SUSE OpenStack Cloud 8

Planning an Installation with Cloud Lifecycle Manager

Planning an Installation with Cloud Lifecycle Manager

SUSE OpenStack Cloud 8

Publication Date: 09/08/2022

SUSE LLC
1800 South Novell Place
Provo, UT 84606
USA
https://documentation.suse.com

All information found in this book has been compiled with utmost attention to detail. However, this does not guarantee complete accuracy. Neither SUSE LLC, its affiliates, the authors nor the translators shall be held liable for possible errors or the consequences thereof.

Contents

I Planning

1 Registering SLES

To get technical support and product updates, you need to register and activate your SUSE product with the SUSE Customer Center. It is recommended to register during the installation, since this will enable you to install the system with the latest updates and patches available. However, if you are offline or want to skip the registration step, you can register at any time later from the installed system.

Note

In case your organization does not provide a local registration server, registering SLES requires a SUSE account. In case you do not have a SUSE account yet, go to the SUSE Customer Center home page (https://scc.suse.com/ ↗) to create one.

1.1 Registering SLES during the Installation

To register your system, provide the E-mail address associated with the SUSE account you or your organization uses to manage subscriptions. In case you do not have a SUSE account yet, go to the SUSE Customer Center home page (https://scc.suse.com/ ↗) to create one.

Enter the Registration Code you received with your copy of SUSE Linux Enterprise Server. Proceed with *Next* to start the registration process.

By default the system is registered with the SUSE Customer Center. However, if your organization provides local registration servers you can either choose one form the list of auto-detected servers or provide the URL at `Register System via local SMT Server`. Proceed with *Next*.

During the registration, the online update repositories will be added to your installation setup. When finished, you can choose whether to install the latest available package versions from the update repositories. This ensures that SUSE Linux Enterprise Server is installed with the latest security updates available. If you choose No, all packages will be installed from the installation media. Proceed with Next.

If the system was successfully registered during installation, YaST will disable repositories from local installation media such as CD/DVD or flash disks when the installation has been completed. This prevents problems if the installation source is no longer available and ensures that you always get the latest updates from the online repositories.

1.2 Registering SLES from the Installed System

1.2.1 Registering from the Installed System

If you have skipped the registration during the installation or want to re-register your system, you can register the system at any time using the YaST module *Product Registration* or the command line tool **`SUSEConnect`**.

Registering with YaST

To register the system start *YaST* › *Software* › *Product Registration*. Provide the E-mail address associated with the SUSE account you or your organization uses to manage subscriptions. In case you do not have a SUSE account yet, go to the SUSE Customer Center homepage (https://scc.suse.com/) to create one.

Enter the Registration Code you received with your copy of SUSE Linux Enterprise Server. Proceed with *Next* to start the registration process.

By default the system is registered with the SUSE Customer Center. However, if your organization provides local registration servers you can either choose one form the list of auto-detected servers or provide the URl at *Register System via local SMT Server*. Proceed with *Next*.

Registering with SUSEConnect

To register from the command line, use the command

```
tux > sudo SUSEConnect -r REGISTRATION_CODE -e EMAIL_ADDRESS
```

Replace *`REGISTRATION_CODE`* with the Registration Code you received with your copy of SUSE Linux Enterprise Server. Replace *`EMAIL_ADDRESS`* with the E-mail address associated with the SUSE account you or your organization uses to manage subscriptions. To register with a local registration server, also provide the URL to the server:

```
tux > sudo SUSEConnect -r REGISTRATION_CODE -e EMAIL_ADDRESS \
--url "https://suse_register.example.com/"
```

1.3 Registering SLES during Automated Deployment

If you deploy your instances automatically using AutoYaST, you can register the system during the installation by providing the respective information in the AutoYaST control file. Refer to https://documentation.suse.com/sles/12-SP5/single-html/SLES-autoyast/#CreateProfile-Register ↗ for details.

2 Hardware and Software Support Matrix

This document lists the details about the supported hardware and software for SUSE OpenStack Cloud 8

2.1 OpenStack Version Information

SUSE OpenStack Cloud 8 services have been updated to the OpenStack Pike (https://www.openstack.org/software/pike) release.

2.2 Supported Hardware Configurations

SUSE OpenStack Cloud 8 supports hardware that is certified for SLES through the YES certification program. You will find a database of certified hardware at https://www.suse.com/yessearch/.

2.3 Support for Core and Non-Core OpenStack Features

OpenStack Service	**Packages**	**Supported**		**OpenStack Service**	**Packages**	**Supported**
Aodh	No	No		Barbican	Yes	Yes
Ceilometer	Yes	Yes		Cinder	Yes	Yes
Designate	Yes	Yes		Freezer	Yes	Yes
Glance	Yes	Yes		Heat	Yes	Yes
Horizon	Yes	Yes		Ironic	Yes	Yes
Keystone	Yes	Yes		Magnum	Yes	Yes
Manila	Yes	Yes		Monasca	Yes	Yes

Monasca-Ceilometer	Yes	Yes		Neutron	Yes	Yes
Neutron(LBaaSv2)	Yes	Yes		Neutron(VPNaaS)	Yes	Yes
Neutron(FWaaS)	Yes	Yes		Nova	Yes	Yes
Octavia	Yes	Yes		Swift	Yes	Yes

Nova

Supported	Not Supported
SLES KVM Hypervisor	Xen hypervisor
VMware ESX Hypervisor	Hyper-V
	Non-x86 Architectures

Neutron

Supported	Not Supported
Tenant networks • IPv6 • SR-IOV • PCI-PT • DPDK	Distributed Virtual Router (DVR) with any of the following: • IPv6 • BGP/Fast Path Exit • L2 gateway • SNAT HA
VMware ESX Hypervisor	QoS

Glance Supported Features

- Swift and Ceph backends

Cinder

Supported	Not Supported
Encrypted & private volumes	VSA
Incremental backup, backup attached volume, encrypted volume backup, backup snapshots	

Swift

Supported	Not Supported
Erasure coding	Geographically distributed clusters
Dispersion report	
Swift zones	

Keystone

Supported	Not Supported
Domains	Web SSO
Fernet tokens	Multi-Factor authentication
LDAP integration	Federation Keystone to Keystone
	Hierarchical multi-tenancy

Barbican Supported Features

- Encryption for the following:
  - Cinder
  - Hardware security model

    - Encrypted data volumes
    - LBaaS
    - Symmetric keys
    - Storage keys
- CADF format auditing events

Ceilometer

Supported	Not Supported
Keystone v3 support	Gnocchi
Glance v2 API	IPMI and SNMP
	Ceilometer Event APIs
	Ceilometer Compute Agent

Heat Features Not Supported

- Multi-region stack

Ironic

Supported	Not Supported
Drivers • Agent_ilo • Agent_ipmitool • PXE_ilo • PXE_ipmitool	UEFI secure
Booting methods	
Power	

Supported	Not Supported
Compute	
Networking	
Images	

Freezer

Supported	Not Supported
Backup of Control Plane data	Backup of tenant VMs, volumes, files, directories and databases
Backup Lifecycle Manager	
Backup of audit logs	
Backup of centralized logging files	

2.4 Cloud Scaling

In SUSE OpenStack Cloud 8 a total of 200 total compute nodes in a single region (Region0) across any of the following hypervisors is supported:

- VMware ESX
- SLES/KVM

You can distribute the compute nodes in any number of deployments as long as the total is no more than 200. Example: 100 ESX + 100 KVM or 50 ESX + 150 KVM.

SUSE OpenStack Cloud 8 supports a total of 8000 virtual machines across a total of 200 compute nodes.

2.5 Supported Software

Supported ESXi versions

SUSE OpenStack Cloud 8 currently supports the following ESXi versions:

- ESXi version 6.0
- ESXi version 6.0 (Update 1b)
- ESXi version 6.5

The following are the requirements for your vCenter server:

- Software: vCenter (It is recommended to run the same server version as the ESXi hosts.)
- License Requirements: vSphere Enterprise Plus license

2.6 Notes About Performance

We have the following recommendations to ensure good performance of your cloud environment:

- On the control plane nodes, you will want good I/O performance. Your array controllers must have cache controllers and we advise against the use of RAID-5.
- On compute nodes, the I/O performance will influence the virtual machine start-up performance. We also recommend the use of cache controllers in your storage arrays.
- If you are using dedicated object storage (Swift) nodes, in particular the account, container, and object servers, we recommend that your storage arrays have cache controllers.
- For best performance on, set the servers power management setting in the iLO to OS Control Mode. This power mode setting is only available on servers that include the HP Power Regulator.

