

# Inside OpenStack

You have a cloud: now what?

# The Plan for today...

Part I: Franz (~80 min or so)

- First steps after the cloud is up and running
- Creation of: images, flavors, networks, VMs, security groups, volumes...
- Hands-on session: play with our dev-cloud

Part II: Leonard (~30 min or so)

- Internal organisation of groups, roles, projects
- Synchronisation with SUPR
- Accounting

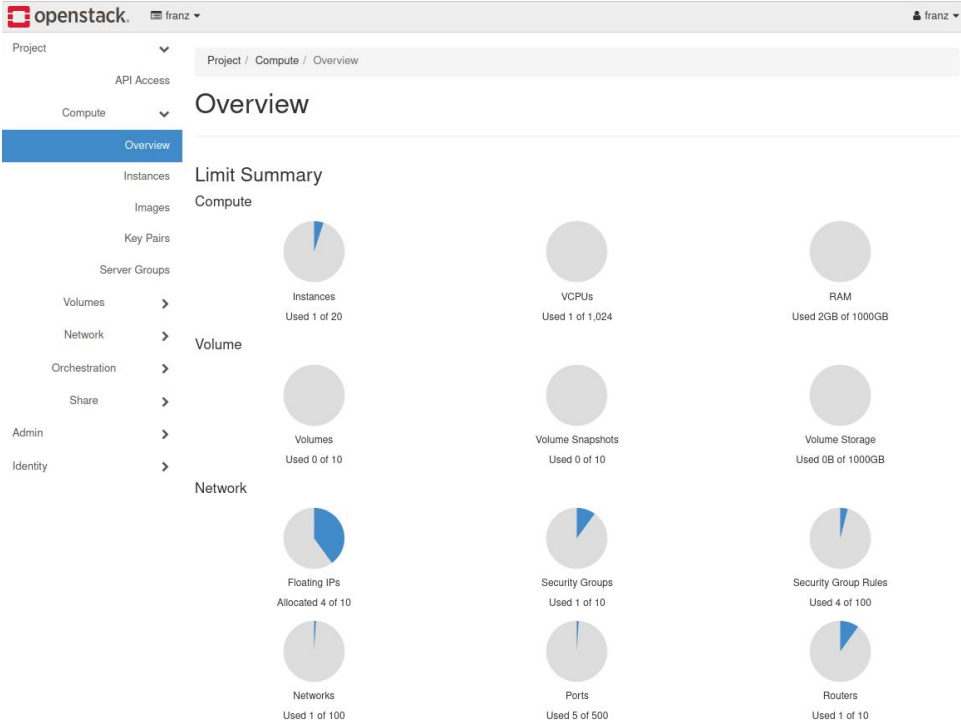
# The cloud is up – first steps

- Kolla-ansible provides script `init-runonce`
  - Great starting point to learn how to use the openstack CLI
  - Sets up the first project + image + network + flavor + quota
- Of course need to install CLI first
  - `pip install python-openstackclient`
- While you're at it, might as well install manila- and octavia-CLI as well:
  - Manila (shared storage): `pip install python-manilaclient`
  - Octavia (loadbalancer): `pip install python-octaviaclient`

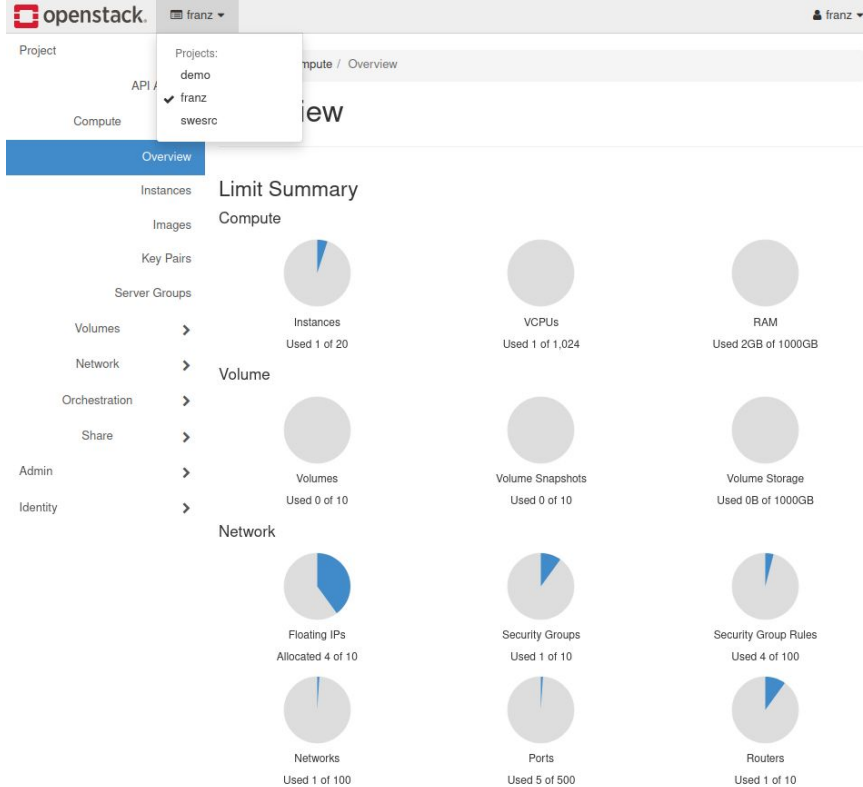
# The cloud is up – first steps

- Let's take a look at `init-runonce`
  - <https://github.com/openstack/kolla-ansible/blob/master/tools/init-runonce>
- What it does:
  - Some sanity checks (CLI installed, access credentials available...)
  - Defines some network masks for both the private and the public network
  - Creates:
    - an image (cirros)
    - a network (router, network, subnet, gateways)
    - security rules (i.e. firewall settings)
    - ssh-key pair
    - a few flavors
    - quotas for the 'admin' project
  - First helping hints on how to create a VM

# The cloud is up – check out Horizon



# The cloud is up – check out Horizon



# The cloud is up – check out Horizon

The screenshot displays the OpenStack Horizon interface. The top navigation bar includes the OpenStack logo, the project name 'franz', and a user profile 'franz'. The left sidebar contains a navigation menu with categories: Project, API Access, Compute (selected), Instances, Images, Key Pairs, Server Groups, Volumes, Network, Orchestration, Share, Admin, and Identity. The main content area is titled 'Overview' and features a 'Limit Summary' section for 'Compute'. This section contains a grid of 12 circular gauges representing resource usage. A dropdown menu is open over the 'Overview' title, listing options: Settings, Help, OpenStack RC File, OpenStack clouds.yaml File, Themes (Default selected, Material), and Sign Out.

Category	Resource	Used	Total
Compute	Instances	Used 1 of 20	20
	VCPUs	Used 1 of 1,024	1,024
	RAM	Used 2GB of 1000GB	1000GB
Volume	Volumes	Used 0 of 10	10
	Volume Snapshots	Used 0 of 10	10
	Volume Storage	Used 0B of 1000GB	1000GB
Network	Floating IPs	Allocated 4 of 10	10
	Security Groups	Used 1 of 10	10
	Security Group Rules	Used 4 of 100	100

# The cloud is up – check out Horizon

Access credentials  
for use with CLI  
client or, e.g.,  
clusterAPI or  
Terraform

The screenshot shows the OpenStack Horizon dashboard interface. The top navigation bar includes the OpenStack logo, the user name 'franz', and a dropdown menu. The left sidebar contains a navigation menu with categories like Project, API Access, Compute, Instances, Images, Key Pairs, Server Groups, Volumes, Network, and Administration. The main content area displays the 'Overview' page for the 'Compute' project, featuring a 'Limit Summary' section with six circular progress indicators for various resources: Instances (Used 1 of 20), VCPUs (Used 1 of 1,024), RAM (Used 2GB of 1000GB), Volumes (Used 0 of 10), Volume Snapshots (Used 0 of 10), and Volume Storage (Used 0B of 1000GB). A user menu dropdown is open, showing options for Settings, Help, OpenStack RC File, OpenStack clouds.yaml File, Themes (Default, Material), and Sign Out. The 'OpenStack RC File' and 'OpenStack clouds.yaml File' options are circled in red.

