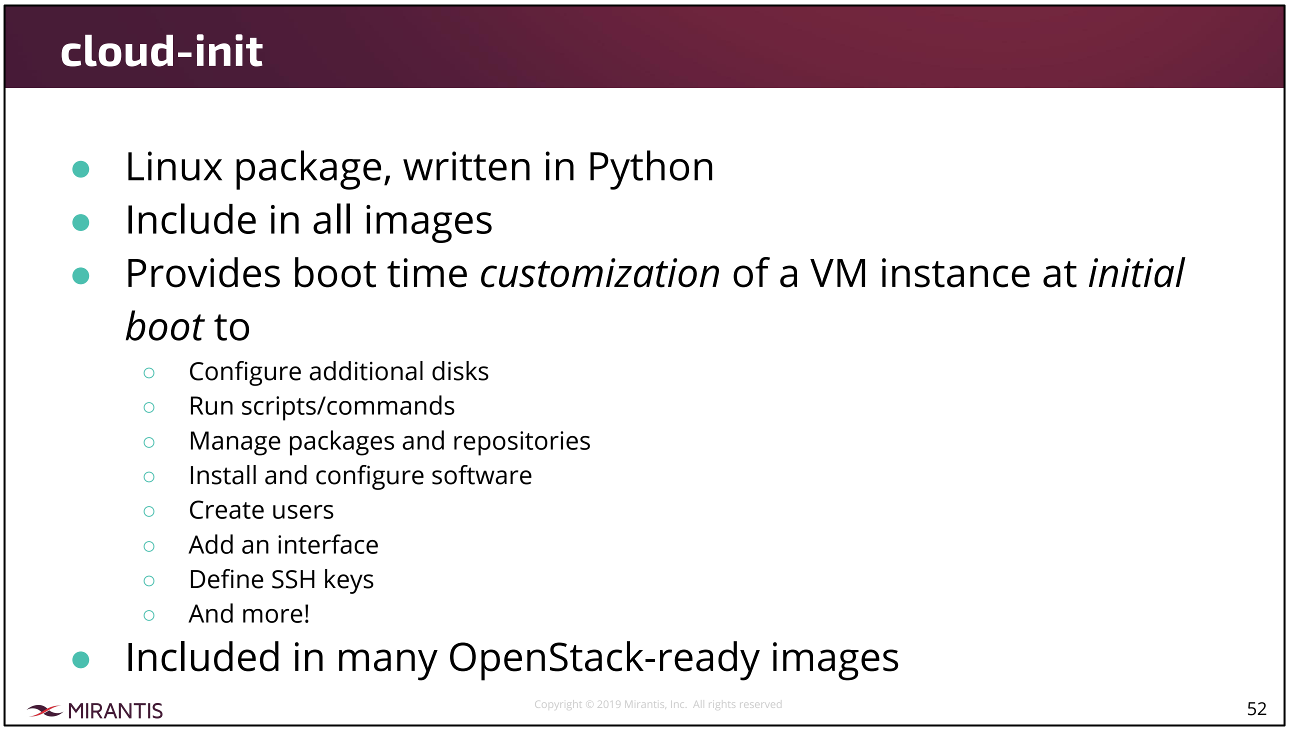
Building your own image - guidelines

- Do not hard-code MAC addresses
  - Use virt-sysprep (Linux) or sysprep.exe (Windows)
- Remove network persistence rules
- Disable firewall
- Install SSH server, configure to start at boot
- Install cloud-init
- Install additional software (Apache, DB,...)
- Convert image to a different format if desired (qemu-img convert)

MIRANTIS

Image guidelines – Overview: (notice the use of cloud-init for some)

- When you create a Linux image, you must decide how to partition the disks. The choice of partition method can affect the resizing functionality.
- You must remove the *network persistence rules* in the image because they might cause the network interface in the instance to come up as an interface other than eth0.
- You must install an ssh server into the image and ensure that it starts up on boot, or you cannot connect to your instance by using ssh when it boots inside of OpenStack.
- In general, OpenStack recommends that you disable any firewalls inside of your image and use OpenStack security groups to restrict access to instances.
- The typical way that users access virtual machines running on OpenStack is to ssh using public key authentication. For this to work, your virtual machine image must be configured to download the ssh public key from the OpenStack metadata service or config drive, at boot time.



cloud-init

- Linux package, written in Python
- Include in all images
- Provides boot time *customization* of a VM instance at *initial boot* to
  - Configure additional disks
  - Run scripts/commands
  - Manage packages and repositories
  - Install and configure software
  - Create users
  - Add an interface
  - Define SSH keys
  - And more!
- Included in many OpenStack-ready images

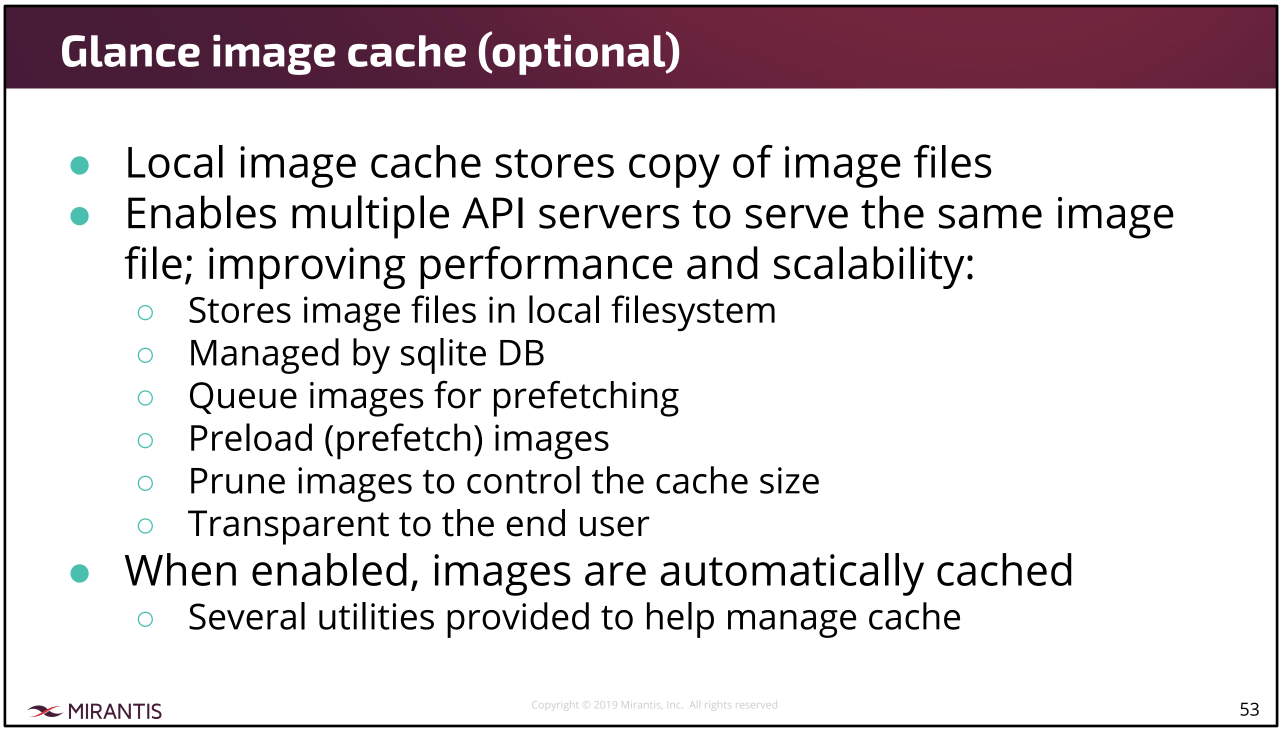
MIRANTIS

Cloud-init is a piece of software created to help with initializing virtual machines on multiple different cloud software platforms. It is a collection of Python scripts that run on a VM's first boot.

Examples of what you can do with cloud-init can be found at:
http://cloudinit.readthedocs.org/en/latest/topics/examples.html

Notes:

- cloud-init is not an OpenStack technology. Rather, it is a package that is designed to support multiple cloud providers, so that the same virtual machine image can be used in different clouds without modification.
- cloudbase-init is the equivalent package for Windows VMs.



Glance image cache (optional)

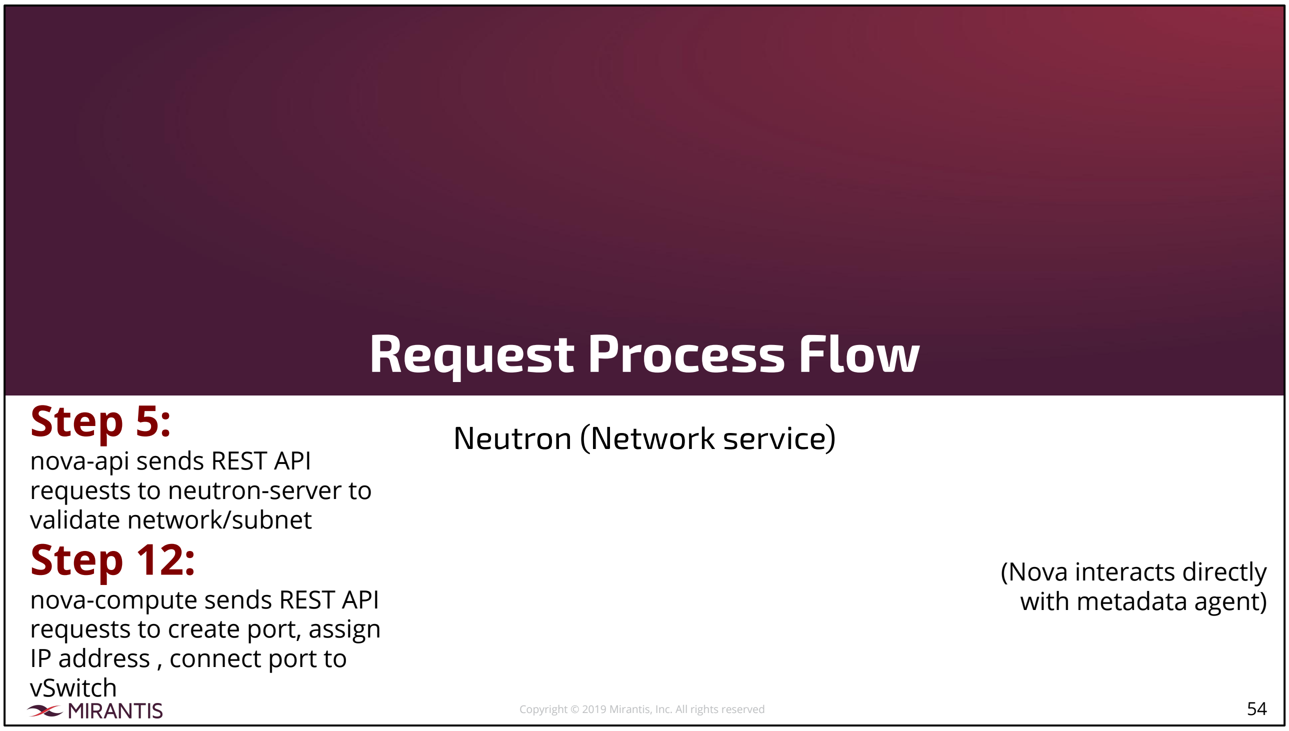
- Local image cache stores copy of image files
- Enables multiple API servers to serve the same image file; improving performance and scalability:
  - Stores image files in local filesystem
  - Managed by sqlite DB
  - Queue images for prefetching
  - Preload (prefetch) images
  - Prune images to control the cache size
  - Transparent to the end user
- When enabled, images are automatically cached
  - Several utilities provided to help manage cache

MIRANTIS 53

The Glance image cache enables multiple glance-api services to use the same image file, providing consistency of the images.
The **glance-cache.conf** file defines properties for image caching, including enabling it.
The **glance-api.conf** file needs to be updated to support image caching, as well.

Several utilities are provided:
glance-cache-pruner: helps control the size of the image cache.
glance-cache-cleaner: removes stalled or invalid images.
glance-cache-manage: lists or deletes the images in the cache. Can also be used to preload (prefetch) images in the cache. For example, many customers preload their golden/master images.