2.7 KVM Guest OS Support

For a list of the supported VM guests, see https://documentation.suse.com/sles/12-SP5/single-html/SLES-virtualization/#virt-support-guests ↗

2.8 ESX Guest OS Support

For ESX, refer to the VMware Compatibility Guide (https://www.vmware.com/resources/compatibility/search.php?deviceCategory=software&details=1&releases=273,274,338&productNames=15&page=1&display_interval=500&sortColumn=Partner&sortOrder=Asc&testConfig=16) ↗. The information for SUSE OpenStack Cloud is below the search form.

2.9 Ironic Guest OS Support

A **Verified** Guest OS has been tested by SUSE and appears to function properly as a bare metal instance on SUSE OpenStack Cloud 8.

A **Certified** Guest OS has been officially tested by the operating system vendor, or by SUSE under the vendor's authorized program, and will be supported by the operating system vendor as a bare metal instance on SUSE OpenStack Cloud 8.

Ironic Guest Operating System	Verified	Certified
SUSE Linux Enterprise Server 12 SP3	Yes	Yes

3 Recommended Hardware Minimums for the Example Configurations

3.1 Recommended Hardware Minimums for an Entry-scale KVM

These recommended minimums are based on example configurations included with the installation models (see *Chapter 9, Example Configurations*). They are suitable only for demo environments. For production systems you will want to consider your capacity and performance requirements when making decisions about your hardware.

Note

The disk requirements detailed below can be met with logical drives, logical volumes, or external storage such as a 3PAR array.

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	**Memory**	**Network**	**CPU**
Dedicated Cloud Lifecycle Manager (optional)	Lifecycle-manager	1	300 GB	8 GB	1 x 10 Gbit/s with PXE Support	8 CPU (64-bit) cores total (Intel x86_64)
Control Plane	Controller	3	• 1 x 600 GB (minimum) - operating	128 GB	2 x 10 Gbit/s with one PXE enabled port	8 CPU (64-bit) cores total (Intel x86_64)

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
			system drive • 2 x 600 GB (minimum) - Data drive • Fast disks or SSDs are recommended.			
Compute	Compute	1-3	2 x 600 GB (minimum)	32 GB (memory must be sized based on the virtual machine instances hosted on the Compute node)	2 x 10 Gbit/s with one PXE enabled port	8 CPU (64-bit) cores total (Intel x86_64) with hardware virtualization support. The CPU cores must be sized

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
						based on the VM instances hosted by the Compute node.

For more details about the supported network requirements, see *Chapter 9, Example Configurations*.

3.2 Recommended Hardware Minimums for an Entry-scale ESX KVM Model

These recommended minimums are based on example configurations included with the installation models (see *Chapter 9, Example Configurations*). They are suitable only for demo environments. For production systems you will want to consider your capacity and performance requirements when making decisions about your hardware.

SUSE OpenStack Cloud currently supports the following ESXi versions:

- ESXi version 6.0
- ESXi version 6.0 (Update 1b)
- ESXi version 6.5

The following are the requirements for your vCenter server:

- Software: vCenter (It is recommended to run the same server version as the ESXi hosts.)
- License Requirements: vSphere Enterprise Plus license

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	**Memory**	**Network**	**CPU**
Dedicated Cloud Lifecycle Manager (optional)	Lifecycle-manager	1	300 GB	8 GB	1 x 10 Gbit/s with PXE Support	8 CPU (64-bit) cores total (Intel x86_64)
Control Plane	Controller	3	• 1 x 600 GB (minimum) - operating system drive • 2 x 600 GB (minimum) - Data drive • Fast disks or SSDs are rec-	128 GB	2 x 10 Gbit/s with one PXE enabled port	8 CPU (64-bit) cores total (Intel x86_64)

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
			ommended.			
Compute (ESXi hypervisor)		2	2 x 1 TB (minimum, shared across all nodes)	128 GB (minimum)	2 x 10 Gbit/s +1 NIC (for DC access)	16 CPU (64-bit) cores total (Intel x86_64)
Compute (KVM hypervisor)	kvm-compute	1-3	2 x 600 GB (minimum)	32 GB (memory must be sized based on the virtual machine instances hosted on the Compute node)	2 x 10 Gbit/s with one PXE enabled port	8 CPU (64-bit) cores total (Intel x86_64) with hardware virtualization support. The CPU cores must be sized based on the VM instances hosted by the Compute node.

3.3 Recommended Hardware Minimums for an Entry-scale ESX, KVM with Dedicated Cluster for Metering, Monitoring, and Logging

These recommended minimums are based on example configurations included with the installation models (see *Chapter 9, Example Configurations*). They are suitable only for demo environments. For production systems you will want to consider your capacity and performance requirements when making decisions about your hardware.

SUSE OpenStack Cloud currently supports the following ESXi versions:

- ESXi version 6.0
- ESXi version 6.0 (Update 1b)
- ESXi version 6.5

The following are the requirements for your vCenter server:

- Software: vCenter (It is recommended to run the same server version as the ESXi hosts.)
- License Requirements: vSphere Enterprise Plus license

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
Dedicated Cloud Lifecycle Manager (optional)	Lifecycle-manager	1	300 GB	8 GB	1 x 10 Gbit/s with PXE Support	8 CPU (64-bit) cores total (Intel x86_64)
Control Plane	Core-API Controller	2	• 1 x 600 GB (minimum) - operat-	128 GB	2 x 10 Gbit/s with PXE Support	24 CPU (64-bit) cores total (Intel x86_64)

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
			ing system drive • 2 x 300 GB (minimum) - Swift drive			
	DBMQ Cluster	3	• 1 x 600 GB (minimum) - operating system drive • 1 x 300 GB (minimum) -	96 GB	2 x 10 Gbit/s with PXE Support	24 CPU (64-bit) cores total (Intel x86_64)

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
			MariaDB drive			
	Metering Mon/Log Cluster	3	• 1 x 600 GB (minimum) - operating system drive	128 GB	2 x 10 Gbit/s with one PXE enabled port	24 CPU (64-bit) cores total (Intel x86_64)
Compute (ESXi hypervisor)		2 (minimum)	2 X 1 TB (minimum, shared across all nodes)	64 GB (memory must be sized based on the virtual machine instances hosted on the Compute node)	2 x 10 Gbit/s +1 NIC (for Data Center access)	16 CPU (64-bit) cores total (Intel x86_64)
Compute (KVM hypervisor)	kvm-compute	1-3	2 X 600 GB (minimum)	32 GB (memory must be sized based on the vir-	2 x 10 Gbit/s with one PXE enabled port	8 CPU (64-bit) cores total (Intel x86_64) with hard-

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
				tual machine instances hosted on the Compute node)		ware virtualization support. The CPU cores must be sized based on the VM instances hosted by the Compute node.

3.4 Recommended Hardware Minimums for an Ironic Flat Network Model

When using the `agent_ilo` driver, you should ensure that the most recent iLO controller firmware is installed. A recommended minimum for the iLO4 controller is version 2.30.

The recommended minimum hardware requirements are based on the *Chapter 9, Example Configurations* included with the base installation and are suitable only for demo environments. For production systems you will want to consider your capacity and performance requirements when making decisions about your hardware.

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
Dedicated Cloud Lifecycle Man-	Lifecycle-manager	1	300 GB	8 GB	1 x 10 Gbit/s with PXE Support	8 CPU (64-bit) cores total (Intel x86_64)

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
ager (optional)						
Control Plane	Controller	3	• 1 x 600 GB (minimum) - operating system drive • 2 x 600 GB (minimum) - Data drive • Fast disks or SSDs are recommended.	128 GB	2 x 10 Gbit/s with one PXE enabled port	8 CPU (64-bit) cores total (Intel x86_64)

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
Compute	Compute	1	1 x 600 GB (minimum)	16 GB	2 x 10 Gbit/s with one PXE enabled port	16 CPU (64-bit) cores total (Intel x86_64)

For more details about the supported network requirements, see *Chapter 9, Example Configurations*.

3.5 Recommended Hardware Minimums for an Entry-scale Swift Model

These recommended minimums are based on the included *Chapter 9, Example Configurations* included with the base installation and are suitable only for demo environments. For production systems you will want to consider your capacity and performance requirements when making decisions about your hardware.

The `entry-scale-swift` example runs the Swift proxy, account and container services on the three controller servers. However, it is possible to extend the model to include the Swift proxy, account and container services on dedicated servers (typically referred to as the Swift proxy servers). If you are using this model, we have included the recommended Swift proxy servers specs in the table below.