Resource	Used	Total
Instances	1	20
VCPUs	1	1,024
RAM	2GB	1000GB
Volumes	0	10
Volume Snapshots	0	10
Volume Storage	0B	1000GB

# The cloud is up – check out Horizon: **Images**

Project / Compute / Images

## Images

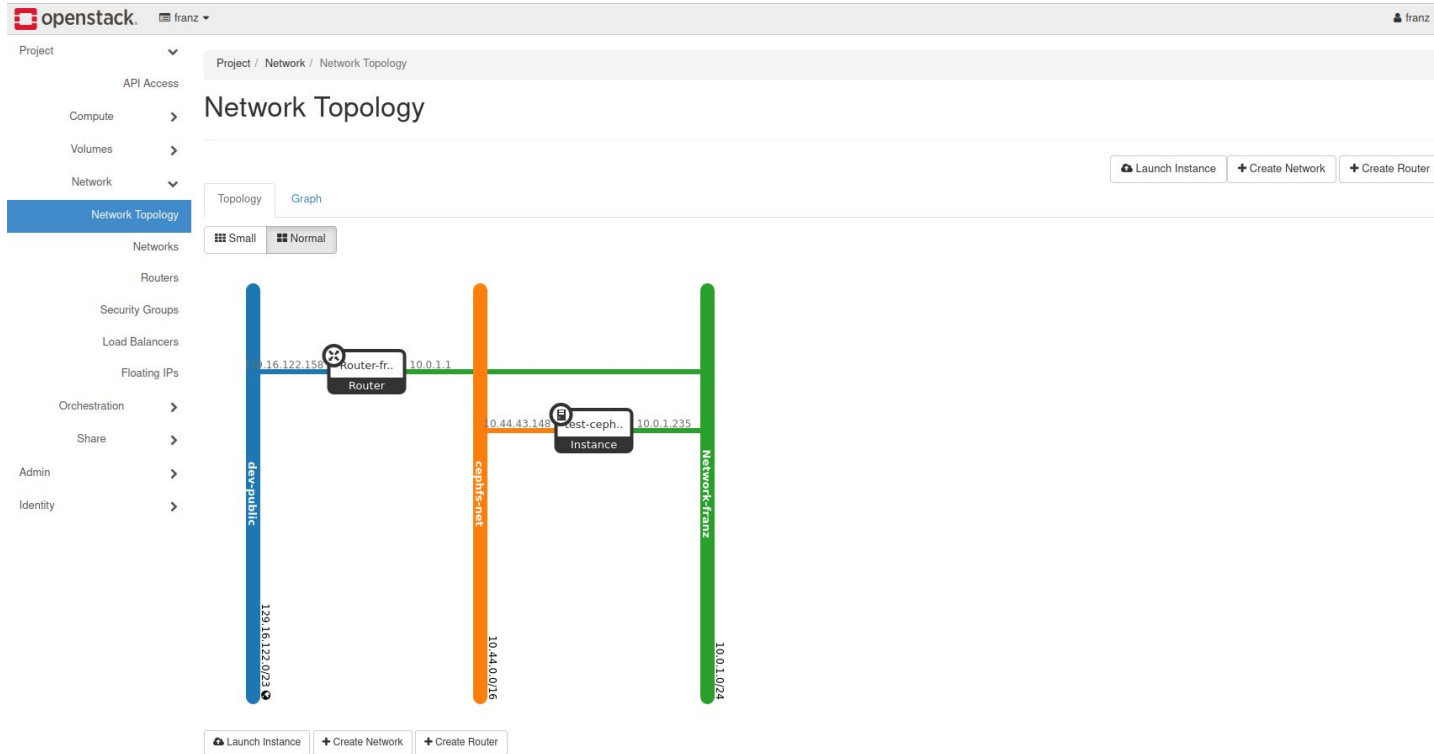
Click here for filters or full text search. + Create Image Delete Images

Displaying 12 items

<input type="checkbox"/>	Owner	Name ^	Type	Status	Visibility	Protected	Disk Format	Size	
<input type="checkbox"/>	admin	centos9-x86_64	Image	Active	Public	No	QCOW2	309.50 MB	Launch
<input type="checkbox"/>	admin	cirros-0.5.2-x86_64	Image	Active	Public	No	QCOW2	15.55 MB	Launch
<input type="checkbox"/>	admin	cirros-0.6.2-x86_64	Image	Active	Public	No	QCOW2	20.44 MB	Launch
<input type="checkbox"/>	yoda	dev-yoda-ver1	Snapshot	Active	Image from Other Project - Non-Public	No	QCOW2	0 bytes	Launch
<input type="checkbox"/>	admin	manila-service-image	Image	Active	Image from Other Project - Non-Public	No	QCOW2	798.25 MB	Launch
<input type="checkbox"/>	admin	rocky8-x86_64	Image	Active	Public	No	QCOW2	1.84 GB	Launch
<input type="checkbox"/>	admin	rocky9-x86_64	Image	Active	Public	No	QCOW2	576.00 MB	Launch
<input type="checkbox"/>	admin	ubuntu-22.04-amd64-kvm	Image	Active	Public	No	QCOW2	587.94 MB	Launch
<input type="checkbox"/>	admin	ubuntu-22.04-server-amd64	Image	Active	Public	No	QCOW2	619.88 MB	Launch
<input type="checkbox"/>	swesc	ubuntu-2204-kube-v1.28.9	Image	Active	Image from Other Project - Non-Public	No	QCOW2	4.71 GB	Launch
<input type="checkbox"/>	swesc	ubuntu-2204-kube-v1.31.4	Image	Active	Image from Other Project - Non-Public	No	QCOW2	4.72 GB	Launch
<input type="checkbox"/>	demo	ubuntu-24.02	Image	Active	Image from Other Project - Non-Public	No	QCOW2	585.29 MB	Launch

Displaying 12 items

# The cloud is up – check out Horizon: **Networks**



# The cloud is up – check out Horizon: Security Groups

openstack. franz

Project / Network / Security Groups

## Security Groups

Filter  + Create Security Group Delete Security Groups

Displaying 1 item

<input type="checkbox"/>	Name	Security Group ID	Description	Shared	Actions
<input type="checkbox"/>	default	0811f6f8-fcc-4b72-84c8-935d90ee4e53	Default security group	False	Manage Rules

Displaying 1 item

Project

- API Access
- Compute
- Volumes
- Network
  - Network Topology
  - Networks
  - Routers
  - Security Groups**
  - Load Balancers
  - Floating IPs
- Orchestration
- Share
- Admin
- Identity

# The cloud is up – check out Horizon: Security Groups

The screenshot shows the OpenStack Horizon web interface. The top navigation bar includes the OpenStack logo, the user name 'franz', and a dropdown menu. The left sidebar contains a navigation tree with categories like Project, API Access, Compute, Volumes, Network, Network Topology, Networks, Routers, Security Groups (highlighted in blue), Load Balancers, Floating IPs, Orchestration, Share, Admin, and Identity. The main content area displays the breadcrumb 'Project / Network / Security Groups / Manage Security Group Rule...' and the title 'Manage Security Group Rules: default (0811f6f8-ffcc-4b72-84c8-935d90ee4e53)'. There are two buttons: '+ Add Rule' and 'Delete Rules'. Below this, a table lists 8 security group rules. The table has columns for checkboxes, Direction, Ether Type, IP Protocol, Port Range, Remote IP Prefix, Remote Security Group, Description, and Actions. Each row has a 'Delete Rule' button in the Actions column.

Project / Network / Security Groups / Manage Security Group Rule...

## Manage Security Group Rules: default (0811f6f8-ffcc-4b72-84c8-935d90ee4e53)

+ Add Rule Delete Rules

Displaying 8 items

<input type="checkbox"/>	Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Description	Actions
<input type="checkbox"/>	Egress	IPv4	Any	Any	0.0.0.0/0	-	-	Delete Rule
<input type="checkbox"/>	Egress	IPv6	Any	Any	:::0	-	-	Delete Rule
<input type="checkbox"/>	Ingress	IPv4	Any	Any	-	default	-	Delete Rule
<input type="checkbox"/>	Ingress	IPv4	ICMP	Any	0.0.0.0/0	-	-	Delete Rule
<input type="checkbox"/>	Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	-	Delete Rule
<input type="checkbox"/>	Ingress	IPv4	TCP	8000	0.0.0.0/0	-	-	Delete Rule
<input type="checkbox"/>	Ingress	IPv4	TCP	8080	0.0.0.0/0	-	-	Delete Rule
<input type="checkbox"/>	Ingress	IPv6	Any	Any	-	default	-	Delete Rule

Displaying 8 items

# The cloud is up – check out Horizon: **Volumes (storage)**

openstack. franz

Project / Volumes / Volumes

## Volumes

Filter

Name	Description	Size	Status	Group	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions
No items to display.										