For more information:
http://docs.openstack.org/developer/glance/cache.html



Request Process Flow

Step 5:
nova-api sends REST API requests to neutron-server to validate network/subnet

Step 12:
nova-compute sends REST API requests to create port, assign IP address , connect port to vSwitch

Neutron (Network service)

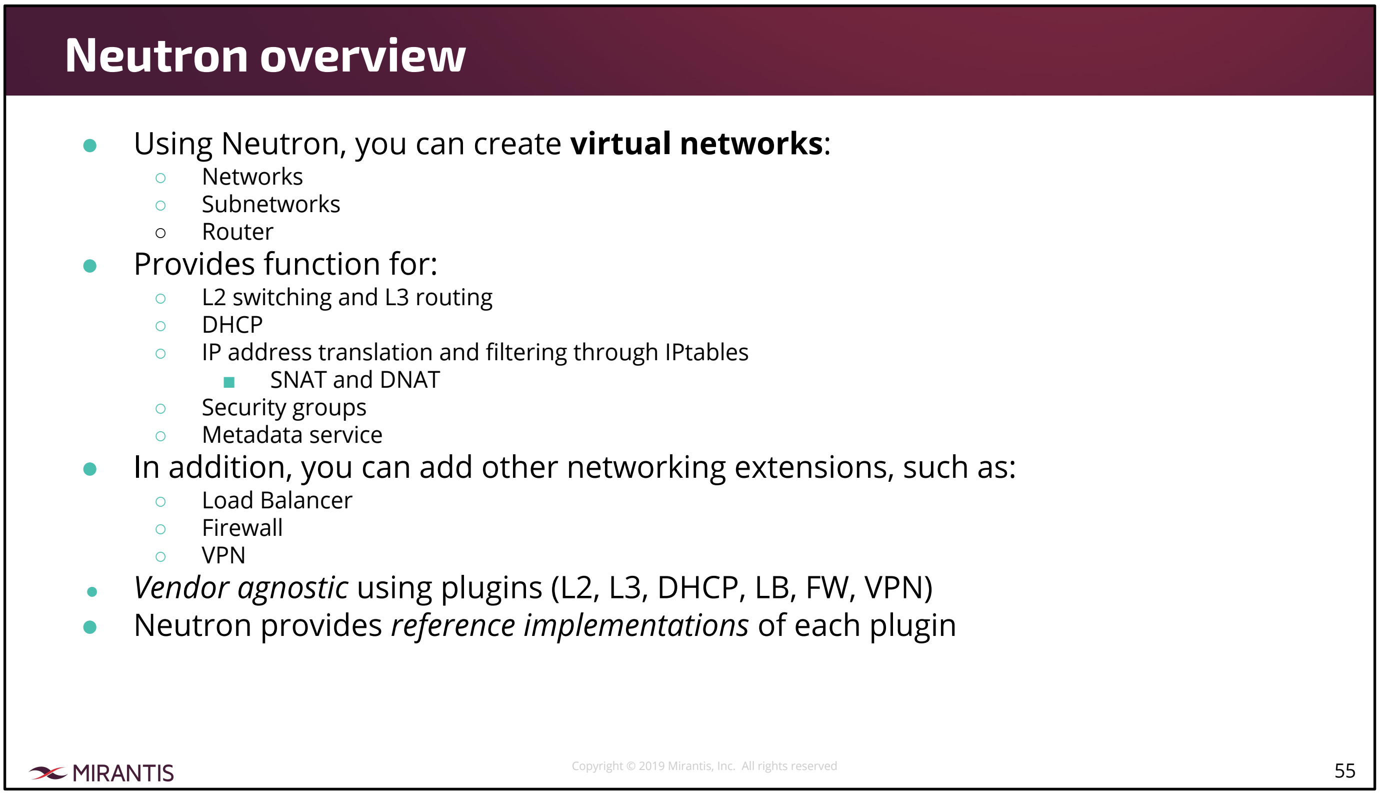
(Nova interacts directly with metadata agent)

MIRANTIS

Neutron is a component (project) of OpenStack that virtualizes network resources, such as networks, subnets, routers, and floating IPs, emulating the physical network. Networks can be created on request from the CLI, REST API, or Horizon Dashboard UI.
Neutron provides support to build advanced network topologies and policies for multi-tier applications.

Vendors provide many Neutron plugins (pluggable python classes), making Neutron *vendor agnostic*. The plugins can be downloaded from the OpenStack Marketplace: https://www.openstack.org/marketplace/driver

This lecture is a very high-level introduction to OpenStack Neutron. Neutron and networking are discussed in more detail in the OS100 course.



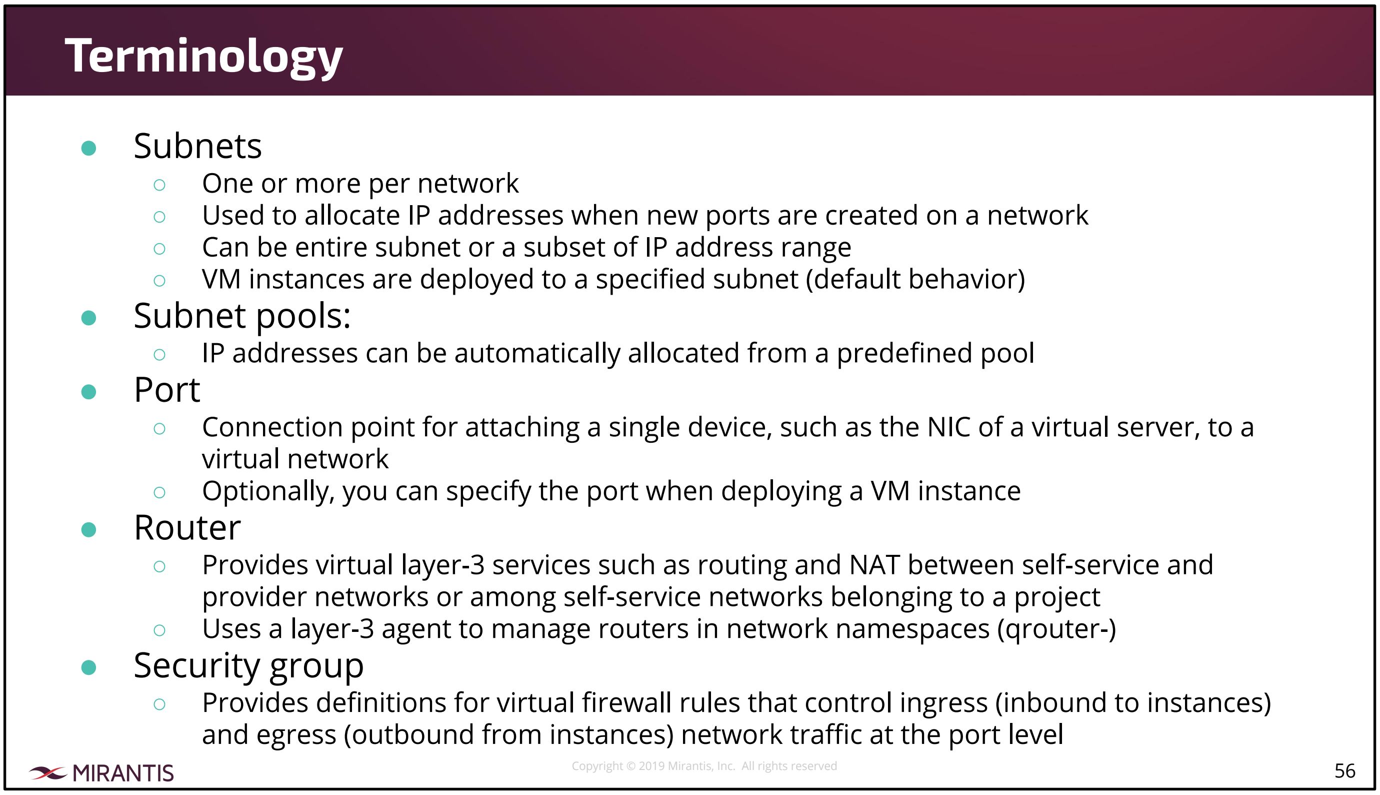
Neutron overview

- Using Neutron, you can create **virtual networks**:
  - Networks
  - Subnetworks
  - Router
- Provides function for:
  - L2 switching and L3 routing
  - DHCP
  - IP address translation and filtering through IPtables
    - SNAT and DNAT
  - Security groups
  - Metadata service
- In addition, you can add other networking extensions, such as:
  - Load Balancer
  - Firewall
  - VPN
- *Vendor agnostic* using plugins (L2, L3, DHCP, LB, FW, VPN)
- Neutron provides *reference implementations* of each plugin

MIRANTIS

Neutron consists of the **neutron-server**, a database for persistent storage, and any number of plugin agents, which provide other services such as interfacing with native Linux networking mechanisms, external devices, or SDN controllers.

Neutron provides control plane functions (layer-2 and layer-3). It allows VMs to obtain IP addresses to gain connectivity. L3/NAT forwarding allows external network access to deployed VMs using floating IP addresses.
The DHCP service uses **dnsmasq** by default.
Neutron security groups behave similar to a virtual firewall at the port level.
The metadata service is implemented by **nova-api**; allowing VM instances to access SSH keys, init scripts, and so on.
The load balancer service is based on HAProxy, by default, but can be configured to use other implementations. HAProxy is an example of a ***reference implementation***.

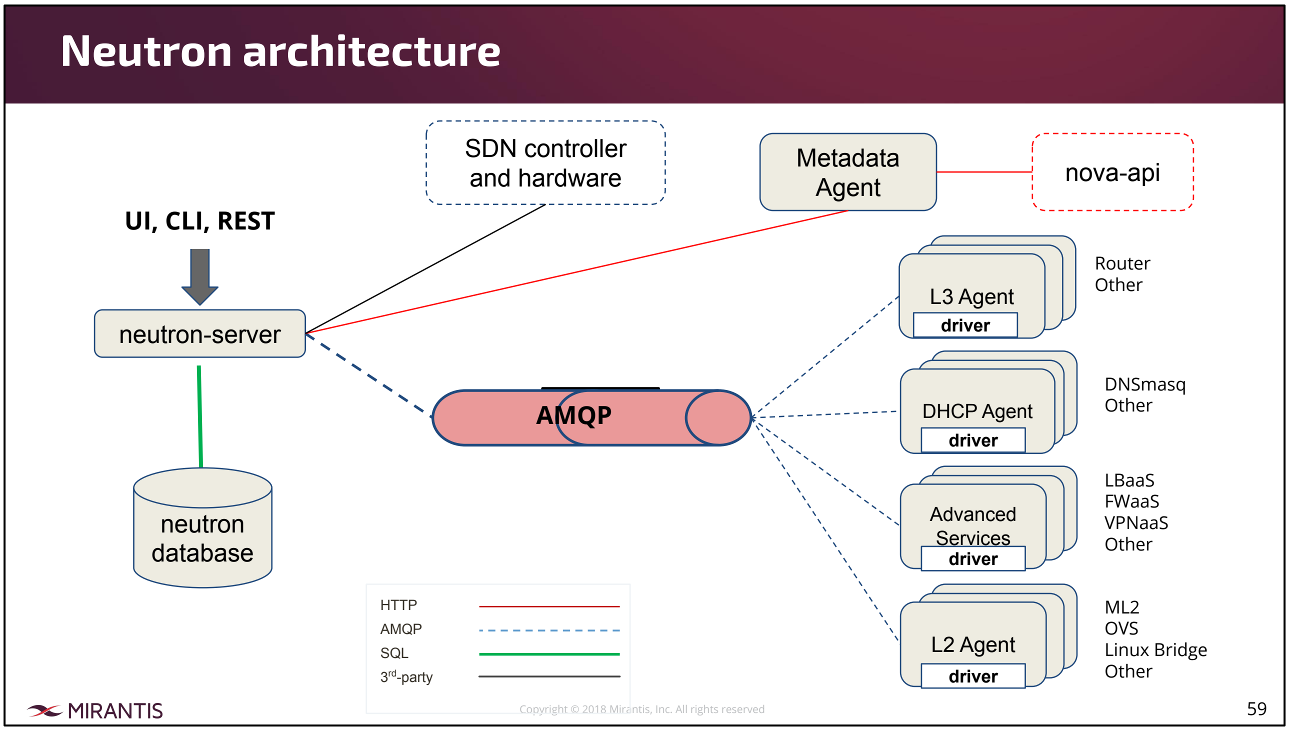


Terminology

- Subnets
  - One or more per network
  - Used to allocate IP addresses when new ports are created on a network
  - Can be entire subnet or a subset of IP address range
  - VM instances are deployed to a specified subnet (default behavior)
- Subnet pools:
  - IP addresses can be automatically allocated from a predefined pool
- Port
  - Connection point for attaching a single device, such as the NIC of a virtual server, to a virtual network
  - Optionally, you can specify the port when deploying a VM instance
- Router
  - Provides virtual layer-3 services such as routing and NAT between self-service and provider networks or among self-service networks belonging to a project
  - Uses a layer-3 agent to manage routers in network namespaces (qrouter-)
- Security group
  - Provides definitions for virtual firewall rules that control ingress (inbound to instances) and egress (outbound from instances) network traffic at the port level

MIRANTIS

This slide contains standard network terminology.
Neutron security groups might be new to you, but their function is not. Neutron provides a default security group that blocks all ingress traffic to all VM instances. For example, if you require SSH access to VM instances, you need to create a security group rule that allows ingress traffic to port 22.