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
Dedicated Cloud Lifecycle Manager (optional)	Lifecycle-manager	1	300 GB	8 GB	1 x 10 Gbit/s with PXE Support	8 CPU (64-bit) cores total (Intel x86_64)

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
Control Plane	Controller	3	• 1 x 600 GB (minimum) - operating system drive • 2 x 600 GB (minimum) - Swift account/container data drive • Fast disks or SSDs are rec-	128 GB	2 x 10 Gbit/s with one PXE enabled port	8 CPU (64-bit) cores total (Intel x86_64)

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
			ommended.			
Swift Object	swobj	3	If using x3 replication only: - 1 x 600 GB (minimum, see considerations at bottom of page for more details) If using Erasure Codes only or a mix of x3 repli-	32 GB (see considerations at bottom of page for more details)	2 x 10 Gbit/s with one PXE enabled port	8 CPU (64-bit) cores total (Intel x86_64)

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
			cation and Erasure Codes: - 6 x 600 GB (minimum, see considerations at bottom of page for more details)			
Swift Proxy, Account, and Container	swpac	3	2 x 600 GB (minimum, see considerations at bottom of page for more details)	64 GB (see considerations at bottom of page for more details)	2 x 10 Gbit/s with one PXE enabled port	8 CPU (64-bit) cores total (Intel x86_64)

Note

The disk speeds (RPM) chosen should be consistent within the same ring or storage policy. It is best to not use disks with mixed disk speeds within the same Swift ring.

Considerations for your Swift object and proxy, account, container servers RAM and disk capacity needs

Swift can have a diverse number of hardware configurations. For example, a Swift object server may have just a few disks (minimum of 6 for erasure codes) or up to 70 and beyond. The memory requirement needs to be increased as more disks are added. The general rule of thumb for memory needed is 0.5 GB per TB of storage. For example, a system with 24 hard drives at 8TB each, giving a total capacity of 192TB, should use 96GB of RAM. However, this does not work well for a system with a small number of small hard drives or a very large number of very large drives. So, if after calculating the memory given this guideline, if the answer is less than 32GB then go with 32GB of memory minimum and if the answer is over 256GB then use 256GB maximum, no need to use more memory than that.

When considering the capacity needs for the Swift proxy, account, and container (PAC) servers, you should calculate 2% of the total raw storage size of your object servers to specify the storage required for the PAC servers. So, for example, if you were using the example we provided earlier and you had an object server setup of 24 hard drives with 8TB each for a total of 192TB and you had a total of 6 object servers, that would give a raw total of 1152TB. So you would take 2% of that, which is 23TB, and ensure that much storage capacity was available on your Swift proxy, account, and container (PAC) server cluster. If you had a cluster of three Swift PAC servers, that would be ~8TB each.

Another general rule of thumb is that if you are expecting to have more than a million objects in a container then you should consider using SSDs on the Swift PAC servers rather than HDDs.

4 High Availability

This chapter covers High Availability concepts overview and cloud infrastructure.

4.1 High Availability Concepts Overview

A highly available (HA) cloud ensures that a minimum level of cloud resources are always available on request, which results in uninterrupted operations for users.

In order to achieve this high availability of infrastructure and workloads, we define the scope of HA to be limited to protecting these only against single points of failure (SPOF). Single points of failure include:

- **Hardware SPOFs**: Hardware failures can take the form of server failures, memory going bad, power failures, hypervisors crashing, hard disks dying, NIC cards breaking, switch ports failing, network cables loosening, and so forth.
- **Software SPOFs**: Server processes can crash due to software defects, out-of-memory conditions, operating system kernel panic, and so forth.

By design, SUSE OpenStack Cloud strives to create a system architecture resilient to SPOFs, and does not attempt to automatically protect the system against multiple cascading levels of failures; such cascading failures will result in an unpredictable state. The cloud operator is encouraged to recover and restore any failed component as soon as the first level of failure occurs.

4.2 Highly Available Cloud Infrastructure

The highly available cloud infrastructure consists of the following:

- High Availability of Controllers
- Availability Zones
- Compute with KVM
- Nova Availability Zones
- Compute with ESX
- Object Storage with Swift

4.3 High Availability of Controllers

The SUSE OpenStack Cloud installer deploys highly available configurations of OpenStack cloud services, resilient against single points of failure.

The high availability of the controller components comes in two main forms.

- Many services are stateless and multiple instances are run across the control plane in active-active mode. The API services (nova-api, cinder-api, etc.) are accessed through the HA proxy load balancer whereas the internal services (nova-scheduler, cinder-scheduler, etc.), are accessed through the message broker. These services use the database cluster to persist any data.

  ### Note

  The HA proxy load balancer is also run in active-active mode and keepalived (used for Virtual IP (VIP) Management) is run in active-active mode, with only one keepalived instance holding the VIP at any one point in time.

- The high availability of the message queue service and the database service is achieved by running these in a clustered mode across the three nodes of the control plane: RabbitMQ cluster with Mirrored Queues and MariaDB Galera cluster.

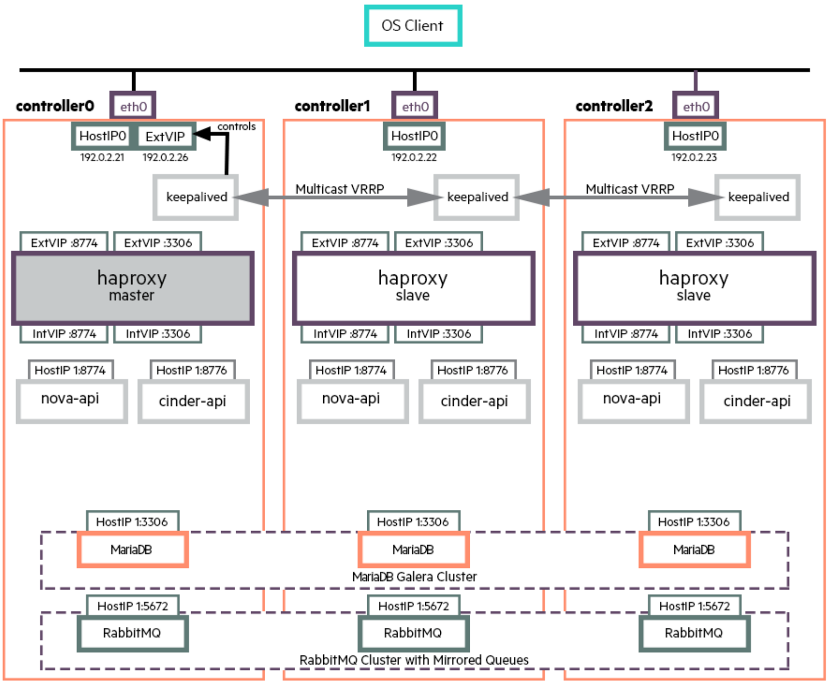


FIGURE 4.1: HA ARCHITECTURE

The above diagram illustrates the HA architecture with the focus on VIP management and load balancing. It only shows a subset of active-active API instances and does not show examples of other services such as nova-scheduler, cinder-scheduler, etc.

In the above diagram, requests from an OpenStack client to the API services are sent to VIP and port combination; for example, 192.0.2.26:8774 for a Nova request. The load balancer listens for requests on that VIP and port. When it receives a request, it selects one of the controller nodes configured for handling Nova requests, in this particular case, and then forwards the request to the IP of the selected controller node on the same port.

The nova-api service, which is listening for requests on the IP of its host machine, then receives the request and deals with it accordingly. The database service is also accessed through the load balancer. RabbitMQ, on the other hand, is not currently accessed through VIP/HA proxy as the clients are configured with the set of nodes in the RabbitMQ cluster and failover between cluster nodes is automatically handled by the clients.

4.4 High Availability Routing - Centralized

Incorporating High Availability into a system involves implementing redundancies in the component that is being made highly available. In Centralized Virtual Router (CVR), that element is the Layer 3 agent (L3 agent). By making L3 agent highly available, upon failure all HA routers are migrated from the primary L3 agent to a secondary L3 agent. The implementation efficiency of an HA subsystem is measured by the number of packets that are lost when the secondary L3 agent is made the master.

In SUSE OpenStack Cloud, the primary and secondary L3 agents run continuously, and failover involves a rapid switchover of mastership to the secondary agent (IEFT RFC 5798). The failover essentially involves a switchover from an already running master to an already running slave. This substantially reduces the latency of the HA. The mechanism used by the master and the slave to implement a failover is implemented using Linux's pacemaker HA resource manager. This CRM (Cluster resource manager) uses VRRP (Virtual Router Redundancy Protocol) to implement the HA mechanism. VRRP is a industry standard protocol and defined in RFC 5798.