Project

API Access

Compute

Volumes

Backups

Snapshots

Groups

Group Snapshots

Network

Orchestration

Share

Admin

Identity

# The cloud is up – check out Horizon: **Instances**

openstack. franz

Project / Compute / Instances

## Instances

Instance ID =  Filter [Launch Instance](#) [Delete Instances](#) More Actions ▾

Displaying 1 item

<input type="checkbox"/>	Instance Name	Image Name	IP Address	Flavor	Key Pair	Status	Availability Zone	Task	Power State	Age	Actions
<input type="checkbox"/>	test-cephfs	rocky9-x86_64	<b>Network-franz</b> 10.0.1.235, 129.16.122.190 <b>cephfs-net</b> 10.44.43.148	m1.small	franz-dev	Active	nova	None	Running	1 month	<a href="#">Create Snapshot</a> ▾

Displaying 1 item

Project

API Access

Compute

Overview

**Instances**

Images

Key Pairs

Server Groups

Volumes

Network

Orchestration

Share

Admin

Identity

# The cloud is up – Hands-on Session!

- Pick a project/user
- Log in at [dev-cloud.swesrc.chalmers.se](https://dev-cloud.swesrc.chalmers.se)
- Create a VM that you can log into from your laptop!

# The cloud is up – Hands-on Session!: Steps

- Upload/create an ssh keypair
- Create a router (pick 'dev-public' as external network)
- Create a network (not shared)
  - Create a subnet (pick a gateway IP from your Network Address range)
    - E.g., if Network Address is 10.3.0.0/23, you could pick 10.3.1.1
- Add your network to the router (Routers → “Add Interface” to the router you created earlier)
- Create a VM (aka: Launch an Instance)
  - “Create New Volume” → No
  - Pick an image
  - Pick a Flavor (for rocky and ubuntu go for at least m1.medium)
  - Pick your recently created network as the network
  - “Launch Instance”
  - Once it's up: Associate Floating IP

# The cloud is up – Hands-on Session!: Steps

- Try and ssh into your machine with the ssh-key you added at creation
  - For rocky images: rocky@<IP>
  - For Ubuntu: ubuntu@<IP>
  - For Cirros: cloud@<IP>
- **Ensure you have the correct Security Rules in place!**
  - Make sure the security group associated with your VM has ssh-access enabled (TCP traffic on port 22; ideally limited to access from Chalmers network)

# The cloud is up – Hands-on Session!: Steps

openstack. seminar1 user1

Project / Network / Security Groups / Manage Security Group Rul...

## Manage Security Group Rules: default (2f1d0f9f-25fb-4dce-8e4a-d2fd7864aa52)

+ Add Rule Delete Rules

Displaying 6 items

<input type="checkbox"/>	Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Description	Actions
<input type="checkbox"/>	Egress	IPv4	Any	Any	0.0.0.0/0	-	-	Delete Rule
<input type="checkbox"/>	Egress	IPv6	Any	Any	:::0	-	-	Delete Rule
<input type="checkbox"/>	Ingress	IPv4	Any	Any	-	default	-	Delete Rule
<input type="checkbox"/>	Ingress	IPv4	ICMP	Any	129.16.0.0/16	-	-	Delete Rule
<input type="checkbox"/>	Ingress	IPv4	TCP	22 (SSH)	129.16.0.0/16	-	-	Delete Rule
<input type="checkbox"/>	Ingress	IPv6	Any	Any	-	default	-	Delete Rule

Displaying 6 items

# The cloud is up – Hands-on Session!: **Alternative access via Horizon**

The screenshot displays the OpenStack Horizon web interface. The top navigation bar shows the OpenStack logo, the project name 'seminar1', and the user 'user1'. The left sidebar contains a navigation menu with categories: Project, API Access, Compute, Overview, Instances (highlighted), Images, Key Pairs, and Server Groups. Under 'Instances', there are sub-items: Volumes, Network, Orchestration, Share, and Identity. The main content area shows the breadcrumb 'Project / Compute / Instances / test3' and the instance name 'test3' with a 'Create Snapshot' button. Below this are tabs for 'Overview', 'Interfaces', 'Log', 'Console' (selected), and 'Action Log'. The 'Instance Console' section has a light blue warning banner: 'If console is not responding to keyboard input: click the grey status bar below. [Click here to show only console](#). To exit the fullscreen mode, click the browser's back button.' Below the banner is a dark grey console window with a blue header 'Connected to QEMU (Instance-00002cc)' and a 'Send Ctrl+Alt+Del' button. The console displays kernel boot logs and system initialization messages, including memory management and GPT table checks. At the bottom, it shows the login prompt: 'login as 'cirros' user, default password: 'gocubsgo', use 'sudo' for root. test3 login:'.

```
[ 1.405365] eum: HMAC attrs: 0x1
[ 1.406600] PM:   Magic number: 2:576:29
[ 1.407902] BAS: Correctable Errors collector initialized.
[ 1.410999] Freeing unused decrypted memory: 2036K
[ 1.413209] Freeing unused kernel image (initmem) memory: 3244K
[ 1.419732] Write protecting the kernel read-only data: 30720k
[ 1.422859] Freeing unused kernel image (text/rodata gap) memory: 2036K
[ 1.425163] Freeing unused kernel image (xdata/data gap) memory: 1448K
[ 1.481525] x86/mm: Checked U-X mappings: passed, no U-X pages found.
[ 1.483255] x86/mm: Checking user space page tables
[ 1.537424] x86/mm: Checked U-X mappings: passed, no U-X pages found.
[ 1.539140] Run /init as init process

further output written to /dev/ttyS0
[ 1.602933] virtio_blk virtio2: [vdw] 10485760 512-byte logical blocks (5.37
GB/5.00 GiB)
[ 1.621192] GPT:Primary header thinks Alt. header is not at the end of the di
sk.
[ 1.623227] GPT:229375 != 10485759
[ 1.624220] GPT:Alternate GPT header not at the end of the disk.
[ 1.625833] GPT:229375 != 10485759
[ 1.626828] GPT: Use GNU Parted to correct GPT errors.
[ 1.629888] virtio_blk virtio3: [vdw] 62914560 512-byte logical blocks (32.2
GB/30.0 GiB)
[ 1.686998] virtio_gpu virtio0: [drm] drm_plane_enable_fb_damage_clips() not called

login as 'cirros' user, default password: 'gocubsgo', use 'sudo' for root.
test3 login:
```

# The cloud is up – Hands-on Session!: Steps

- Try and ssh into your machine with the ssh-key you added at creation
  - For rocky images: rocky@<IP>
  - For Ubuntu: ubuntu@<IP>
  - For Cirros: cloud@<IP>
- Add a volume
  - Create one of size, say, 1 GB
  - Attach it to your VM
  - Once that's done, you will still need to add a partition table and create a file system on the device

# Doing the steps above via the CLI

- Create ssh-keypair:
  - `openstack keypair create <key_name> [--public-key <path-to-existing-key>]`
- Create a router:
  - `openstack router create <router_name>`
- Create a network
  - `openstack network create <network_name>`
- Create a subnet
  - `openstack subnet create <subnet_name>`
    - `--network <network_name>`
    - `--subnet-range 10.10.1.0/24`
    - `--gateway 10.10.1.1`
- Add Network to the router
  - `openstack router add subnet <router_name> <subnet_name>`
- Set router's external network to enable floating IP access
  - `openstack router set --external-gateway dev-public <router_name>`

# Doing the steps above via the CLI

- Create server/instance/virtual machine
  - `openstack server create <server_name>`
    - `--image <image>`
    - `--flavor <flavor>`
    - `--network <network_name>`
    - `--key-name <key_name>`
- Associate floating IP
  - Create a new floating IP first
    - `openstack floating ip create dev-public`
  - Add this floating IP to your server
    - `openstack server add floating ip <server_name> <floating-ip-address>`
- Fix security rules
  - Without initial specification, the default security group added to the VM will be the security group called “default”
  - But we can create new ones and add them to the server (also remove the default one if we want)

# Doing the steps above via the CLI

- Fix security rules (continued...)
  - To just add ssh-access to the existing 'default' group:
    - `openstack security group rule create default`  
`--protocol tcp --dst-port 22 --ingress --remote-ip 129.16.0.0/16`
  - Create a new group, add ssh-access, remove 'default' from and add new group to server
    - `openstack security group create <sg_name>`
    - `openstack security group rule create <sg_name>`  
`--protocol tcp --dst-port 22 --ingress --remote-ip 129.16.0.0/16`
    - `openstack server add security group <server_name> <sg_name>`
    - `openstack server remove security group <server_name> default`
  - Names in OpenStack are not unique, i.e. there may very well exist several servers/networks/subnets/routers/volumes/security groups with the same name. In that case one needs to replace the names above by the actual ID of that particular service; this can be determined by running `openstack [server, network, subnet...] list`

# Doing the steps above via the CLI

- Create a volume
  - `openstack volume create --size <size_in_GB> <volume_name>`
- Add volume to server
  - `openstack server add volume <server_name> <volume_name>`

# Controlling where VMs are scheduled

## Host aggregates

- Group hosts according to certain metadata (e.g. “gpu=true”; and/or “ssd=true”)
- Essentially helps the scheduler to place VMs on the right host
- Invisible to the user
- Same host can be in several different aggregates

## Availability Zones

- A special kind of host aggregates (metadata key: availability\_zone=<zone>)
- Same host can be in at most one AZ
- Visible to the user; i.e. they can choose which AZ their resources are placed in

Time for Part II\*

\*If time permits we can play a little more towards the end.