Neutron architecture

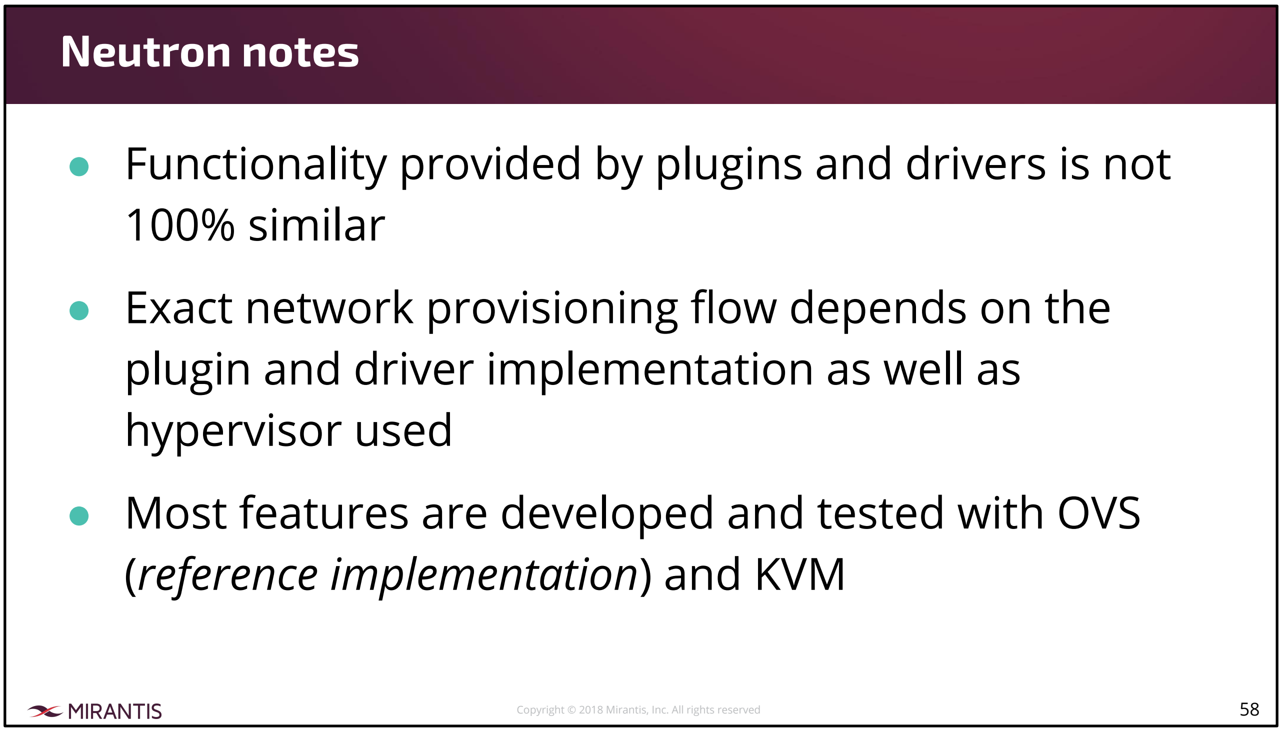
This slide provides an introduction to the Neutron component architecture:

- neutron-server: main API point, handles both core and extension (API and plugins) functions. It also enforces the network model and IP addressing of each port. The neutron-server and plugin agents require access to a database for persistent storage and access to a message queue for inter-communication.
    - Extension plugins are used to implement routers (L3 function), LBaaS, FWaaS, VPNaaS.
    - Quotas and security groups are also implemented as service extensions.
- Neutron Open vSwitch (OVS) agent: (example of an L2 agent) Manages OVS and OVS processes
- Neutron DHCP agent: Manages dnsmasq processes. Provides DHCP services to tenant networks. This agent is the same across all plugins and is responsible for maintaining DHCP configuration. The neutron-dhcp-agent requires message queue access
- Neutron L3 agent: Provides L3/NAT forwarding for external network access of VMs on tenant networks. Requires message queue access. *Optional depending on plugin.*
- Metadata server: (not shown) Interact with metadata from nova on provisioning to assign port and IP address for VM provisioning
- Service plug-ins: Provide or manage extra capabilities such as load balancer, firewall, or VPN support
- **network provider services** (SDN server/services): Provide additional networking services that are provided to tenant networks. These SDN services may interact with the neutron-server, neutron-plugin, and/or plugin-agents via REST APIs or other communication channels.
- ML2 plugin provides network type drivers as well as mechanism drivers. Mechanism drivers are used to implement the types of networks. The supported network types

- are:
  - Local: Single network; can only be realized on a single host. This is only used in proof-of-concept or development environments.
  - Flat: Single network, no segregation – all servers are able to see the same broadcast traffic and can contact each other without requiring a router.
  - VLAN: Network segregation with VLAN and VLAN tagging; communication at L2 level (limitation of 4K VLAN IDs)
  - VXLAN (Virtual Extensible LAN): Same basic function as VLAN, but supports extended address ranges beyond VLAN limit
  - GRE (Generic Routing Encapsulation): Overlay network with tunneling protocol for transporting L3 data over IP

In a multiple node implementation, assuming an Open vSwitch implementation:

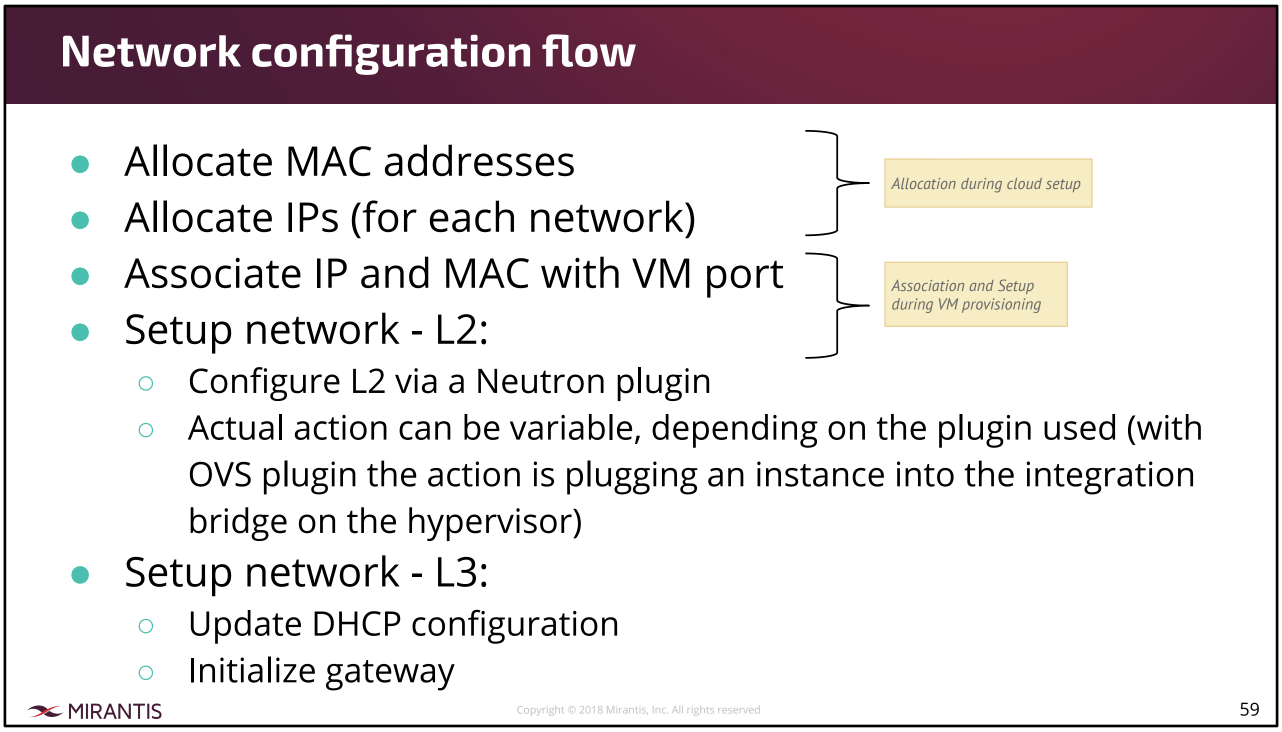
- Neutron-server with the L2 agents run on the controller node
- OVS agent runs on the compute node, along with nova-compute
- Metadata, L3 (routing), DHCP, and OVS agents run on the network node



Neutron notes

- Functionality provided by plugins and drivers is not 100% similar
- Exact network provisioning flow depends on the plugin and driver implementation as well as hypervisor used
- Most features are developed and tested with OVS (*reference implementation*) and KVM

MIRANTIS



Network configuration flow

- Allocate MAC addresses
- Allocate IPs (for each network)
- Associate IP and MAC with VM port
- Setup network - L2:
  - Configure L2 via a Neutron plugin
  - Actual action can be variable, depending on the plugin used (with OVS plugin the action is plugging an instance into the integration bridge on the hypervisor)
- Setup network - L3:
  - Update DHCP configuration
  - Initialize gateway

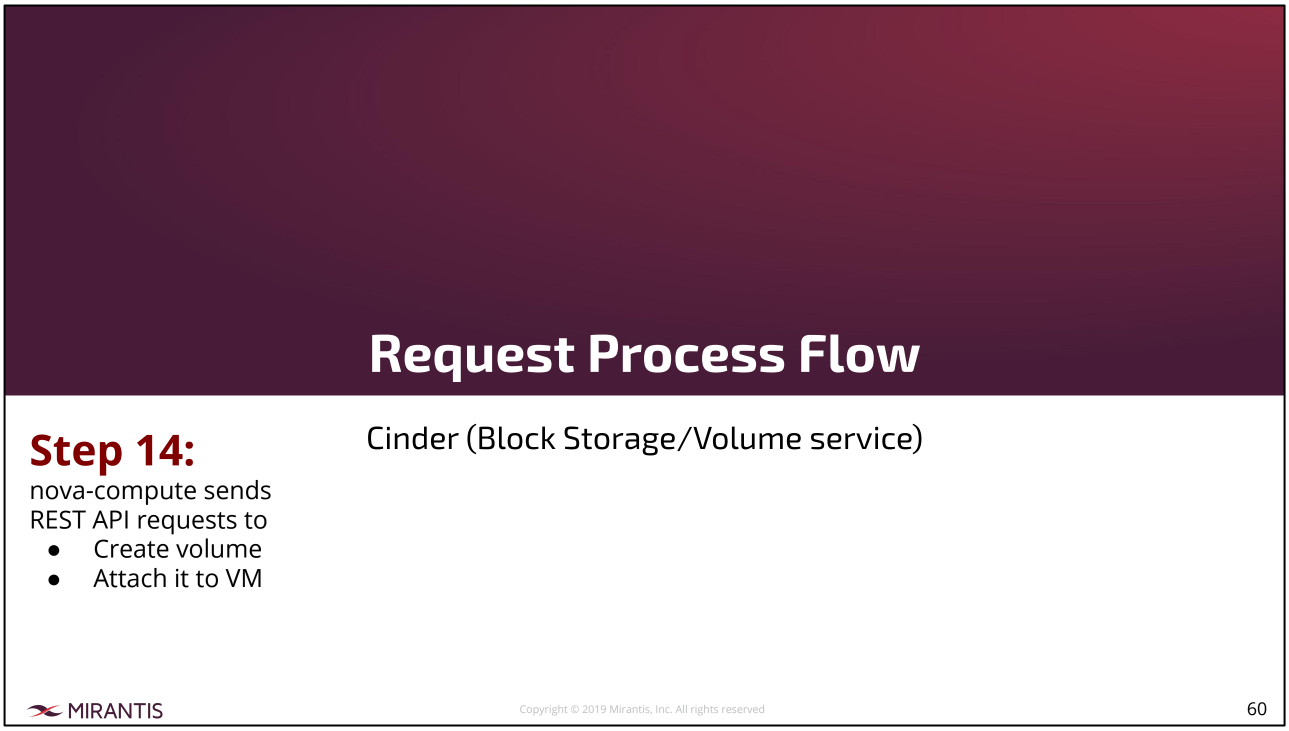
Allocation during cloud setup

Association and Setup during VM provisioning

MIRANTIS

59

This slide provides a very high-level overview of the Neutron process flow using the reference implementation (ML2 plugins, OVS, etc). More details are discussed in the *Neutron Deep Dive* lecture.