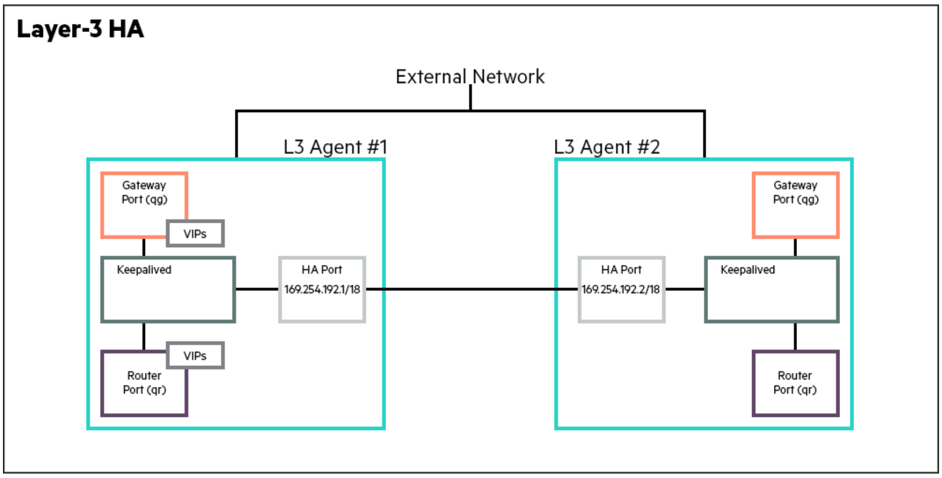


FIGURE 4.2: LAYER-3 HA

L3 HA uses of VRRP comes with several benefits.

The primary benefit is that the failover mechanism does not involve interprocess communication overhead. Such overhead would be in the order of 10s of seconds. By not using an RPC mechanism to invoke the secondary agent to assume the primary agents role enables VRRP to achieve failover within 1-2 seconds.

In VRRP, the primary and secondary routers are all active. As the routers are running, it is a matter of making the router aware of its primary/master status. This switchover takes less than 2 seconds instead of 60+ seconds it would have taken to start a backup router and failover.

The failover depends upon a heartbeat link between the primary and secondary. That link in SUSE OpenStack Cloud uses keepalived package of the pacemaker resource manager. The heartbeats are sent at a 2 second intervals between the primary and secondary. As per the VRRP protocol, if the secondary does not hear from the master after 3 intervals, it assumes the function of the primary.

Further, all the routable IP addresses, that is the VIPs (virtual IPs) are assigned to the primary agent.

4.5 High Availability Routing - Distributed

The OpenStack Distributed Virtual Router (DVR) function delivers HA through its distributed architecture. The one centralized function remaining is source network address translation (SNAT), where high availability is provided by DVR SNAT HA.

DVR SNAT HA is enabled on a per router basis and requires that two or more L3 agents capable of providing SNAT services be running on the system. If a minimum number of L3 agents is configured to 1 or lower, the neutron server will fail to start and a log message will be created. The L3 Agents must be running on a control-plane node, L3 agents running on a compute node do not provide SNAT services.

4.6 Availability Zones

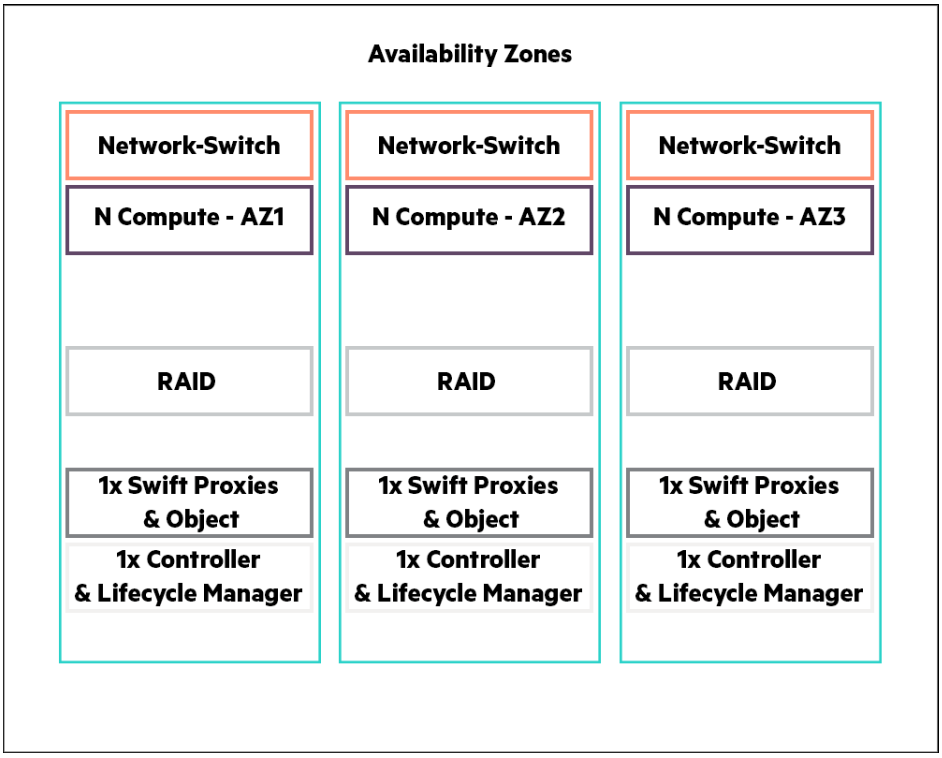

Availability Zones
Network-Switch
N Compute - AZ1
RAID
1x Swift Proxies & Object
1x Controller & Lifecycle Manager
Network-Switch
N Compute - AZ2
RAID
1x Swift Proxies & Object
1x Controller & Lifecycle Manager
Network-Switch
N Compute - AZ3
RAID
1x Swift Proxies & Object
1x Controller & Lifecycle Manager

FIGURE 4.3: AVAILABILITY ZONES

While planning your OpenStack deployment, you should decide on how to zone various types of nodes - such as compute, block storage, and object storage. For example, you may decide to place all servers in the same rack in the same zone. For larger deployments, you may plan more elaborate redundancy schemes for redundant power, network ISP connection, and even physical firewalling between zones (*this aspect is outside the scope of this document*).

SUSE OpenStack Cloud offers APIs, CLIs and Horizon UIs for the administrator to define and user to consume, availability zones for Nova, Cinder and Swift services. This section outlines the process to deploy specific types of nodes to specific physical servers, and makes a statement of available support for these types of availability zones in the current release.

Note

By default, SUSE OpenStack Cloud is deployed in a single availability zone upon installation. Multiple availability zones can be configured by an administrator post-install, if required. Refer to OpenStack Docs:Scaling your environment (https://docs.openstack.org/openstack-ansible/pike/admin/maintenance-tasks/scale-environment.html) ↗

4.7 Compute with KVM

You can deploy your KVM nova-compute nodes either during initial installation or by adding compute nodes post initial installation.

While adding compute nodes post initial installation, you can specify the target physical servers for deploying the compute nodes.

Learn more about adding compute nodes in *Book "Operations Guide", Chapter 13 "System Maintenance", Section 13.1 "Planned System Maintenance", Section 13.1.3 "Planned Compute Maintenance", Section 13.1.3.4 "Adding Compute Node"*.

4.8 Nova Availability Zones

Nova host aggregates and Nova availability zones can be used to segregate Nova compute nodes across different failure zones.

4.9 Compute with ESX Hypervisor

Compute nodes deployed on ESX Hypervisor can be made highly available using the HA feature of VMware ESX Clusters. For more information on VMware HA, please refer to your VMware ESX documentation.

4.10 Cinder Availability Zones

Cinder availability zones are not supported for general consumption in the current release.

4.11 Object Storage with Swift

High availability in Swift is achieved at two levels.

Control Plane

The Swift API is served by multiple Swift proxy nodes. Client requests are directed to all Swift proxy nodes by the HA Proxy load balancer in round-robin fashion. The HA Proxy load balancer regularly checks the node is responding, so that if it fails, traffic is directed to the remaining nodes. The Swift service will continue to operate and respond to client requests as long as at least one Swift proxy server is running.

If a Swift proxy node fails in the middle of a transaction, the transaction fails. However it is standard practice for Swift clients to retry operations. This is transparent to applications that use the python-swiftclient library.