# Projects, Users, Groups and Roles

- Managed internally by Keystone (Identity service)
- Used for authentication and authorization
- Authentication - Who is this person? Are they who they claim they are?
  - Can map OS-keystone user to an IdP such as KeyCloak or Active Directory
  - Can also create users on demand from other identity providers (federation)
- Authorization - What is this person allowed to do? Are they a regular user or an administrator? Which resources are they allowed to manage?
- Handled by reference to roles and project membership
  - Can be done by group mapping in the IdP
- Questions to be resolved:
  - Bootstrapping – How does a user log in the first time?
  - Persistence – do we create ephemeral or persistent users?

# Users in OpenStack

OpenStack has a relatively minimalistic user model:

- Username (string)
  - Enforced to be unique if and only if user is persistent
- Email (optional)
- Domain
- UUID
- Description (Free text)
- Enabled (Boolean)
- Primary Project
- Type – Local (persistent) or Ephemeral. Ephemeral is the **default** when using an external IdP

We want to:

- Securely link verifiable physical-biological persons to an identifiable OpenStack user
- Allow them to log in by identifying themselves

# Groups and Roles

## Role

- e.g. Reader, User, Manager, Admin
- A role is set to manage a privilege but just specifies a particular “level” of privilege, rather than rich permissions
- Actual privileges bestowed through roles is defined by each service (e.g. Keystone gives certain rights to Managers for identity)
- A role is always assigned with a project link (a user is given a project-role)

## Group

- Plain old list of users
- Something like Active Directory Organisational Units (OU)
- Roles can be mapped to groups and projects to automatically give users roles
- External identity providers can specify group membership

# Algebra of user-role mapping

## Direct assignment

USER	GROUP	ROLE	PROJECT
LEONIEL		MEMBER	PROJ1

## Group-based assignment

USER	GROUP	ROLE	PROJECT
	PROJ12-MEMBERS	MEMBER	PROJ1
	PROJ12-MEMBERS	MEMBER	PROJ2
LEONIEL	PROJ12-MEMBERS		
LEONIEL		MEMBER	PROJ1
LEONIEL		MEMBER	PROJ2

# Projects

- Projects own VM:s and storage containers
  - Membership of a project confers the right to access and/or manage those resources
  - Membership is equivalent to having a *role* like **member** or **admin** scoped to that project.
- Projects are themselves sparse objects
  - id, name, description, domain\_id, enabled
  - options (arbitrary kv-pairs)
  - tags (arbitrary list)
  - Parents and children (projects can be nested)
- Many rich objects like resource quotas reference projects, making them a central “pivot” object that OpenStack revolves around
- An “admin” project owns all resources

# Rich and sparse objects

## Project (sparse)

```
{"description": "",  
  "domain_id": "default",  
  "enabled": true,  
  "id": "b1902...",  
  "is_domain": false,  
  "name": "demo",  
  "options": {},  
  "parent_id": "default",  
  "tags": []}
```

## Flavor (rich)

```
{"disabled": false,  
  "ephemeral": 30,  
  "access_project_ids": null,  
  "description": null,  
  "disk": 10,  
  "id": "2",  
  "name": "m1.small",  
  "is_public": true,  
  "properties": {},  
  "ram": 2048,  
  "rxtx_factor": 1.0,  
  "swap": 0,  
  "vcpus": 1}
```

## Compute quota (rich)

```
[{"Resource": "volumes",  
  "Limit": 10},  
 {"Resource": "snapshots",  
  "Limit": 10},  
 {"Resource": "gigabytes",  
  "Limit": 1000},  
 {"Resource": "backups",  
  "Limit": 10},  
 {"Resource": "groups",  
  "Limit": 10},  
 {"Resource": "backup-gigabytes",  
  "Limit": 1000},  
 {"Resource": "per-volume-gigabytes",  
  "Limit": -1}]
```

# Database perspective

Project - simple object (id, name, domain, etc)

User - simple object (id, name, domain, etc)

Role - simple object (id, name, domain)

Group - simple object (id, name, domain)

Assignments - foreignkey objects (User-Group, User-Role-Proj, Group-Role-Proj)

Quota - rich object (references project + quota settings like number of cores)

Flavor - rich object (has actual data, references projects that can access it)

Server - very rich object (specifies detailed VM properties, references project and user, inlines flavor information... )

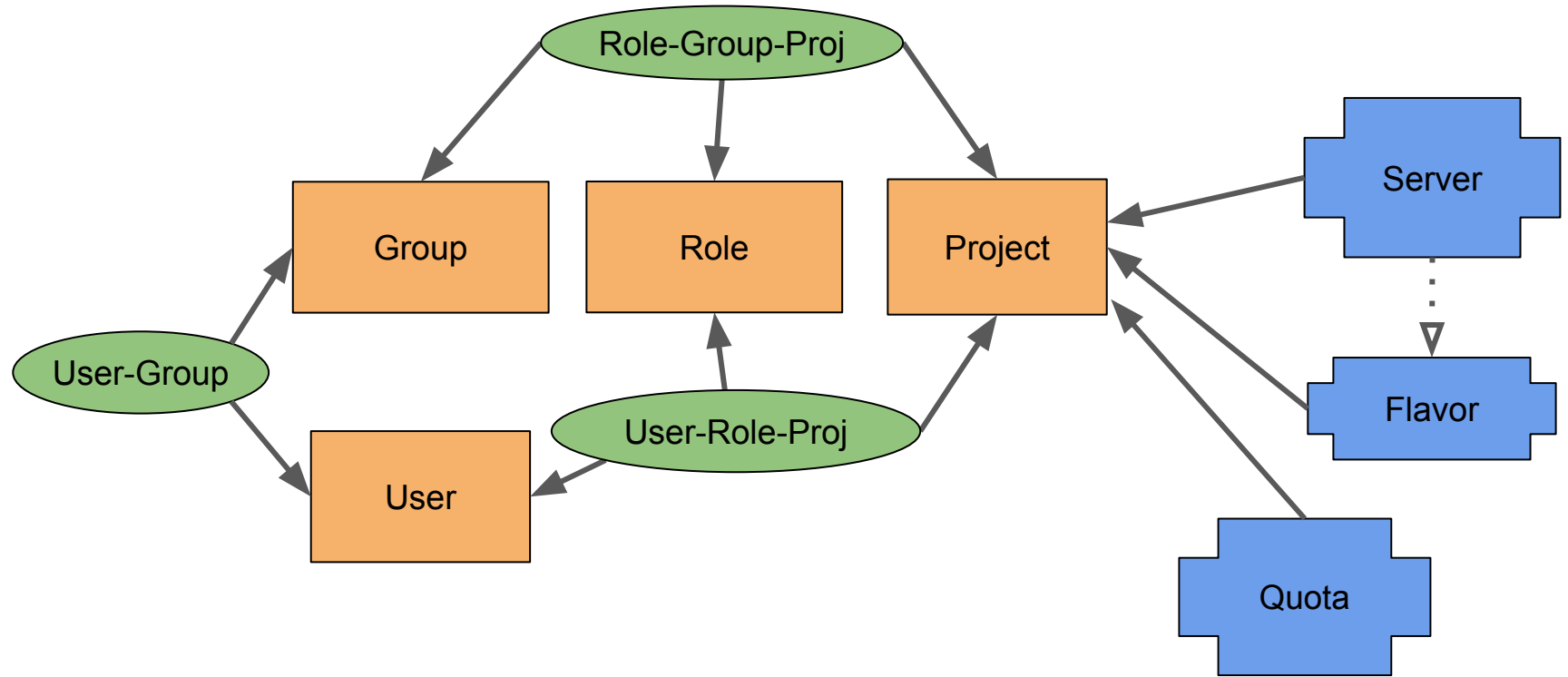
**Understanding this architecture is helpful when managing OpenStack!**