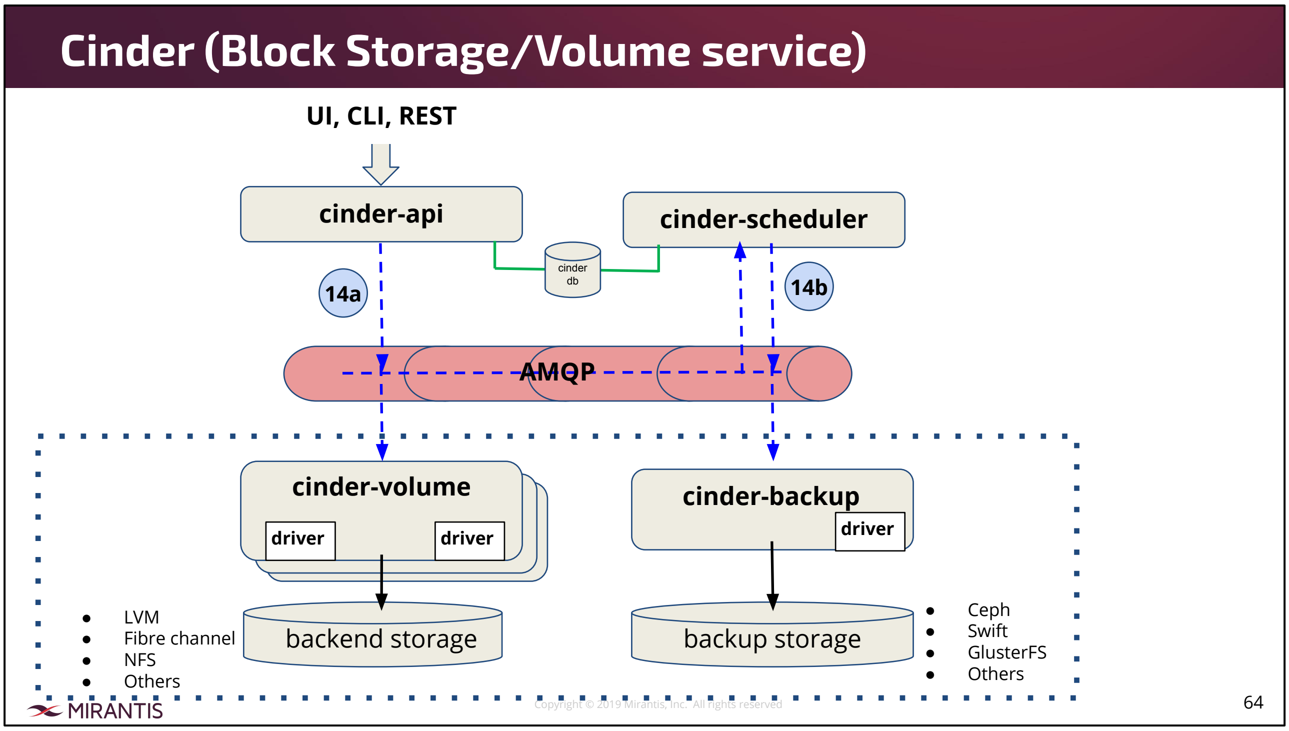
Request Process Flow

Step 14:
nova-compute sends REST API requests to
- Create volume
- Attach it to VM

Cinder (Block Storage/Volume service)

MIRANTIS

Cinder, the OpenStack Block Storage service, virtualizes pools of block storage that can be used with (attached to) virtual machine instances. Users can attach the storage (typically referred to as volumes) without much knowledge of where the storage is deployed, what type of device is used, and so on. Cinder evolved from the nova-volume service.
Cinder volumes are persistent. Once created, they can be attached, detached, then attached to a different virtual machine, for example. Data stored on Cinder volumes is not deleted unless it is deleted or the volume is deleted. Cinder supports multiple back-ends (for many vendors) through the use of plugin drivers.
A Cinder volume might contain a fully bootable instance of an operating system. If so, users can boot VM instances from the volume.



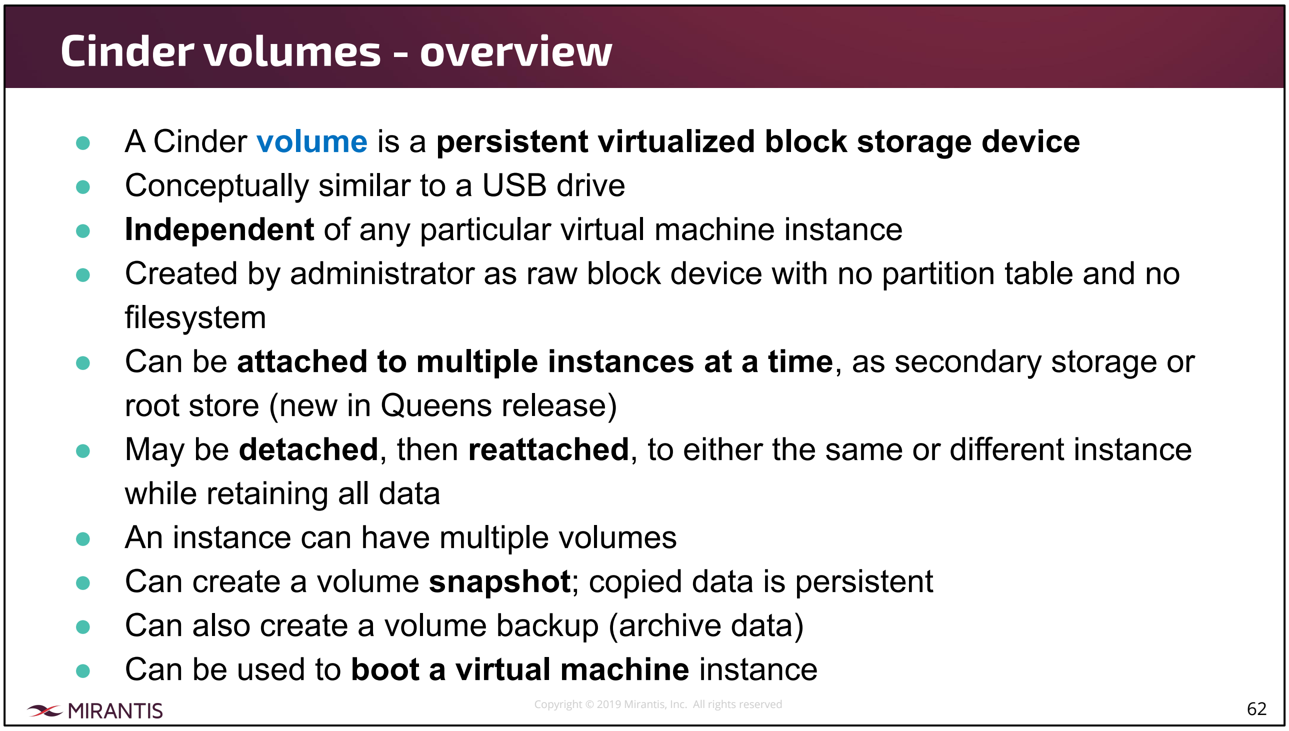
<u>Step 14:</u>

- nova-compute performs a REST call, passing the **auth-token**, to the cinder-api to create and attach volumes to the instance.
- cinder-api validates the **auth-token** with keystone.
- cinder-api routes the request over AMQP to the cinder-scheduler
- cinder-scheduler applies its filters and weights to determine the best volume node for the request
- cinder-scheduler routes the request for the volume to the cinder-volume process on the appropriate volume node.
- nova-compute then retrieves the block storage info. The volume will be attached to the VM instance after it is created.

<u>Cinder overview</u>:

- **cinder-api:** Accepts API requests (from the CLI, REST API, or Horizon Dashboard) and routes to the scheduler or backup services
- **cinder-scheduler:** Schedules and routes requests to the appropriate cinder-volume service. Depending upon your configuration, this may be simple round-robin scheduling to the running volume services, or it can be more sophisticated through the use of the Filter Scheduler.
- **cinder-volume:** Manages a variety of storage providers (backends) and devices through a pluggable driver architecture. Responds to requests to read from and write to the database tp Block Storage database to maintain state. Interacts with other processes (such as, cinder-scheduler) through a message queue and directly on block storage
- **backend storage:** Cinder requires at least 1 form of backend storage. The default

- implementation is Logical Volume Manager (LVM) with iSCSI devices. Other backends are supported through drivers that are downloaded from the OpenStack Marketplace, such as: EMC, NetApp, IBM, and so on.
- **cinder-backup:** Provides a means to backup Block Storage volumes. For example, as objects in the OpenStack Object Storage (Swift) service

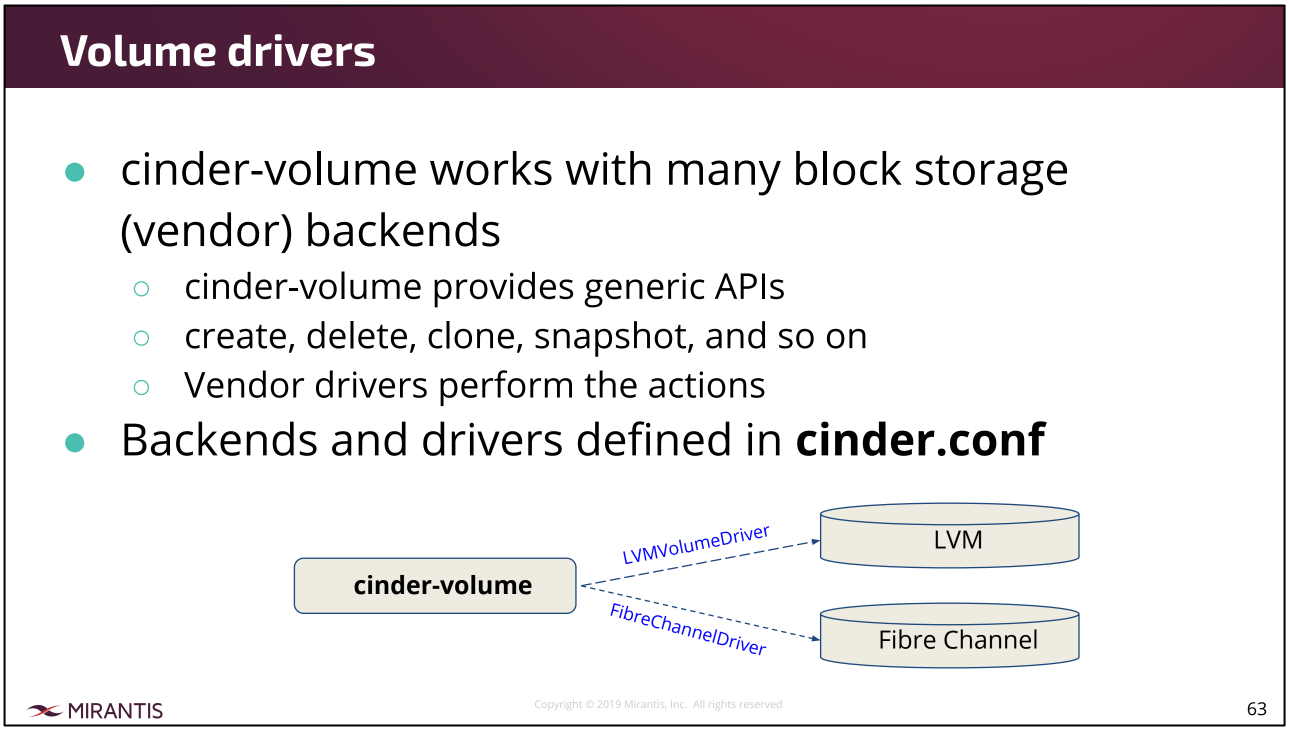


Cinder volumes - overview

- A Cinder volume is a **persistent virtualized block storage device**
- Conceptually similar to a USB drive
- **Independent** of any particular virtual machine instance
- Created by administrator as raw block device with no partition table and no filesystem
- Can be **attached to multiple instances at a time**, as secondary storage or root store (new in Queens release)
- May be **detached**, then **reattached**, to either the same or different instance while retaining all data
- An instance can have multiple volumes
- Can create a volume **snapshot**; copied data is persistent
- Can also create a volume backup (archive data)
- Can be used to **boot a virtual machine** instance