The entry-scale example cloud models contain three Swift proxy nodes. However, it is possible to add additional clusters with additional Swift proxy nodes to handle a larger workload or to provide additional resiliency.

Data

Multiple replicas of all data is stored. This happens for account, container and object data. The example cloud models recommend a replica count of three. However, you may change this to a higher value if needed.

When Swift stores different replicas of the same item on disk, it ensures that as far as possible, each replica is stored in a different zone, server or drive. This means that if a single server of disk drives fails, there should be two copies of the item on other servers or disk drives.

If a disk drive is failed, Swift will continue to store three replicas. The replicas that would normally be stored on the failed drive are “handed off” to another drive on the system. When the failed drive is replaced, the data on that drive is reconstructed by the replication process. The replication process re-creates the “missing” replicas by copying them to the drive using one of the other remaining replicas. While this is happening, Swift can continue to store and retrieve data.

4.12 Highly Available Cloud Applications and Workloads

Projects writing applications to be deployed in the cloud must be aware of the cloud architecture and potential points of failure and architect their applications accordingly for high availability.

Some guidelines for consideration:

1. Assume intermittent failures and plan for retries

- **OpenStack Service APIs**: invocations can fail - you should carefully evaluate the response of each invocation, and retry in case of failures.
- **Compute**: VMs can die - monitor and restart them
- **Network**: Network calls can fail - retry should be successful
- **Storage**: Storage connection can hiccup - retry should be successful

2. Build redundancy into your application tiers

    - Replicate VMs containing stateless services such as Web application tier or Web service API tier and put them behind load balancers (you must implement your own HA Proxy type load balancer in your application VMs until SUSE OpenStack Cloud delivers the LBaaS service).
    - Boot the replicated VMs into different Nova availability zones.
    - If your VM stores state information on its local disk (Ephemeral Storage), and you cannot afford to lose it, then boot the VM off a Cinder volume.
    - Take periodic snapshots of the VM which will back it up to Swift through Glance.
    - Your data on ephemeral may get corrupted (but not your backup data in Swift and not your data on Cinder volumes).
    - Take regular snapshots of Cinder volumes and also back up Cinder volumes or your data exports into Swift.

3. Instead of rolling your own highly available stateful services, use readily available SUSE OpenStack Cloud platform services such as Designate, the DNS service.

4.13 What is not Highly Available?

Cloud Lifecycle Manager

The Cloud Lifecycle Manager in SUSE OpenStack Cloud is not highly available. The Cloud Lifecycle Manager state/data are all maintained in a filesystem and are backed up by the Freezer service. In case of Cloud Lifecycle Manager failure, the state/data can be recovered from the backup.

Control Plane

High availability (HA) is supported for the Network Services LBaaS and FWaaS. HA is **not** supported for VPNaaS.

Nova-consoleauth

Nova-consoleauth is a singleton service, it can only run on a single node at a time. While nova-consoleauth is not high availability, some work has been done to provide the ability to switch nova-consoleauth to another controller node in case of a failure.

Cinder Volume and Backup Services

Cinder Volume and Backup Services are not high availability and started on one controller node at a time. More information on Cinder Volume and Backup Services can be found in *Book “Operations Guide”, Chapter 7 “Managing Block Storage”, Section 7.1 “Managing Block Storage using Cinder”, Section 7.1.3 “Managing Cinder Volume and Backup Services”*.

Keystone Cron Jobs

The Keystone cron job is a singleton service, which can only run on a single node at a time. A manual setup process for this job will be required in case of a node failure. More information on enabling the cron job for Keystone on the other nodes can be found in *Book “Operations Guide”, Chapter 4 “Managing Identity”, Section 4.12 “Identity Service Notes and Limitations”, Section 4.12.4 “System cron jobs need setup”*.

4.14 More Information

- OpenStack High-availability Guide (https://docs.openstack.org/ha-guide/)
- 12-Factor Apps (http://12factor.net/)

II Cloud Lifecycle Manager Overview

5 Input Model

5.1 Introduction to the Input Model

This document describes how SUSE OpenStack Cloud input models can be used to define and configure the cloud.

SUSE OpenStack Cloud ships with a set of example input models that can be used as starting points for defining a custom cloud. An input model allows you, the cloud administrator, to describe the cloud configuration in terms of:

- Which OpenStack services run on which server nodes
- How individual servers are configured in terms of disk and network adapters
- The overall network configuration of the cloud
- Network traffic separation
- CIDR and VLAN assignments

The input model is consumed by the configuration processor which parses and validates the input model and outputs the effective configuration that will be deployed to each server that makes up your cloud.

The document is structured as follows:

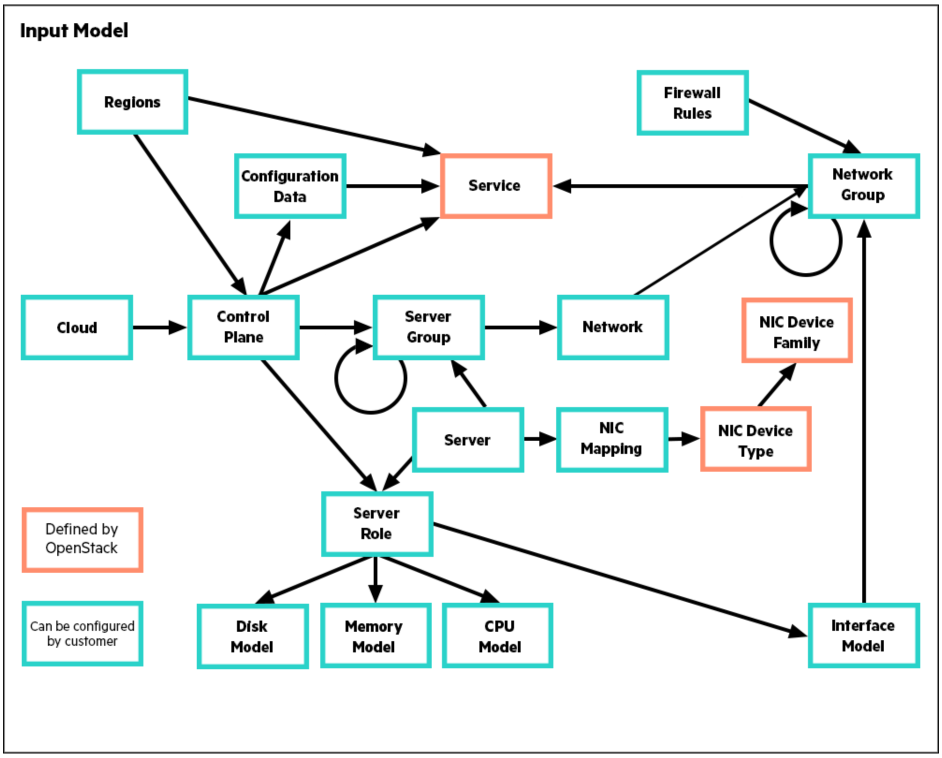
- *Concepts* - This explains the ideas behind the declarative model approach used in SUSE OpenStack Cloud 8 and the core concepts used in describing that model
- *Input Model* - This section provides a description of each of the configuration entities in the input model
- *Core Examples* - In this section we provide samples and definitions of some of the more important configuration entities

5.2 Concepts

An SUSE OpenStack Cloud 8 cloud is defined by a declarative model that is described in a series of configuration objects. These configuration objects are represented in YAML files which together constitute the various example configurations provided as templates with this release.