# Visualizing relations

Sparse

FK set

Rich



# Why SUPR?

- SUPR is already used for other services and was used for SSC
- Ensures identity of external users is verified
- Solves bootstrapping by allowing login through SUPR
- Can handle proposals, allocations and user administration
- External source of truth mitigates some risks
- Eliminates manual work in writing user-project management system
- Well-documented API for importing and exporting objects
- OpenStack limited when managing user and project metadata
- OpenStack Python bindings  $\Rightarrow$  implementation straightforward

# Relevant objects in SUPR

- Resource - E.g. cluster, storage, cloud - Alvis, Cephyr, C3SE-Cloud.
  - Resource has Core-hours/month or GiB storage. OpenStack cloud will use Coins/month - 1 coin
- Person – Represents physical-biological person with verified identity
- Project – Represents an allocation on a set of resources and has a list of members (persons)
  - ResourceProject: Allocation on a specific resource
    - Coins for cloud, quota for storage
- Account – A ForeignKey mapping that points to a Person and a Resource, along with a small amount of metadata (username, Enabled/Disabled state)

# Synchronizing with SUPR

- Goal: Create SUPR-resident projects and users in OpenStack, let OpenStack create accounts in SUPR
- Should be idempotent, consistent, not lead to duplication, etc.
- We can assume that the number of users and projects will not be very large

Solution: Fetch data in a stateless manner

- Fetch all active projects from SUPR and create missing OpenStack projects
- Fetch all SUPR account requests and create users
- Push accounts back to SUPR
- Add users to projects
- Use SUPR accounts to authenticate

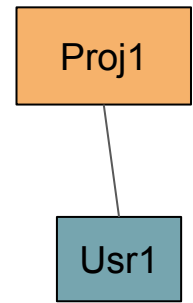
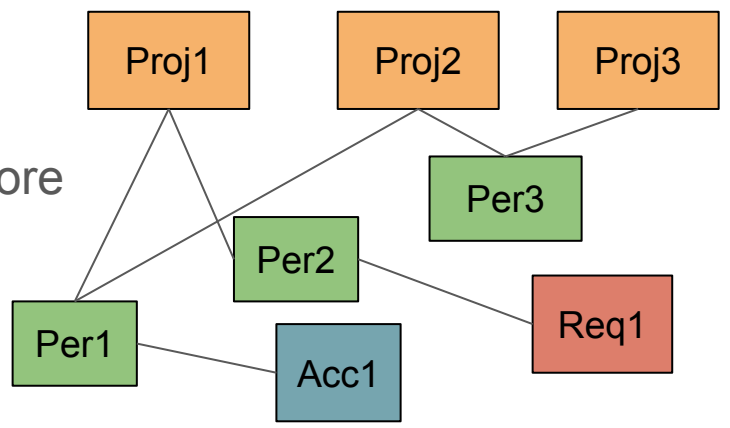
# Synchronizing with SUPR

- Fetch all active projects on Cloud resource in SUPR, enumerate all projects in OpenStack
- Ensure all active projects in SUPR exist in OpenStack (match by name), otherwise create them.
- Disable OpenStack projects that have expired in SUPR
- Create/update project resource quotas. (Restrict quotas if usage exceeds quota)
- Import account requests from SUPR and create OpenStack users
- Push accounts to SUPR, deleting the account request.
- Add users to projects (through roles) in OpenStack, enable accounts if disabled.
- Disable OpenStack users which do not have active or recently expired projects.
- (Optional) Synchronize account status in OpenStack with status in SUPR
  - By not synchronizing we can modify the SUPR status to ban users

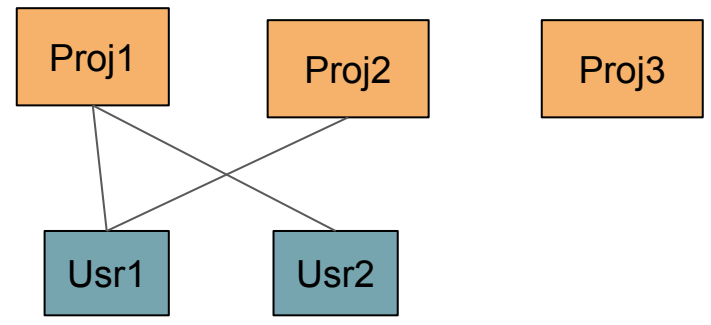
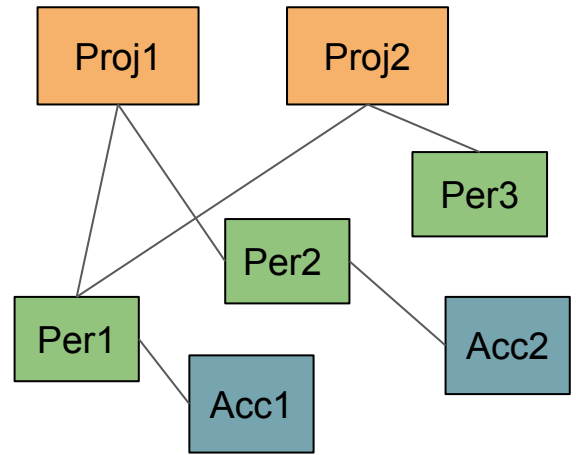
# SUPR

# OpenStack

Before



After



# OpenStack Python API

```
import openstack

class OpenstackObjects:
    def __init__(self, cloud):
        self.cloud = cloud
        self._connection = openstack.connect(cloud)

    @property
    def connection(self):
        return self._connection

    @property
    def member(self):
        return self.connection.identity.find_role('member')

    def get_projects(self):
        return self.connection.identity.projects()

    def get_users(self):
        return self.connection.identity.users()

    def get_servers(self):
        return self.connection.compute.servers(all_projects=True)

    def get_volumes(self):
        return self.connection.volume.volumes(all_projects=True)

    def get_backups(self):
        return self.connection.volume.backups(all_projects=True)
```

```
def set_project_storage_quota(self,
                               project_id,
                               storage_in_gb=None,
                               number_of_volumes=None,
                               number_of_snapshots=None,
                               number_of_backups=None):
    """
    Sets the 'gigabytes' storage quota which modifies the t
    of volumes and volume snapshots, as well as the number

    If any one value is not set, it is left unchanged.
    """
    kwargs = {}
    if storage_in_gb is not None:
        kwargs['gigabytes'] = storage_in_gb
    if number_of_volumes is not None:
        kwargs['volumes'] = number_of_volumes
    if number_of_snapshots is not None:
        kwargs['snapshots'] = number_of_snapshots
    if number_of_backups is not None:
        kwargs['backups'] = number_of_backups
    return self.connection.block_storage.update_quota_set(
        project_id, **kwargs)
```

# Combining with the SUPR API

```
def import_supr_projects(dry_run=False, verbose=False):
    supr = SUPR()
    # Search parameters
    params = {
        'resource_id': config['supr']['resource_id'], # C3SE
        'end_date_ge': datetime.date.today() - datetime.timedelta(days=30),
    }
    try:
        supr_projects = supr.get('/project/search/', params=params)
        supr_resource = supr.get(f'/resource/{config["supr"]["resource_id"]}')
    except SUPRHTTPError as e:
        # We want to show the text received if we get an HTTP Error
        logger.info("HTTP error {0} from SUPR:".format(e.status_code))
        logger.info(e.text)
        raise

    if verbose:
        logger.info("Currently there are {0} active projects at C3SE present in SUPR".format(
            len(supr_projects.matches)))
    # active_project_requests = ProjectRequest.objects.all().values_list('suprid', flat=True)
    openstack_projects = {o.name: o for o in openstack_objects.get_projects()}
    for supr_project in supr_projects.matches:
        if supr_project.name in openstack_projects:
            openstack_project = openstack_projects[supr_project.name]
        else:
            openstack_project = openstack_objects.create_project(supr_project.name)
    import_project_members(supr_project, openstack_project, supr_resource, dry_run)
```

# Handling account requests

- Using account requests is optional, we could just create accounts directly
- But account requests allow users to pick preferred usernames
  - Plus, we avoid creating users for people who are added to a project but never log in...
- User submits up to 3 preferred usernames
- We clean the 3 usernames (force lowercase, strip everything except [a-z0-9])
  - this way we can identify administrators by adding a prefix, since dashes will be stripped
- Check usernames against a profanity filter, discard if they fail the check
- See if remaining usernames already exist, discard if so
- If valid username remains, create user with first remaining one
- Otherwise use up to 8 letters of the user's given name, add numerical suffix until unique (leonard, leonard1, leonard2, leonard3 ...)
- Could use more complicated rule like Chalmers PDB but this risks creating inappropriate-looking usernames unless we use the profanity check, and some given names can fail a profanity check (Scunthorpe problem)

# Accounting

- Each flavour of VM has a cost in coins/hour
- An accounting service frequently (say once per minute) probes for running VM:s and looks up the cost of each VM, its associated project, and other metadata.
- A list of running VM:s and their cost per hour is fed into an SQL query that trapezoidally time-integrates the cost per hour into a total cost for each VM and puts that into a relation with a row for each VM and columns for the cost, integration time, and metadata.
- A second service regularly (say, once per hour) extracts chunks of time-integrated data which have been integrated over a sufficiently long time and puts in a “staging” relation for reporting.
- A third, daily service gets data and metadata from staging entries for reporting to SAMS. After reporting, data is placed into a third, archive relation.
- Utilities can be used to query locally for the usage of a project in a given time span, which can be used to impose restrictions on projects exceeding their allocations.