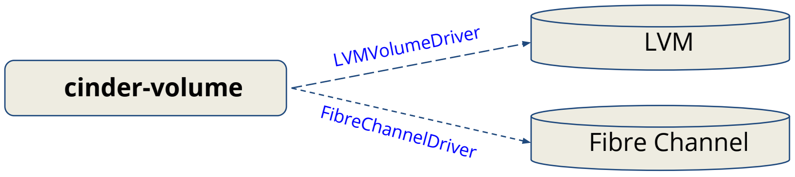
MIRANTIS

For more details related to multi-attach volumes:
https://docs.openstack.org/cinder/latest/admin/blockstorage-volume-multiattach.html

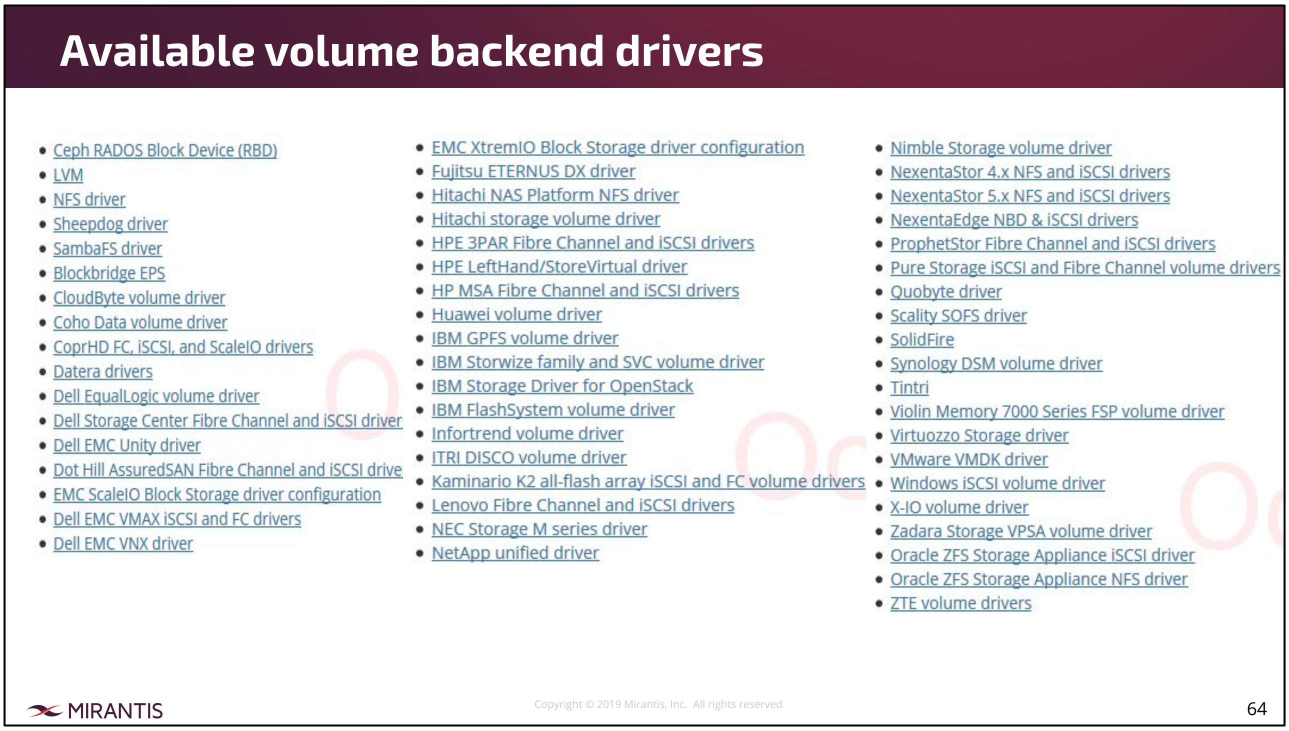


Volume drivers

- cinder-volume works with many block storage (vendor) backends
  - cinder-volume provides generic APIs
  - create, delete, clone, snapshot, and so on
  - Vendor drivers perform the actions
- Backends and drivers defined in **cinder.conf**



Typically, the cinder-volume process is running on a separate node. It supports multiple backends. You might choose to run multiple cinder-volume processes on their own storage nodes.
Several vendors provide backends for Cinder: For example, EMC, IBM, Hitachi, Oracle, VMware, Lenovo, HP, Dell, Citrix, and NetApp. The supported volume drivers are:
https://docs.openstack.org/cinder/latest/drivers.html#volume-drivers



Available volume backend drivers

- Ceph RADOS Block Device (RBD)
- LVM
- NFS driver
- Sheepdog driver
- SambaFS driver
- Blockbridge EPS
- CloudByte volume driver
- Coho Data volume driver
- CoprHD FC, iSCSI, and ScaleIO drivers
- Datera drivers
- Dell EqualLogic volume driver
- Dell Storage Center Fibre Channel and iSCSI driver
- Dell EMC Unity driver
- Dot Hill AssuredSAN Fibre Channel and iSCSI drive
- EMC ScaleIO Block Storage driver configuration
- Dell EMC VMAX iSCSI and FC drivers
- Dell EMC VNX driver
- EMC XtremIO Block Storage driver configuration
- Fujitsu ETERNUS DX driver
- Hitachi NAS Platform NFS driver
- Hitachi storage volume driver
- HPE 3PAR Fibre Channel and iSCSI drivers
- HPE LeftHand/StoreVirtual driver
- HP MSA Fibre Channel and iSCSI drivers
- Huawei volume driver
- IBM GPFS volume driver
- IBM Storwize family and SVC volume driver
- IBM Storage Driver for OpenStack
- IBM FlashSystem volume driver
- Infortrend volume driver
- ITRI DISCO volume driver
- Kaminario K2 all-flash array iSCSI and FC volume drivers
- Lenovo Fibre Channel and iSCSI drivers
- NEC Storage M series driver
- NetApp unified driver
- Nimble Storage volume driver
- NexentaStor 4.x NFS and iSCSI drivers
- NexentaStor 5.x NFS and iSCSI drivers
- NexentaEdge NBD & iSCSI drivers
- ProphetStor Fibre Channel and iSCSI drivers
- Pure Storage iSCSI and Fibre Channel volume drivers
- Quobyte driver
- Scality SOFS driver
- SolidFire
- Synology DSM volume driver
- Tintri
- Violin Memory 7000 Series FSP volume driver
- Virtuozzo Storage driver
- VMware VMDK driver
- Windows iSCSI volume driver
- X-IO volume driver
- Zadara Storage VPSA volume driver
- Oracle ZFS Storage Appliance iSCSI driver
- Oracle ZFS Storage Appliance NFS driver
- ZTE volume drivers

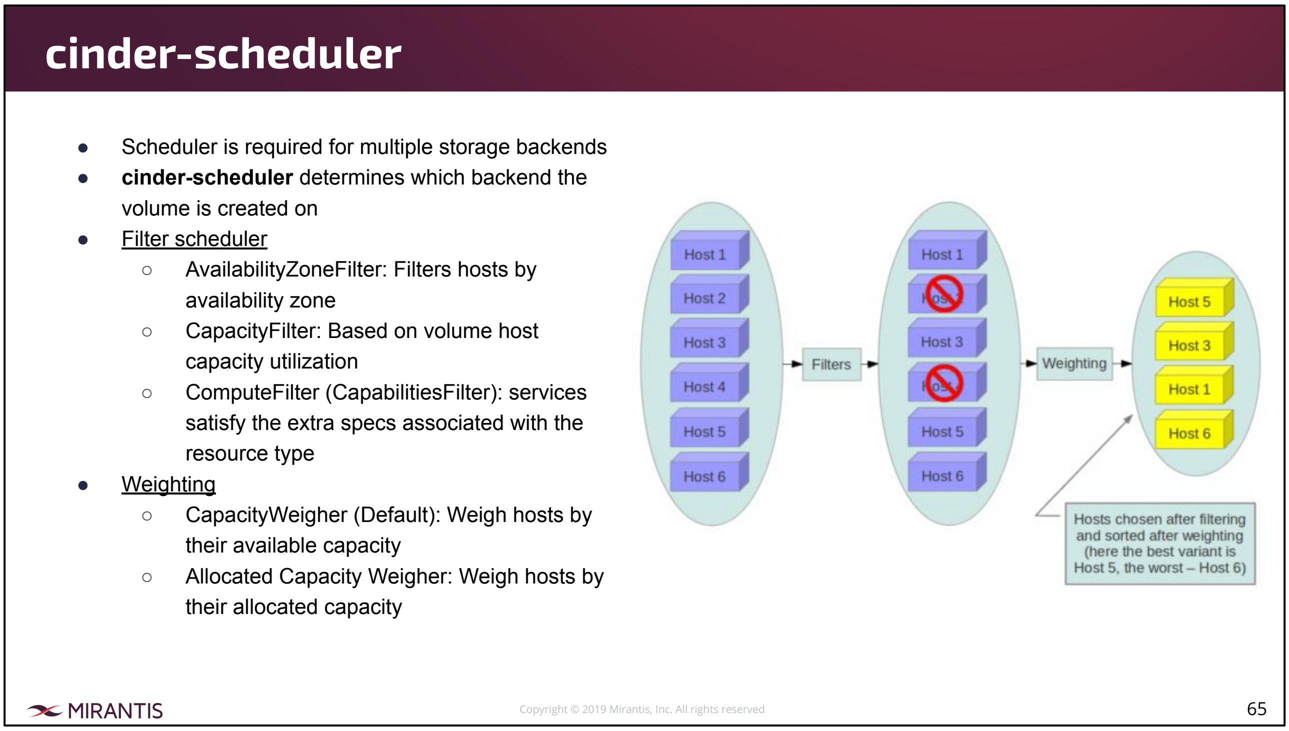
MIRANTIS

List of available backend drivers:
https://docs.openstack.org/ocata/config-reference/block-storage/volume-drivers.html

To define a volume driver, edit **cinder.conf**, set the **volume_driver** property. The default uses LVM:
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver

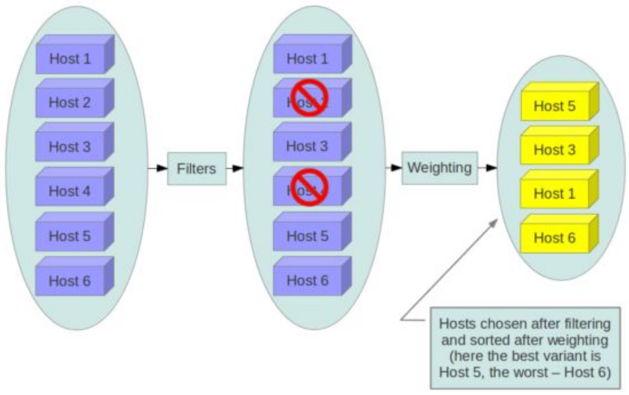
Notes:

- Functionality provided by drivers is not 100% similar.
- Exact volume provisioning and attachment flow depends on driver implementation and Hypervisor used.
- Most features are developed and tested with LVM over ISCSI and KVM.



cinder-scheduler

- Scheduler is required for multiple storage backends
- **cinder-scheduler** determines which backend the volume is created on
- Filter scheduler
  - AvailabilityZoneFilter: Filters hosts by availability zone
  - CapacityFilter: Based on volume host capacity utilization
  - ComputeFilter (CapabilitiesFilter): services satisfy the extra specs associated with the resource type
- Weighting
  - CapacityWeigher (Default): Weigh hosts by their available capacity
  - Allocated Capacity Weigher: Weigh hosts by their allocated capacity