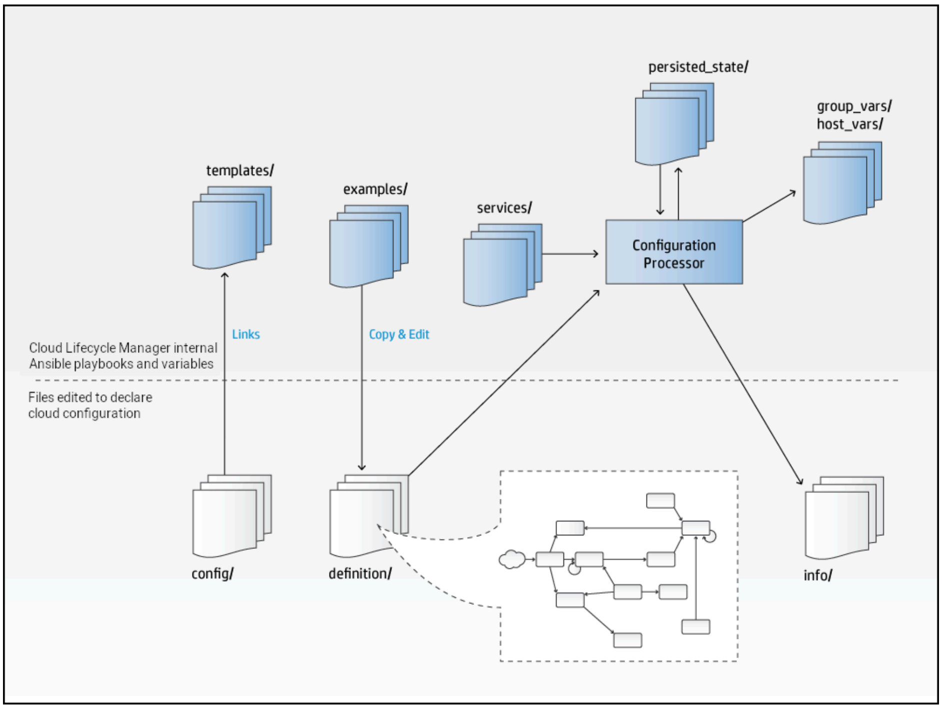
These examples can be used nearly unchanged, with the exception of necessary changes to IP addresses and other site and hardware-specific identifiers. Alternatively, the examples may be customized to meet site requirements.

The following diagram shows the set of configuration objects and their relationships. All objects have a name that you may set to be something meaningful for your context. In the examples these names are provided in capital letters as a convention. These names have no significance to SUSE OpenStack Cloud, rather it is the relationships between them that define the configuration.



The configuration processor reads and validates the input model described in the YAML files discussed above, combines it with the service definitions provided by SUSE OpenStack Cloud and any persisted state information about the current deployment to produce a set of Ansible variables that can be used to deploy the cloud. It also produces a set of information files that provide details about the configuration.

The relationship between the file systems on the SUSE OpenStack Cloud deployment server and the configuration processor is shown in the following diagram. Below the line are the directories that you, the cloud administrator, edit to declare the cloud configuration. Above the line are the directories that are internal to the Cloud Lifecycle Manager such as Ansible playbooks and variables.



The input model is read from the `~/openstack/my_cloud/definition` directory. Although the supplied examples use separate files for each type of object in the model, the names and layout of the files have no significance to the configuration processor, it simply reads all of the .yml files in this directory. Cloud administrators are therefore free to use whatever structure is best for their context. For example, you may decide to maintain separate files or sub-directories for each physical rack of servers.

As mentioned, the examples use the conventional upper casing for object names, but these strings are used only to define the relationship between objects. They have no specific significance to the configuration processor.

5.2.1 Cloud

The Cloud definition includes a few top-level configuration values such as the name of the cloud, the host prefix, details of external services (NTP, DNS, SMTP) and the firewall settings.

The location of the cloud configuration file also tells the configuration processor where to look for the files that define all of the other objects in the input model.

5.2.2 Control Planes

A control-plane runs one or more services distributed across clusters and resource groups.

A control-plane uses servers with a particular server-role.

A *control-plane* provides the operating environment for a set of *services*; normally consisting of a set of shared services (MariaDB, RabbitMQ, HA Proxy, Apache, etc.), OpenStack control services (API, schedulers, etc.) and the *resources* they are managing (compute, storage, etc.).

A simple cloud may have a single *control-plane* which runs all of the *services*. A more complex cloud may have multiple *control-planes* to allow for more than one instance of some services. Services that need to consume (use) another service (such as Neutron consuming MariaDB, Nova consuming Neutron) always use the service within the same *control-plane*. In addition a control-plane can describe which services can be consumed from other control-planes. It is one of the functions of the configuration processor to resolve these relationships and make sure that each consumer/service is provided with the configuration details to connect to the appropriate provider/service.

Each *control-plane* is structured as *clusters* and *resources*. The *clusters* are typically used to host the OpenStack services that manage the cloud such as API servers, database servers, Neutron agents, and Swift proxies, while the *resources* are used to host the scale-out OpenStack services such as Nova-Compute or Swift-Object services. This is a representation convenience rather than a strict rule, for example it is possible to run the Swift-Object service in the management cluster in a smaller-scale cloud that is not designed for scale-out object serving.

A cluster can contain one or more *servers* and you can have one or more *clusters* depending on the capacity and scalability needs of the cloud that you are building. Spreading services across multiple *clusters* provides greater scalability, but it requires a greater number of physical servers. A common pattern for a large cloud is to run high data volume services such as monitoring and logging in a separate cluster. A cloud with a high object storage requirement will typically also run the Swift service in its own cluster.

Clusters in this context are a mechanism for grouping service components in physical servers, but all instances of a component in a *control-plane* work collectively. For example, if HA Proxy is configured to run on multiple clusters within the same *control-plane* then all of those instances will work as a single instance of the ha-proxy service.

Both *clusters* and *resources* define the type (via a list of *server-roles*) and number of servers (min and max or count) they require.

The *control-plane* can also define a list of failure-zones (*server-groups*) from which to allocate servers.

5.2.2.1 Control Planes and Regions

A region in OpenStack terms is a collection of URLs that together provide a consistent set of services (Nova, Neutron, Swift, etc). Regions are represented in the Keystone identity service catalog. In SUSE OpenStack Cloud, multiple regions are not supported. Only `Region0` is valid.

In a simple single control-plane cloud, there is no need for a separate region definition and the control-plane itself can define the region name.

5.2.3 Services

A control-plane runs one or more services.

A service is the collection of *service-components* that provide a particular feature; for example, Nova provides the compute service and consists of the following service-components: nova-api, nova-scheduler, nova-conductor, nova-novncproxy, and nova-compute. Some services, like the authentication/identity service Keystone, only consist of a single service-component.

To define your cloud, all you need to know about a service are the names of the *service-components*. The details of the services themselves and how they interact with each other is captured in service definition files provided by SUSE OpenStack Cloud.

When specifying your SUSE OpenStack Cloud cloud you have to decide where components will run and how they connect to the networks. For example, should they all run in one *control-plane* sharing common services or be distributed across multiple *control-planes* to provide separate instances of some services? The SUSE OpenStack Cloud supplied examples provide solutions for some typical configurations.

Where services run is defined in the *control-plane*. How they connect to networks is defined in the *network-groups*.

5.2.4 Server Roles

Clusters and resources use servers with a particular set of server-roles.

You are going to be running the services on physical *servers*, and you are going to need a way to specify which type of servers you want to use where. This is defined via the *server-role*. Each *server-role* describes how to configure the physical aspects of a server to fulfill the needs of a particular role. You will generally use a different role whenever the servers are physically different (have different disks or network interfaces) or if you want to use some specific servers in a particular role (for example to choose which of a set of identical servers are to be used in the control plane).

Each *server-role* has a relationship to four other entities:

- The *disk-model* specifies how to configure and use a server's local storage and it specifies disk sizing information for virtual machine servers. The disk model is described in the next section.
- The *interface-model* describes how a server's network interfaces are to be configured and used. This is covered in more details in the networking section.
- An optional *memory-model* specifies how to configure and use huge pages. The memory-model specifies memory sizing information for virtual machine servers.
- An optional ***cpu-model*** specifies how the CPUs will be used by Nova and by DPDK. The cpu-model specifies CPU sizing information for virtual machine servers.

5.2.5 Disk Model

Each physical disk device is associated with a device-group or a volume-group.

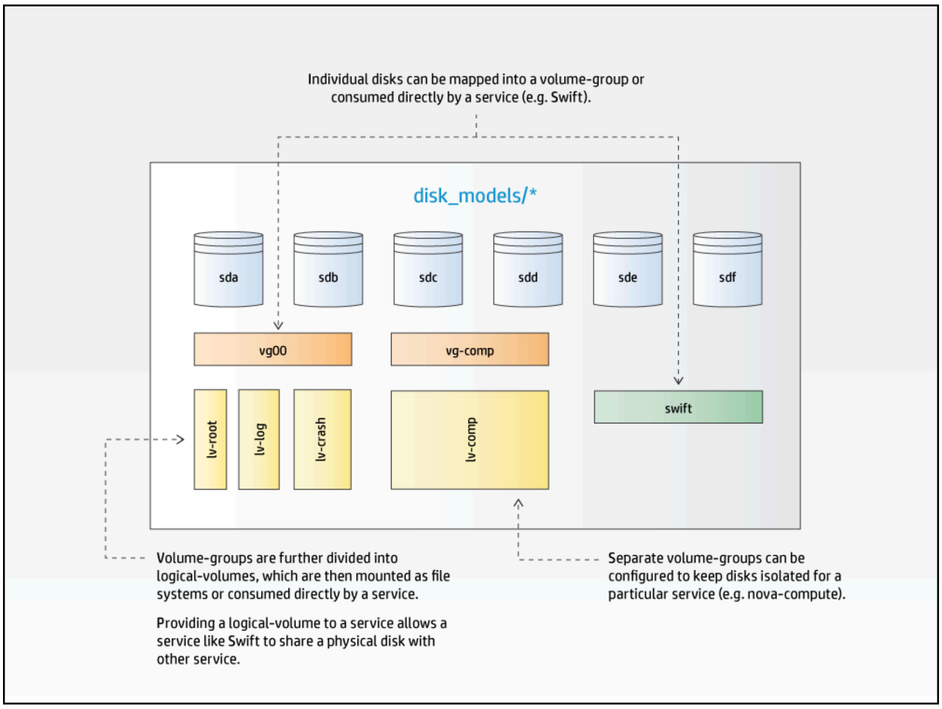
Device-groups are consumed by services.