# Accounting

## Active

VM 1  
13:04 - 13:45  
10.3 coins  
metadata: { ... }

VM 2  
13:04 - 13:45  
4.7 coins  
metadata: { ... }

## Staging

VM 1  
12:04 - 13:04  
2.1 coins (not  
continuous)  
metadata: { ... }

VM 1  
10:45 - 11:30  
11 coins  
metadata: { ... }

VM 2  
12:04 - 13:04  
6.9 coins  
metadata: { ... }

## Archive

VM 1  
...

VM 1  
...

VM 1  
...

VM 2  
...

VM 2  
...

VM 2  
...

# Keycloak, SUPR and OpenID Connect

- Goal: Let users log in in a secure and verified manner
  - Use authentication against SUPR exposed over OpenID Connect
- Keycloak uses the SUPR Authentication API to let a user log in through SUPR.
- On successful login, Keycloak examines user metadata whether a user has an Account on a particular resource which is enabled.
- An ephemeral Keycloak user is created with the Account username that was originally pushed from OpenStack.
- User information is forwarded to OpenStack which matches the identifying user against the OpenStack user and logs them in.

keycloak\_mapping.json

```
[
  {
    "local": [
      {
        "user": {
          "name": "{0}",
          "type": "local",
          "domain": {
            "name": "default"
          }
        }
      }
    ],
    "remote": [
      {
        "type": "OIDC-preferred_username"
      }
    ]
  }
]
```

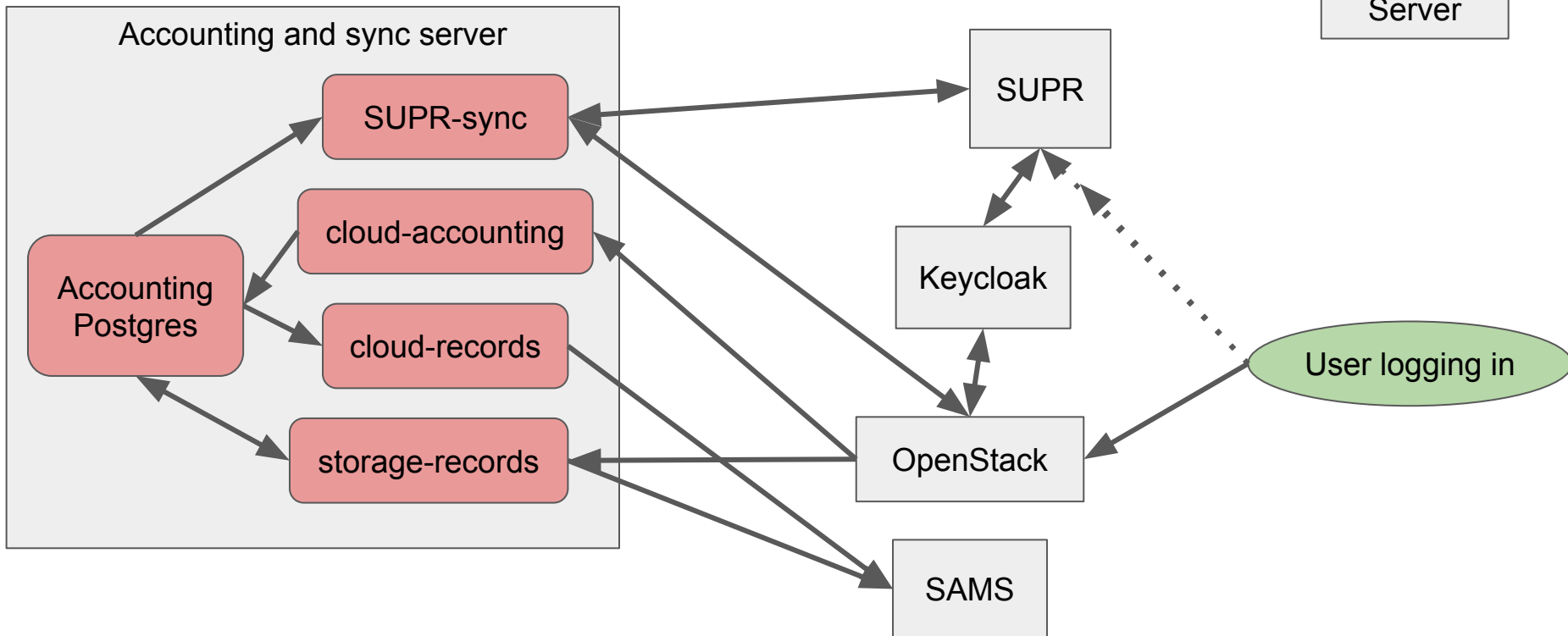
# Other approaches

- We could get active projects from SUPR and create an OpenID Connect claim from those
- The claim could be mapped to groups in OpenStack which automatically enroll users in projects
- But - no real advantage since we anyway need to set up projects and be able to enable and disable accounts in SUPR
- However group claims could be used managing administrative users with varying levels of privilege through Keycloak tooling

# Architecture and data flow

Service

Server



Back to Part I – more play time!\*

\*If time permits and everyone is keen.

# Just a quick word on quotas

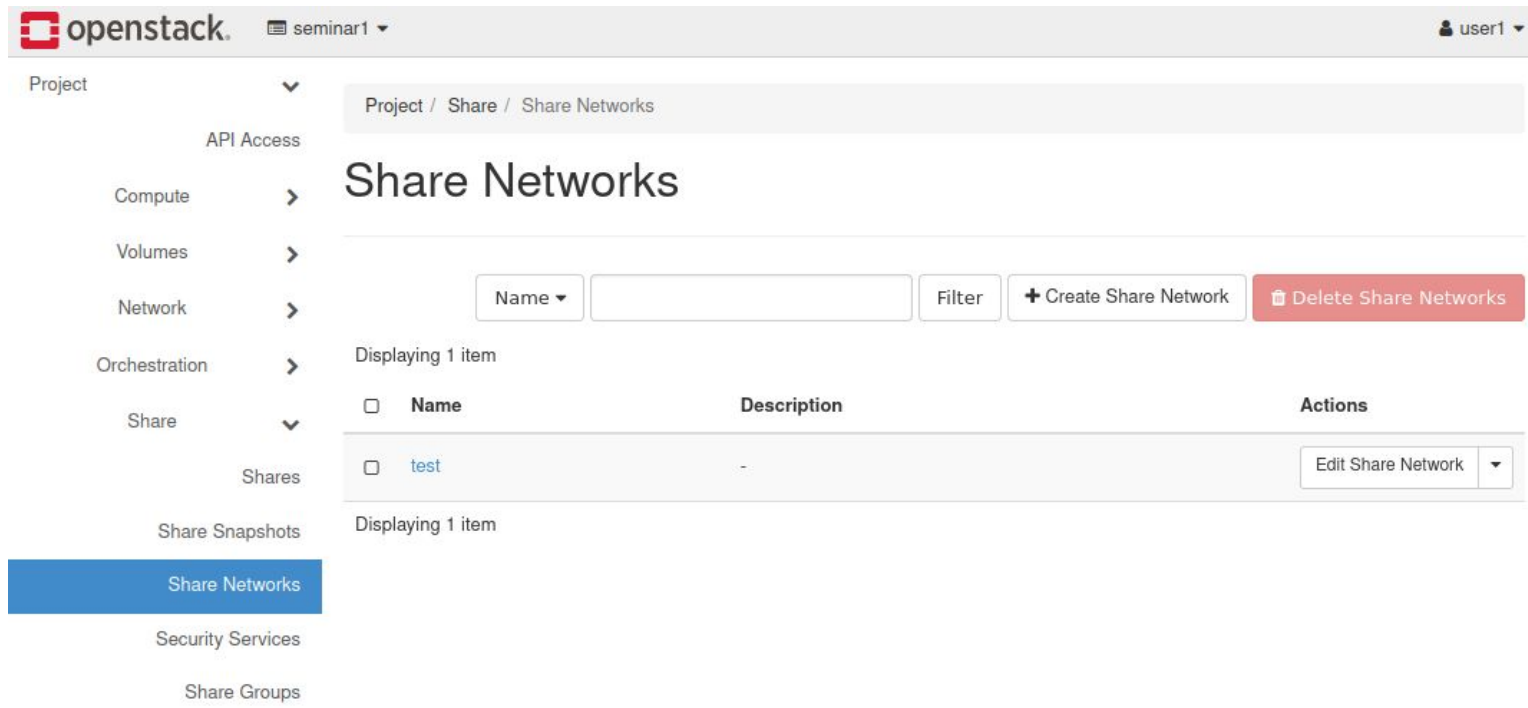
- There are default quotas set per project for pretty much everything:
  - Number of CPUs, routers, floating IPs, security groups...
  - Maximum size of volumes
  - Total maximum of allowed storage
  - Total maximum of allowed **shared** storage (yes it's counted separately)
- As admin these can be set per project
  - `openstack quota set --gigabytes 1024`
  - `openstack share quota set --gigabytes 1024`