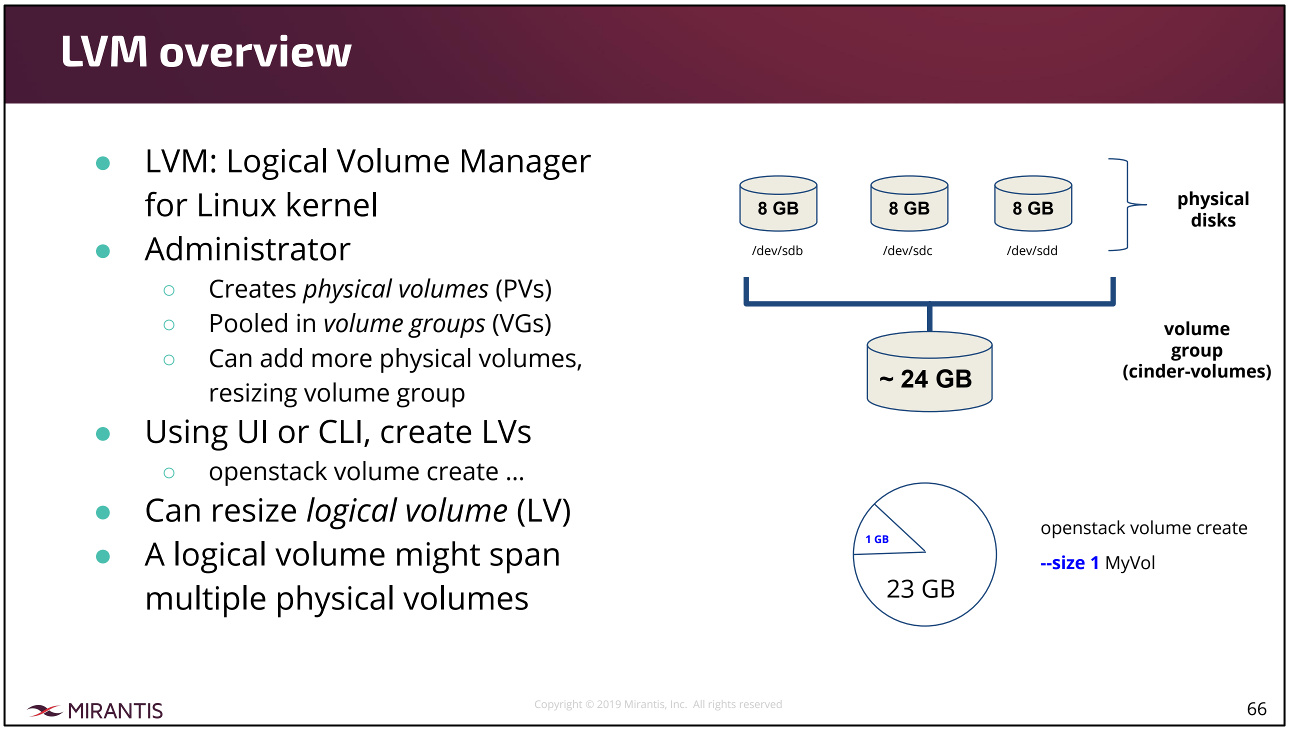

Host 1
Host 2
Host 3
Host 4
Host 5
Host 6
Filters
Host 1
Host 3
Host 5
Host 6
Weighting
Host 5
Host 3
Host 1
Host 6
Hosts chosen after filtering and sorted after weighting (here the best variant is Host 5, the worst – Host 6)

Available filters:

- **AvailabilityZoneFilter:** Filters Backends by availability zone.
- **CapabilitiesFilter:** BackendFilter to work with resource (instance & volume) type records.
- **CapacityFilter:** Capacity filters based on volume backend's capacity utilization.
- **DifferentBackendFilter:** Schedule volume on a different back-end from a set of volumes.
- **DriverFilter:** DriverFilter filters backend based on a 'filter function' and metrics.
- **InstanceLocalityFilter:** Schedule volume on the same host (hypervisor) as a given instance.
- **JsonFilter:** Backend filter for simple JSON-based grammar for selecting backends.
- **RetryFilter:** Filter out previously attempted hosts
- **SameBackendFilter:** Schedule volume on the same back-end as another volume.

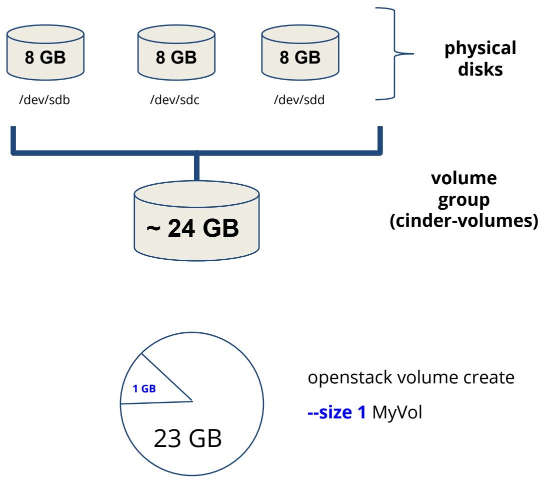
Available weights:

- **AllocatedCapacityWeigher:** Allocated Capacity Weigher weighs hosts by their allocated capacity.
- **CapacityWeigher:** Capacity Weigher weighs hosts by their virtual or actual free capacity. (DEFAULT)
- **ChanceWeigher:** Chance Weigher assigns random weights to hosts.
- **GoodnessWeigher:** Assign weights based on a host's *goodness function*.
- **VolumeNumberWeigher:** Weigher that weighs hosts by volume number in backends.



LVM overview

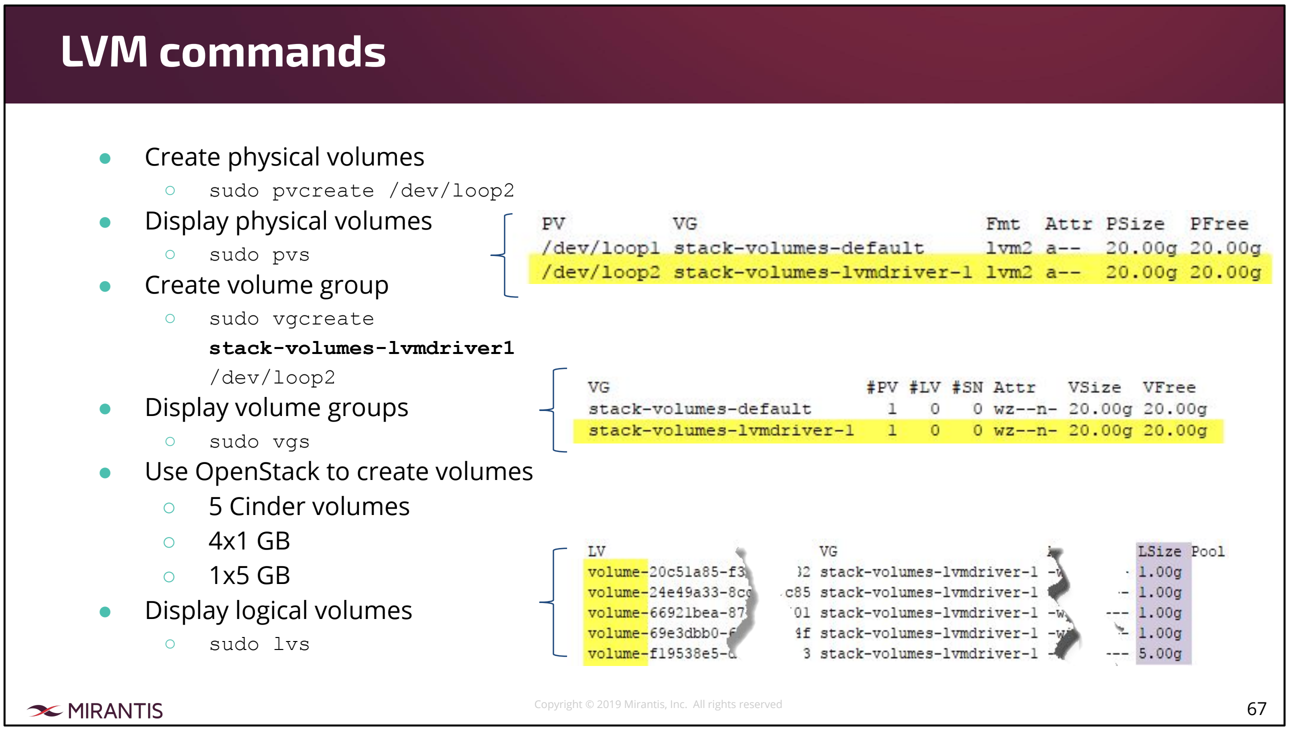
- LVM: Logical Volume Manager for Linux kernel
- Administrator
  - Creates *physical volumes* (PVs)
  - Pooled in *volume groups* (VGs)
  - Can add more physical volumes, resizing volume group
- Using UI or CLI, create LVs
  - openstack volume create ...
- Can resize *logical volume* (LV)
- A logical volume might span multiple physical volumes



MIRANTIS

Logical Volume Manager (LVM) creates an abstraction layer over the physical storage (disks). Users do not need to know the details of the physical layer. For example, in this case, a pool of 24GB of storage is available (from the volume group) for use.

Suppose you create (openstack volume create command) a 1 GB Cinder volume. The cinder-volume process uses the LVMVolumeDriver (defined in cinder.conf) to create and manage the 1 GB volume.



LVM commands

- Create physical volumes
  - `sudo pvcreate /dev/loop2`
- Display physical volumes
  - `sudo pvs`
- Create volume group
  - `sudo vgcreate` **`stack-volumes-lvmdriver1`** `/dev/loop2`
- Display volume groups
  - `sudo vgs`
- Use OpenStack to create volumes
  - 5 Cinder volumes
  - 4x1 GB
  - 1x5 GB
- Display logical volumes
  - `sudo lvs`

```
PV          VG                         Fmt  Attr PSize  PFree
/dev/loop1  stack-volumes-default      lvm2 a--  20.00g 20.00g
/dev/loop2  stack-volumes-lvmdriver-1  lvm2 a--  20.00g 20.00g
```

```
VG                         #PV #LV #SN Attr   VSize  VFree
stack-volumes-default        1   0   0 wz--n- 20.00g 20.00g
stack-volumes-lvmdriver-1    1   0   0 wz--n- 20.00g 20.00g
```

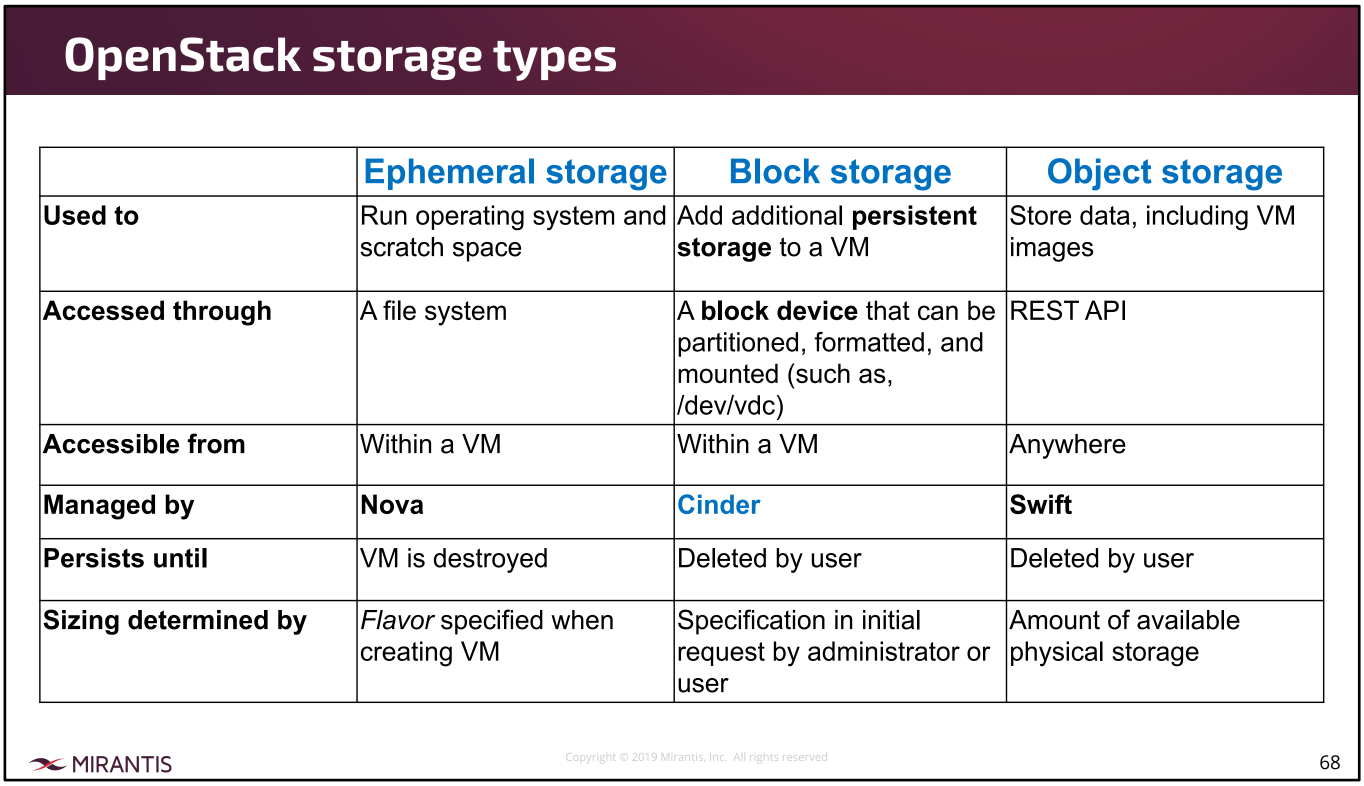
```
LV                       VG                                  LSize Pool
volume-20c51a85-f3     32 stack-volumes-lvmdriver-1 -w         1.00g
volume-24e49a33-8cc   c85 stack-volumes-lvmdriver-1         -  1.00g
volume-66921bea-87     01 stack-volumes-lvmdriver-1 -w     --- 1.00g
volume-69e3dbb0-6      4f stack-volumes-lvmdriver-1 -w      -  1.00g
volume-f19538e5-d       3 stack-volumes-lvmdriver-1 -      --- 5.00g
```