Volume-groups are divided into logical-volumes.

Logical-volumes are mounted as file systems or consumed by services.

Disk-models define how local storage is to be configured and presented to *services*. Disk-models are identified by a name, which you will specify. The SUSE OpenStack Cloud examples provide some typical configurations. As this is an area that varies with respect to the services that are hosted on a server and the number of disks available, it is impossible to cover all possible permutations you may need to express via modifications to the examples.

Within a *disk-model*, disk devices are assigned to either a *device-group* or a *volume-group*.


Individual disks can be mapped into a volume-group or consumed directly by a service (e.g. Swift).
disk_models/*
sda
sdb
sdc
sdd
sde
sdf
vg00
vg-comp
swift
lv-root
lv-log
lv-crash
lv-comp
Volume-groups are further divided into logical-volumes, which are then mounted as file systems or consumed directly by a service.
Providing a logical-volume to a service allows a service like Swift to share a physical disk with other service.
Separate volume-groups can be configured to keep disks isolated for a particular service (e.g. nova-compute).

A *device-group* is a set of one or more disks that are to be consumed directly by a service. For example, a set of disks to be used by Swift. The device-group identifies the list of disk devices, the service, and a few service-specific attributes that tell the service about the intended use (for example, in the case of Swift this is the ring names). When a device is assigned to a device-group, the associated service is responsible for the management of the disks. This management includes the creation and mounting of file systems. (Swift can provide additional data integrity when it has full control over the file systems and mount points.)

A *volume-group* is used to present disk devices in a LVM volume group. It also contains details of the logical volumes to be created including the file system type and mount point. Logical volume sizes are expressed as a percentage of the total capacity of the volume group. A *logical-volume* can also be consumed by a service in the same way as a *device-group*. This allows services to manage their own devices on configurations that have limited numbers of disk drives.

Disk models also provide disk sizing information for virtual machine servers.

5.2.6 Memory Model

Memory models define how the memory of a server should be configured to meet the needs of a particular role. It allows a number of HugePages to be defined at both the server and numa-node level.

Memory models also provide memory sizing information for virtual machine servers.

Memory models are optional - it is valid to have a server role without a memory model.

5.2.7 CPU Model

CPU models define how CPUs of a server will be used. The model allows CPUs to be assigned for use by components such as Nova (for VMs) and Open vSwitch (for DPDK). It also allows those CPUs to be isolated from the general kernel SMP balancing and scheduling algorithms.

CPU models also provide CPU sizing information for virtual machine servers.

CPU models are optional - it is valid to have a server role without a cpu model.

5.2.8 Servers

Servers have a server-role which determines how they will be used in the cloud.

Servers (in the input model) enumerate the resources available for your cloud. In addition, in this definition file you can either provide SUSE OpenStack Cloud with all of the details it needs to PXE boot and install an operating system onto the server, or, if you prefer to use your own operating system installation tooling you can simply provide the details needed to be able to SSH into the servers and start the deployment.

The address specified for the server will be the one used by SUSE OpenStack Cloud for lifecycle management and must be part of a network which is in the input model. If you are using SUSE OpenStack Cloud to install the operating system this network must be an untagged VLAN. The first server must be installed manually from the SUSE OpenStack Cloud ISO and this server must be included in the input model as well.

In addition to the network details used to install or connect to the server, each server defines what its *server-role* is and to which *server-group* it belongs.

5.2.9 Server Groups

A server is associated with a server-group.

A control-plane can use server-groups as failure zones for server allocation.

A server-group may be associated with a list of networks.

A server-group can contain other server-groups.

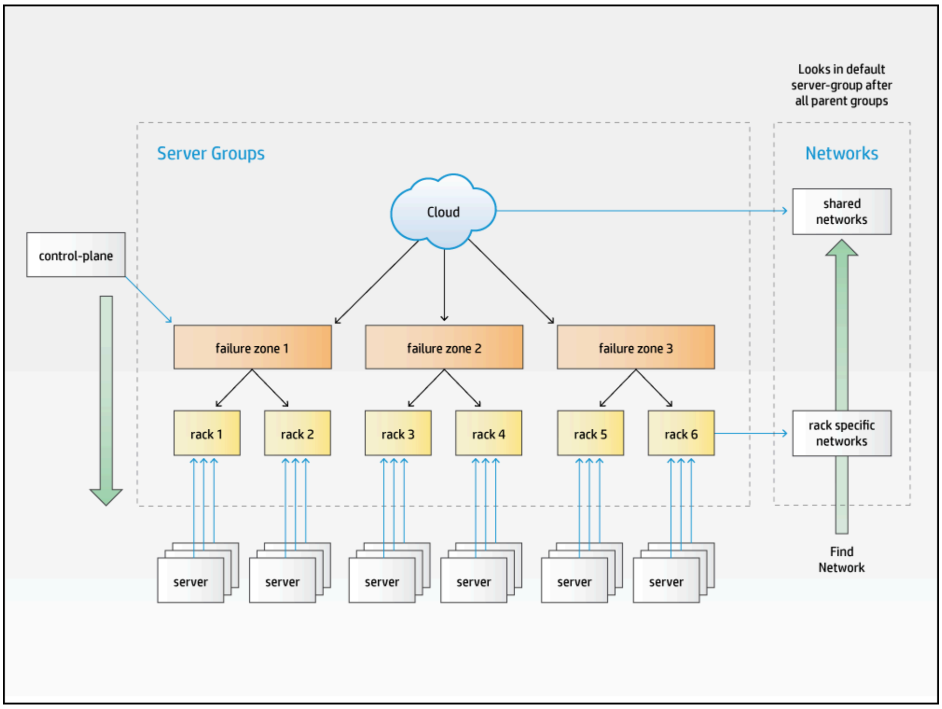
The practice of locating physical servers in a number of racks or enclosures in a data center is common. Such racks generally provide a degree of physical isolation that allows for separate power and/or network connectivity.

In the SUSE OpenStack Cloud model we support this configuration by allowing you to define a hierarchy of *server-groups*. Each *server* is associated with one *server-group*, normally at the bottom of the hierarchy.

Server-groups are an optional part of the input model - if you do not define any, then all *servers* and *networks* will be allocated as if they are part of the same *server-group*.

5.2.9.1 Server Groups and Failure Zones

A *control-plane* defines a list of *server-groups* as the failure zones from which it wants to use servers. All servers in a *server-group* listed as a failure zone in the *control-plane* and any *server-groups* they contain are considered part of that failure zone for allocation purposes. The following example shows how three levels of *server-groups* can be used to model a failure zone consisting of multiple racks, each of which in turn contains a number of *servers*.



When allocating *servers*, the configuration processor will traverse down the hierarchy of *server-groups* listed as failure zones until it can find an available server with the required *server-role*. If the allocation policy is defined to be strict, it will allocate *servers* equally across each of the failure zones. A *cluster* or *resource-group* can also independently specify the failure zones it wants to use if needed.

5.2.9.2 Server Groups and Networks

Each L3 *network* in a cloud must be associated with all or some of the *servers*, typically following a physical pattern (such as having separate networks for each rack or set of racks). This is also represented in the SUSE OpenStack Cloud model via *server-groups*, each group lists zero or more networks to which *servers* associated with *server-groups* at or below this point in the hierarchy are connected.

When the configuration processor needs to resolve the specific *network* a *server* should be configured to use, it traverses up the hierarchy of *server-groups*, starting with the group the server is directly associated with, until it finds a server-group that lists a network in the required network group.