# Let's create shared storage

- First we need to create a Share Network, i.e. a network through which VMs can access common storage

# Let's create shared storage

- First we need to create a Share Network, i.e. a network through which VMs can access common storage



The screenshot shows the OpenStack dashboard interface. At the top, the OpenStack logo is on the left, followed by the project name 'seminar1' and the user 'user1'. The left sidebar contains a navigation menu with categories: Project, API Access, Compute, Volumes, Network, Orchestration, Share (expanded), Shares, Share Snapshots, Share Networks (highlighted in blue), Security Services, and Share Groups. The main content area displays the 'Share Networks' page. It includes a breadcrumb 'Project / Share / Share Networks', a search bar with a 'Name' dropdown, a 'Filter' button, a '+ Create Share Network' button, and a 'Delete Share Networks' button. Below this, it states 'Displaying 1 item' and shows a table with columns for Name, Description, and Actions. The table contains one entry with the name 'test' and an 'Edit Share Network' action button.

Project / Share / Share Networks

## Share Networks

Name  Filter [+ Create Share Network](#) [Delete Share Networks](#)

Displaying 1 item

<input type="checkbox"/>	Name	Description	Actions
<input type="checkbox"/>	test	-	<a href="#">Edit Share Network</a>

Displaying 1 item

# Let's create shared storage

- First we need to create a Share Network, i.e. a network through which VMs can access common storage
- Pick as “neutron subnet” the subnet that you created for your VM(s)

## Create Share Network ✕

Share Network \*

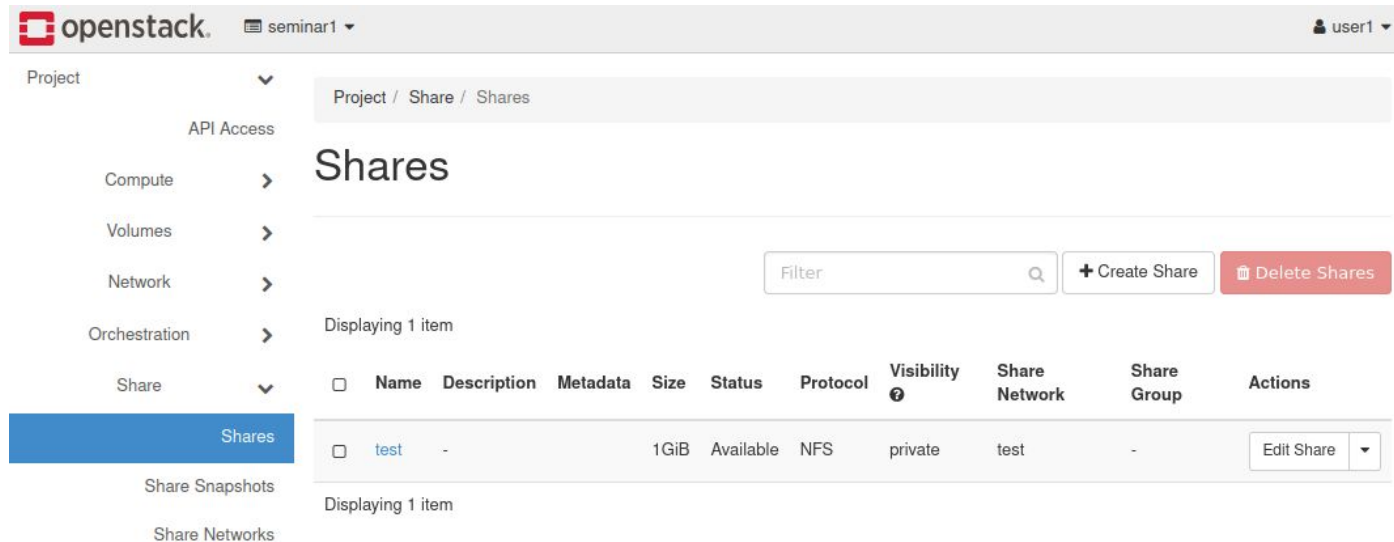
**Availability Zone**

**Neutron Net**  
  
None  
dev-public  
test  
cephfs-net

Specify an Availability Zone or an existing subnet. If no details are specified, then a default subnet with a null Availability Zone will be created automatically.

# Let's create shared storage

- First we need to create a Share Network, i.e. a network through which VMs can access common storage
- Pick as “neutron subnet” the subnet that you created for your VM(s)
- Create a share
  - Pick as share type “test\_type”, protocol: “nfs”, and your share network



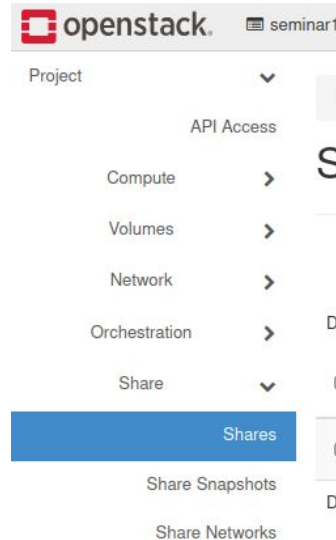
The screenshot shows the OpenStack dashboard interface. The top navigation bar includes the OpenStack logo, the project name 'seminar1', and the user 'user1'. The left sidebar contains a navigation menu with categories like Project, API Access, Compute, Volumes, Network, and Orchestration. The 'Share' category is expanded, and the 'Shares' sub-page is selected. The main content area is titled 'Shares' and includes a search filter, '+ Create Share' button, and 'Delete Shares' button. Below this, it states 'Displaying 1 item' and shows a table with the following data:

<input type="checkbox"/>	Name	Description	Metadata	Size	Status	Protocol	Visibility	Share Network	Share Group	Actions
<input type="checkbox"/>	test	-		1GiB	Available	NFS	private	test	-	Edit Share

Below the table, it states 'Displaying 1 item'.

# Let's create share

- First we need to pick a network through which VMs can access it
- Pick as "new" network
- Create a share
  - Pick as "new" share



openstack. seminar

- Project
- API Access
- Compute
- Volumes
- Network
- Orchestration
- Share
- Shares**
- Share Snapshots
- Share Networks

### Create Share

Share Name \*

Description

Share Protocol \*  
NFS

Size (GiB) \*

Share Type \*  
test\_type

Availability Zone

Share Group

Share Network \*  
test

Metadata

Make visible to users from all projects

Cancel Create

### Description:

Select parameters of share you want to create.