MIRANTIS

pvcreate initializes *physical volumes* for later use by the Logical Volume Manager (LVM). Use the **pvs** command to display the physical volumes. In this case, there are 2 physical volumes allocated, each with 20GB of space.

vgcreate allocates the physical volumes to a *volume group*, stack-volumes-lvmdriver-1. The name**, stack-volumes-lvmdriver-1**, must match the volume-group property specified in the cinder.conf file. Use the **vgs** command to display the volume groups. Notice the total volume group size (VSize) is 20GB with 20GB free (VFree).

Use the **lvs** command to display all *logical volumes*. In this example, there are 5 Cinder (logical) volumes created for a total of 9GB. Using the **lvs** command does not provide you with the project associated with the logical volumes. You should manage logical volumes from OpenStack (CLI or UI).



OpenStack storage types

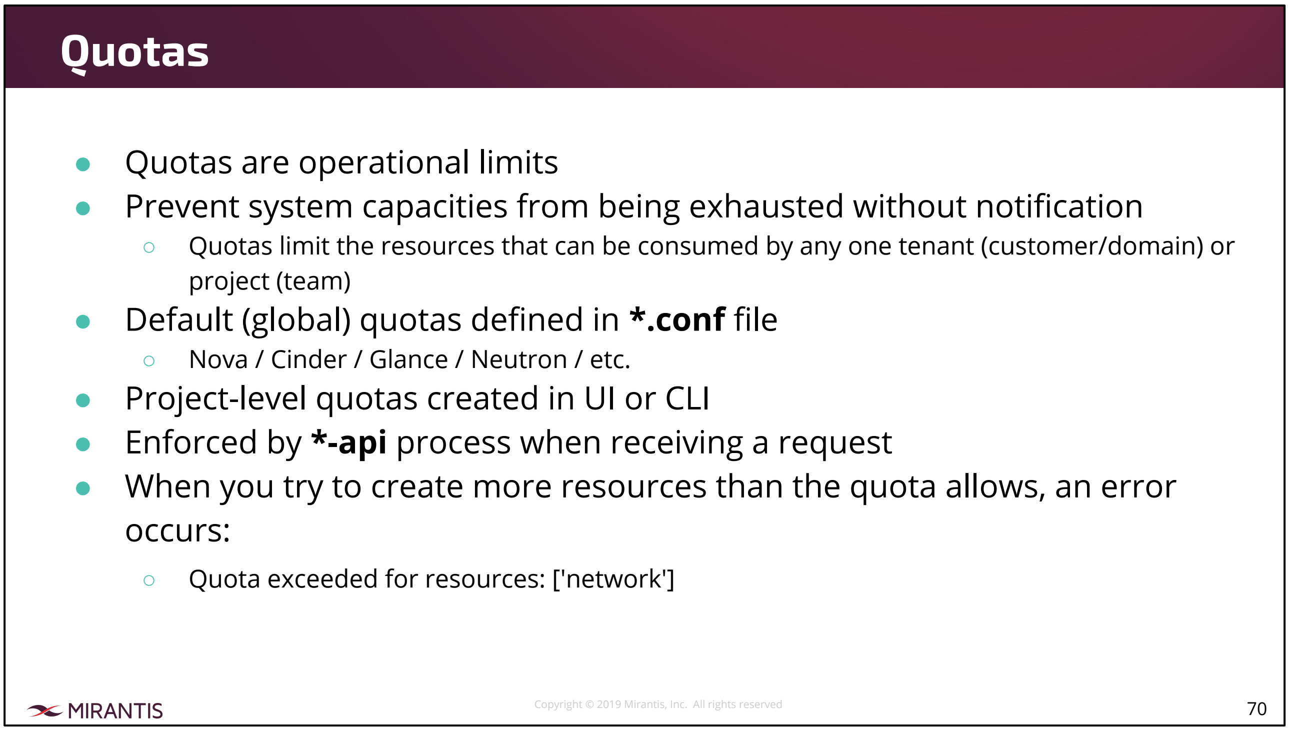
	Ephemeral storage	Block storage	Object storage
Used to	Run operating system and scratch space	Add additional **persistent storage** to a VM	Store data, including VM images
Accessed through	A file system	A **block device** that can be partitioned, formatted, and mounted (such as, /dev/vdc)	REST API
Accessible from	Within a VM	Within a VM	Anywhere
Managed by	**Nova**	**Cinder**	**Swift**
Persists until	VM is destroyed	Deleted by user	Deleted by user
Sizing determined by	*Flavor* specified when creating VM	Specification in initial request by administrator or user	Amount of available physical storage

MIRANTIS

This slide compares the types of storage provided by the core OpenStack components. The focus of this lesson is the Block Storage service, known as Cinder.

Resource quotas

This lesson discusses quotas for OpenStack resources



Quotas

- Quotas are operational limits
- Prevent system capacities from being exhausted without notification
  - Quotas limit the resources that can be consumed by any one tenant (customer/domain) or project (team)
- Default (global) quotas defined in ***.conf** file
  - Nova / Cinder / Glance / Neutron / etc.
- Project-level quotas created in UI or CLI
- Enforced by ***-api** process when receiving a request
- When you try to create more resources than the quota allows, an error occurs:
  - Quota exceeded for resources: ['network']

MIRANTIS

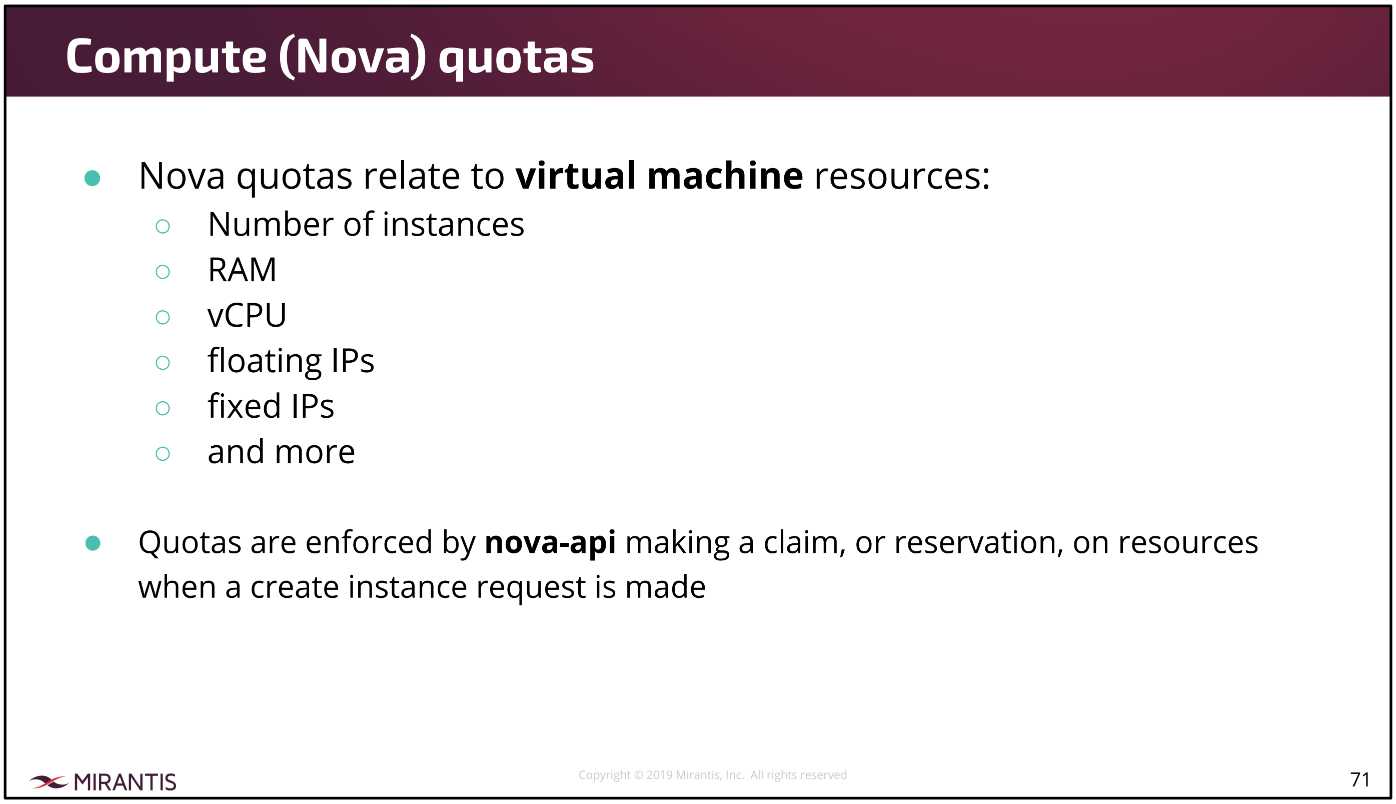
Quotas prevent any one project from consuming all of your cloud resources - IP addresses, amount of vCPU, number of instances, networks, vm resources, volumes, images, etc.

Quotas are defined at the component level, such as Nova quotas for virtual machine resources (vCPU, RAM, etc).

From the CLI, use the **openstack quota {set | list}** command to set project-level quotas.

Managing quotas requires the admin role.

For more information:
https://docs.openstack.org/horizon/latest/admin/set-quotas.html



Compute (Nova) quotas

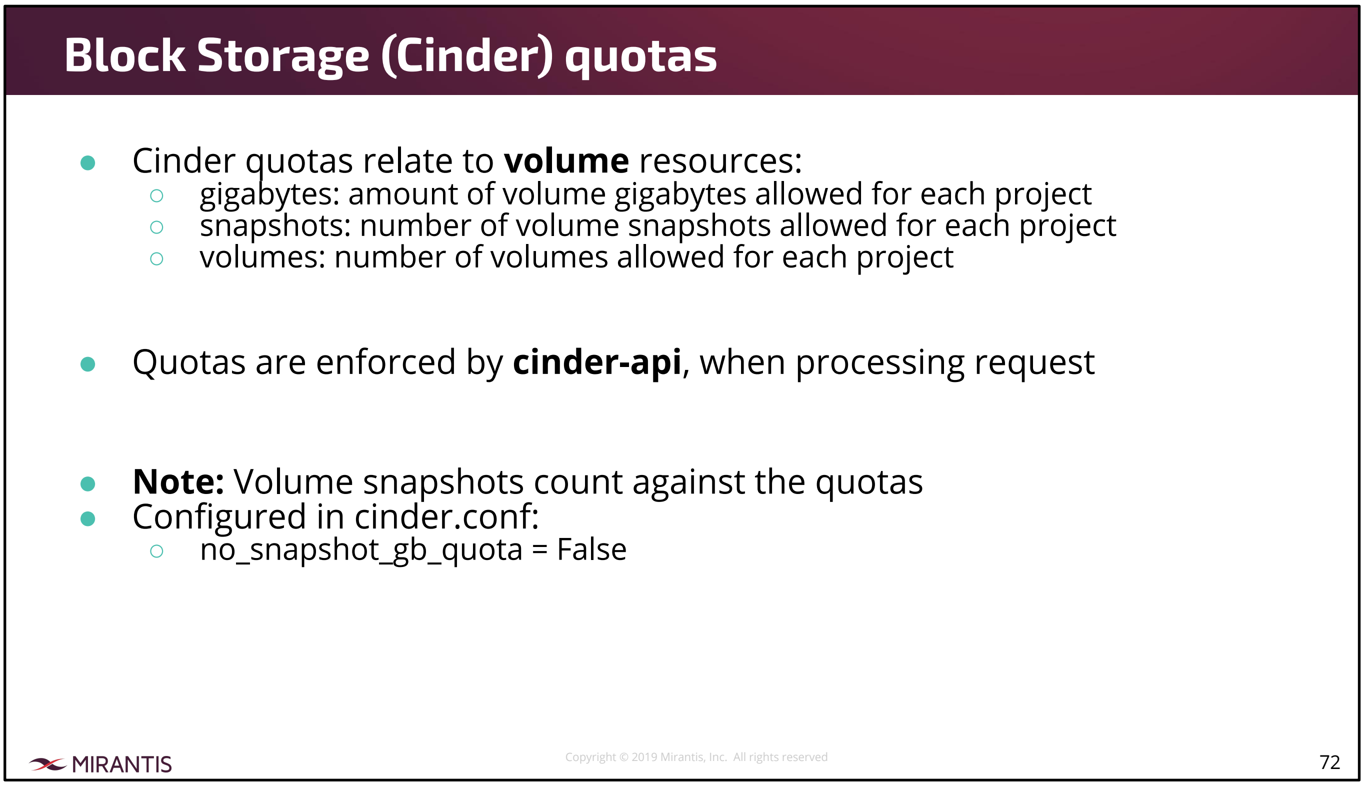
- Nova quotas relate to **virtual machine** resources:
  - Number of instances
  - RAM
  - vCPU
  - floating IPs
  - fixed IPs
  - and more
- Quotas are enforced by **nova-api** making a claim, or reservation, on resources when a create instance request is made

MIRANTIS

71

Default (domain-level) quotas are commented out in the **nova.conf** file, including the default driver to use for quota checks.

Notice **there is no quota for disk usage**. If needed, you can use Linux to set limits. Currently, there is a development item to implement a **local_gb (disk)** quota. It might be available in a future release.



Block Storage (Cinder) quotas

- Cinder quotas relate to **volume** resources:
  - gigabytes: amount of volume gigabytes allowed for each project
  - snapshots: number of volume snapshots allowed for each project
  - volumes: number of volumes allowed for each project

- Quotas are enforced by **cinder-api**, when processing request

- **Note:** Volume snapshots count against the quotas
- Configured in cinder.conf:
  - no_snapshot_gb_quota = False

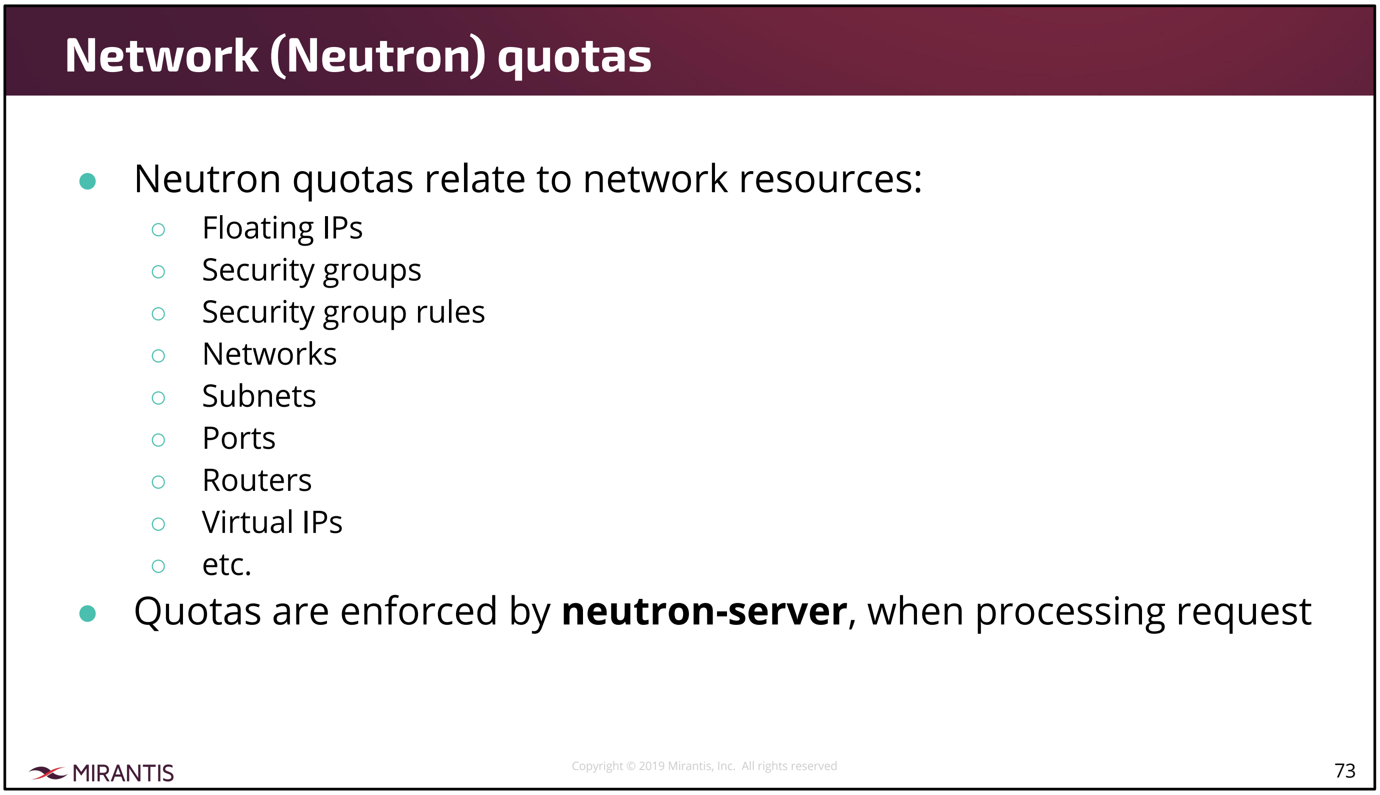
MIRANTIS

72

Quotas are discussed in the OpenStack Administrator Guide:
http://docs.openstack.org/admin-guide/cli-set-quotas.html

Default (domain-level) quotas are commented out in the **cinder.conf** file.

In general, you should use the **openstack quota** commands. In some cases, you might need to use the **cinder quota-*** commands.



Network (Neutron) quotas

- Neutron quotas relate to network resources:
  - Floating IPs
  - Security groups
  - Security group rules
  - Networks
  - Subnets
  - Ports
  - Routers
  - Virtual IPs
  - etc.
- Quotas are enforced by **neutron-server**, when processing request

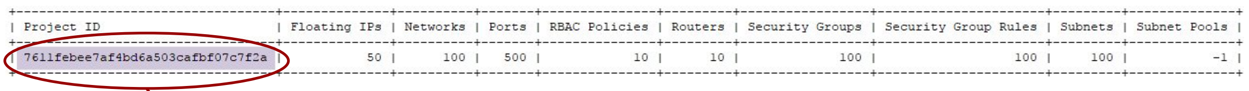
MIRANTIS

Default (domain-level) quotas are commented out in the **neutron.conf** file.

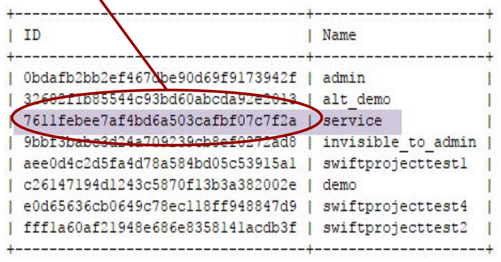
Neutron supports quotas for the network resources: networks, routers, subnets, ports, floating IP addresses, and more.
For example, the number of floating IP addresses allowed for each project can be controlled so that cloud resources are optimized. Quotas can be enforced for each project and for each domain.

CLI quota example

openstack quota list --network



```
+----------------------------------+--------------+----------+-------+---------------+---------+-----------------+----------------------+---------+--------------+
| Project ID                       | Floating IPs | Networks | Ports | RBAC Policies | Routers | Security Groups | Security Group Rules | Subnets | Subnet Pools |
+----------------------------------+--------------+----------+-------+---------------+---------+-----------------+----------------------+---------+--------------+
| 7611febee7af4bd6a503cafbf07c7f2a |           50 |      100 |   500 |            10 |      10 |             100 |                  100 |     100 |           -1 |
+----------------------------------+--------------+----------+-------+---------------+---------+-----------------+----------------------+---------+--------------+
```

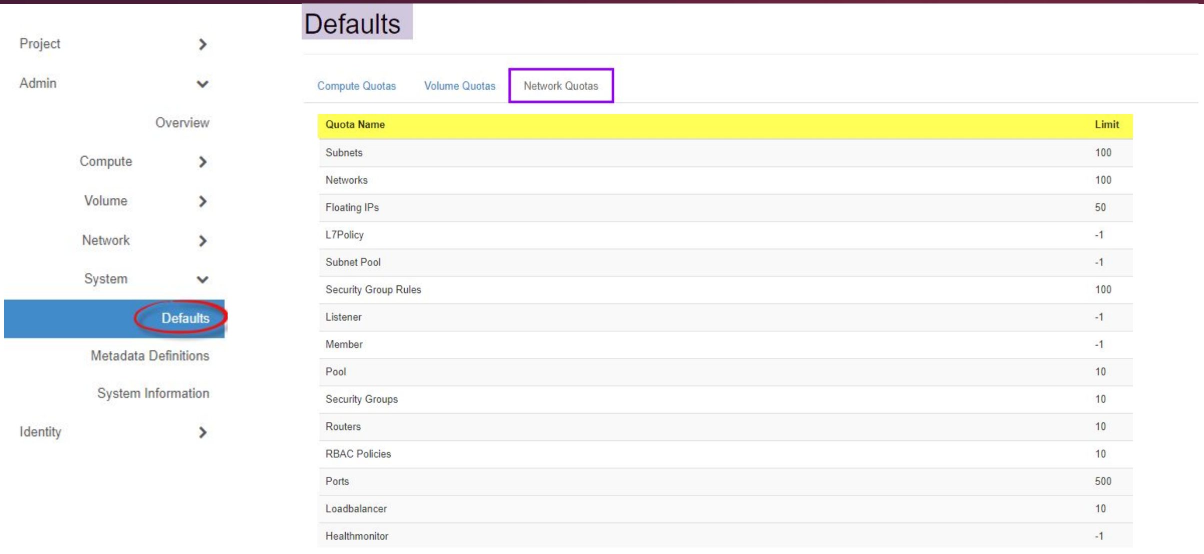


```
+----------------------------------+--------------------+
| ID                               | Name               |
+----------------------------------+--------------------+
| 0bdafb2bb2ef467dbe90d69f9173942f | admin              |
| 32682f1b85544c93bd60abcda92e2013 | alt_demo           |
| 7611febee7af4bd6a503cafbf07c7f2a | service            |
| 9bbf3babc3d24a709239cb8ef0272ad8 | invisible_to_admin |
| aee0d4c2d5fa4d78a584bd05c53915a1 | swiftprojecttest1  |
| c26147194d1243c5870f13b3a382002e | demo               |
| e0d65636cb0649c78ec118ff948847d9 | swiftprojecttest4  |
| fff1a60af21948e686e8358141acdb3f | swiftprojecttest2  |
+----------------------------------+--------------------+
```

openstack project list

- Default (domain-level) neutron quotas defined against the *service* project

UI quota example



Quota Name	Limit
Subnets	100
Networks	100
Floating IPs	50
L7Policy	-1
Subnet Pool	-1
Security Group Rules	100
Listener	-1
Member	-1
Pool	10
Security Groups	10
Routers	10
RBAC Policies	10
Ports	500
Loadbalancer	10
Healthmonitor	-1

Summary

Summary

You should now be able to:

- Explain the flow to deploy an instance and the role of each component
  - Horizon (Dashboard UI)
  - Keystone (Identity service)
  - Nova (Compute service)
  - Neutron (Network service)
  - Glance (Image service)
  - Cinder (Block Storage service)

MIRANTIS

Lab exercises

Lab 4: Block Storage service (Cinder)

Lab 5: Identity service (Keystone)