### Metadata:

One line - one action. Empty strings will be ignored.  
To add metadata use:

key=value

### Share Limits

Total Gibibytes 1 of 1,000 GiB Used

Number of Shares 1 of 50 Used

work through which VMs

for your VM(s)

and your share network

user1

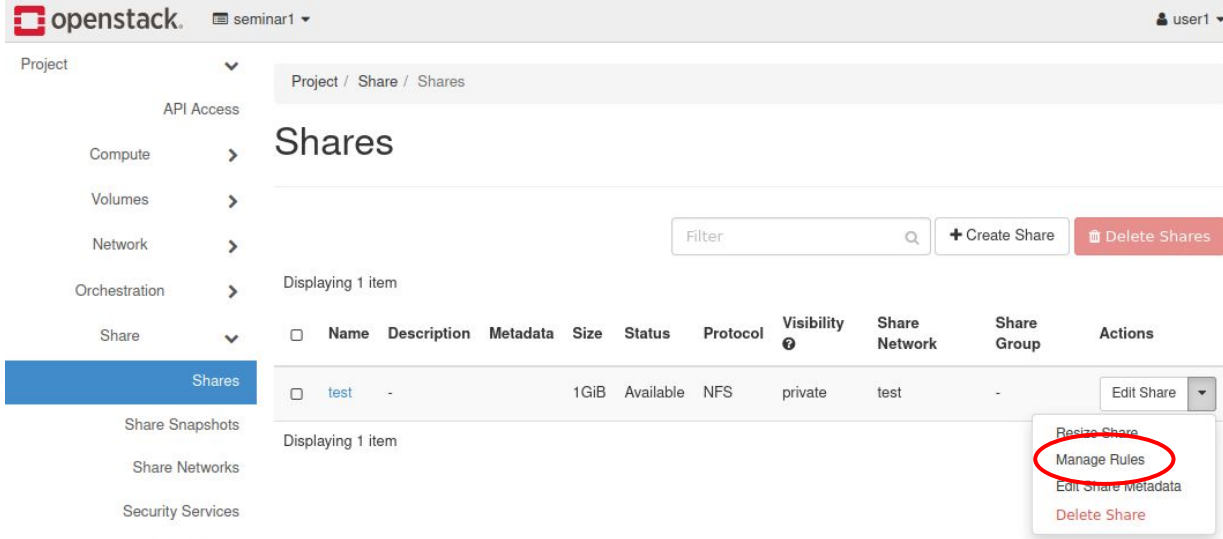
Delete Shares

### Actions

Edit Share

# Let's create shared storage

- First we need to create a Share Network, i.e. a network through which VMs can access common storage
- Pick as “neutron subnet” the subnet that you created for your VM(s)
- Create a share
  - Pick as share type “test\_type”, protocol: “nfs”, and your share network
- Add access rules; i.e. control which VM can access the share



The screenshot displays the OpenStack Shares management interface. The top navigation bar shows the OpenStack logo, the project name 'seminar1', and the user 'user1'. The left sidebar contains a navigation menu with 'Share' selected and 'Shares' highlighted. The main content area shows the 'Shares' page with a breadcrumb 'Project / Share / Shares'. Below the breadcrumb, there is a search filter, a '+ Create Share' button, and a 'Delete Shares' button. A table displays one share with the following details:

<input type="checkbox"/>	Name	Description	Metadata	Size	Status	Protocol	Visibility	Share Network	Share Group	Actions
<input type="checkbox"/>	test	-		1GiB	Available	NFS	private	test	-	Edit Share

The 'Actions' column for the 'test' share has a dropdown menu open, showing the following options: 'Resize Share', 'Manage Rules' (highlighted with a red circle), 'Edit Share Metadata', and 'Delete Share'.

# Let's create shared storage

- First we need to create a Share Network, i.e. a network through which VMs can access common storage
- Pick as “neutron subnet” the subnet that you created for your VM(s)
- Create a share
  - Pick as share type “test\_type”, protocol: “nfs”, and your share network
- Add access rules; i.e. control which VM can access the share

**Add Rule** ✕ user1 ▾

Project

**Access Type \***  
ip ▾

**Access Level \***  
read-write ▾

**Access To \***  
IP of VM on the Neutron Subnet

**Metadata**

**Description:**  
Add policy rule to share, 'ip' rule represents IPv4 or IPv6 address, 'user' rule represents username or usergroup, 'cephx' rule represents ceph auth ID.

[+ Create Share](#) [Delete Shares](#)

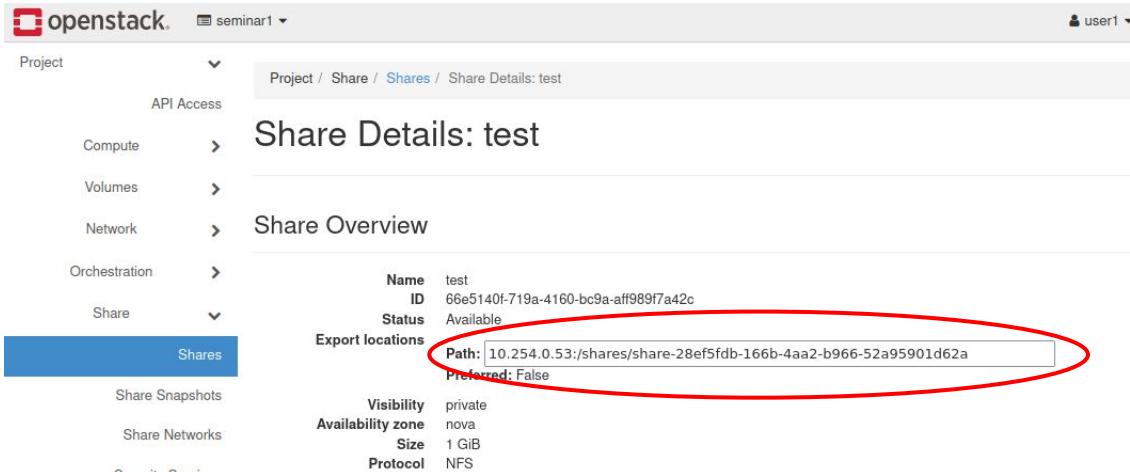
Share Group	Actions
-	Edit Share ▾

- Resize Share
- Manage Rules**
- Edit Share Metadata
- Delete Share

[Cancel](#) [Add](#)

# Let's create shared storage

- First we need to create a Share Network, i.e. a network through which VMs can access common storage
- Pick as “neutron subnet” the subnet that you created for your VM(s)
- Create a share
  - Pick as share type “test\_type”, protocol: “nfs”, and your share network
- Add access rules; i.e. control which VM can access the share
- Figure out the IP and path of the share



The screenshot shows the OpenStack dashboard interface. The top navigation bar includes the OpenStack logo, the project name 'seminar1', and the user 'user1'. The left sidebar contains a navigation menu with categories like Project, API Access, Compute, Volumes, Network, Orchestration, and Share. The 'Share' category is expanded, and the 'Shares' sub-item is highlighted in blue. The main content area displays the 'Share Details: test' page. Under the 'Share Overview' section, a table lists the share's properties. The 'Export locations' section is circled in red, highlighting the 'Path' field which contains the value '10.254.0.53:/shares/share-28ef5fdb-166b-4aa2-b966-52a95901d62a'. Other visible properties include Name (test), ID (66e5140f-719a-4160-bc9a-aff989f7a42c), Status (Available), Visibility (private), Availability zone (nova), Size (1 GiB), and Protocol (NFS).

<b>Name</b>	test
<b>ID</b>	66e5140f-719a-4160-bc9a-aff989f7a42c
<b>Status</b>	Available
<b>Export locations</b>	<b>Path:</b> 10.254.0.53:/shares/share-28ef5fdb-166b-4aa2-b966-52a95901d62a
	<b>Preferred:</b> False
<b>Visibility</b>	private
<b>Availability zone</b>	nova
<b>Size</b>	1 GiB
<b>Protocol</b>	NFS

# Let's create shared storage

- First we need to create a Share Network, i.e. a network through which VMs can access common storage
- Pick as “neutron subnet” the subnet that you created for your VM(s)
- Create a share
  - Pick as share type “test\_type”, protocol: “nfs”, and your share network
- Add access rules; i.e. control which VM can access the share
- Figure out the IP and path of the share
- On your VM, assuming you have “nfs-utils” installed, you can now mount the share using that path into a directory of your choosing.

# Let's create shared storage – via the CLI

- Create the share network
  - `openstack share network create --name <share_network_name> --neutron-subnet-id <subnet_id>`
- Create the share
  - `openstack share create <share_protocol (e.g. nfs)> <size (in GB)> --name <share_name> --share-network <share_network_name/ID> --share-type <type (e.g. test_type)>`
- Add access rules
  - `openstack share access create <share_name/id> ip <ip_address>`
- Figure out the IP and path to the share via
  - `openstack share show <share_name/id>`
- Mount the share in your VM using that IP and Path