The level in the *server-group* hierarchy at which a *network* is associated will depend on the span of connectivity it must provide. In the above example there might be networks in some *network-groups* which are per rack (that is Rack 1 and Rack 2 list different networks from the same *network-group*) and *networks* in a different *network-group* that span failure zones (the network used to provide floating IP addresses to virtual machines for example).

5.2.10 Networking

In addition to the mapping of *services* to specific *clusters* and *resources* we must also be able to define how the *services* connect to one or more *networks*.

In a simple cloud there may be a single L3 network but more typically there are functional and physical layers of network separation that need to be expressed.

Functional network separation provides different networks for different types of traffic; for example, it is common practice in even small clouds to separate the External APIs that users will use to access the cloud and the external IP addresses that users will use to access their virtual machines. In more complex clouds it is common to also separate out virtual networking between virtual machines, block storage traffic, and volume traffic onto their own sets of networks. In the input model, this level of separation is represented by *network-groups*.

Physical separation is required when there are separate L3 network segments providing the same type of traffic; for example, where each rack uses a different subnet. This level of separation is represented in the input model by the *networks* within each *network-group*.

5.2.10.1 Network Groups

Service endpoints attach to networks in a specific network-group.

Network-groups can define routes to other networks.

Network-groups encapsulate the configuration for services via network-tags

A *network-group* defines the traffic separation model and all of the properties that are common to the set of L3 networks that carry each type of traffic. They define where services are attached to the network model and the routing within that model.

In terms of *service* connectivity, all that has to be captured in the *network-groups* definition are the same service-component names that are used when defining *control-planes*. SUSE OpenStack Cloud also allows a default attachment to be used to specify "all service-components" that are not explicitly connected to another *network-group*. So, for example, to isolate Swift traffic, the swift-account, swift-container, and swift-object service components are attached to an "Object" *network-group* and all other services are connected to "MANAGEMENT" *network-group* via the default relationship.

Note

The name of the "MANAGEMENT" *network-group* cannot be changed. It must be upper case. Every SUSE OpenStack Cloud requires this network group in order to be valid.

The details of how each service connects, such as what port it uses, if it should be behind a load balancer, if and how it should be registered in Keystone, and so forth, are defined in the service definition files provided by SUSE OpenStack Cloud.

In any configuration with multiple networks, controlling the routing is a major consideration. In SUSE OpenStack Cloud, routing is controlled at the *network-group* level. First, all *networks* are configured to provide the route to any other *networks* in the same *network-group*. In addition, a *network-group* may be configured to provide the route any other *networks* in the same *network-group*; for example, if the internal APIs are in a dedicated *network-group* (a common configuration in a complex network because a network group with load balancers cannot be segmented) then other *network-groups* may need to include a route to the internal API *network-group* so that services can access the internal API endpoints. Routes may also be required to define how to access an external storage network or to define a general default route.

As part of the SUSE OpenStack Cloud deployment, networks are configured to act as the default route for all traffic that was received via that network (so that response packets always return via the network the request came from).

Note that SUSE OpenStack Cloud will configure the routing rules on the servers it deploys and will validate that the routes between services exist in the model, but ensuring that gateways can provide the required routes is the responsibility of your network configuration. The configuration processor provides information about the routes it is expecting to be configured.

For a detailed description of how the configuration processor validates routes, refer to *Section 7.6, "Network Route Validation"*.

5.2.10.1.1 Load Balancers

Load-balancers provide a specific type of routing and are defined as a relationship between the virtual IP address (VIP) on a network in one *network group* and a set of service endpoints (which may be on *networks* in the same or a different *network-group*).

As each *load-balancer* is defined providing a virtual IP on a *network-group*, it follows that those *network-groups* can each only have one *network* associated to them.

The *load-balancer* definition includes a list of *service-components* and endpoint roles it will provide a virtual IP for. This model allows service-specific *load-balancers* to be defined on different *network-groups*. A "default" value is used to express "all service-components" which require a virtual IP address and are not explicitly configured in another *load-balancer* configuration. The details of how the *load-balancer* should be configured for each service, such as which ports to use, how to check for service liveness, etc., are provided in the SUSE OpenStack Cloud supplied service definition files.

Where there are multiple instances of a service (for example, in a cloud with multiple control-planes), each control-plane needs its own set of virtual IP address and different values for some properties such as the external name and security certificate. To accommodate this in SUSE OpenStack Cloud 8, load-balancers are defined as part of the control-plane, with the network groups defining just which load-balancers are attached to them.

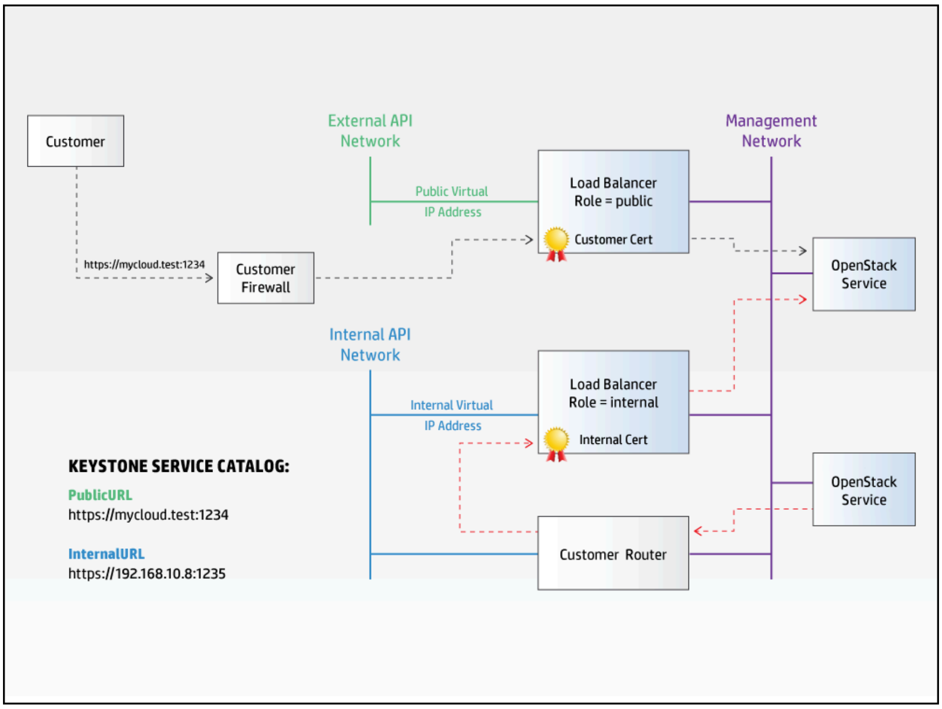
Load balancers are always implemented by an ha-proxy service in the same control-plane as the services.

5.2.10.1.2 Separation of Public, Admin, and Internal Endpoints

The list of endpoint roles for a *load-balancer* make it possible to configure separate *load-balancers* for public and internal access to services, and the configuration processor uses this information to both ensure the correct registrations in Keystone and to make sure the internal traffic is routed to the correct endpoint. SUSE OpenStack Cloud services are configured to only connect to other services via internal virtual IP addresses and endpoints, allowing the name and security certificate of public endpoints to be controlled by the customer and set to values that may not be resolvable/accessible from the servers making up the cloud.

Note that each *load-balancer* defined in the input model will be allocated a separate virtual IP address even when the load-balancers are part of the same *network-group*. Because of the need to be able to separate both public and internal access, SUSE OpenStack Cloud will not allow a single *load-balancer* to provide both public and internal access. *Load-balancers* in this context are logical entities (sets of rules to transfer traffic from a virtual IP address to one or more endpoints).

The following diagram shows a possible configuration in which the hostname associated with the public URL has been configured to resolve to a firewall controlling external access to the cloud. Within the cloud, SUSE OpenStack Cloud services are configured to use the internal URL to access a separate virtual IP address.


Customer
External API Network
Management Network
Public Virtual IP Address
Load Balancer Role = public
Customer Cert
https://mycloud.test:1234
Customer Firewall
OpenStack Service
Internal API Network
Internal Virtual IP Address
Load Balancer Role = internal
Internal Cert
OpenStack Service
Customer Router
KEYSTONE SERVICE CATALOG:
PublicURL
https://mycloud.test:1234
InternalURL
https://192.168.10.8:1235

5.2.10.1.3 Network Tags

Network tags are defined by some SUSE OpenStack Cloud *service-components* and are used to convey information between the network model and the service, allowing the dependent aspects of the service to be automatically configured.

Network tags also convey requirements a service may have for aspects of the server network configuration, for example, that a bridge is required on the corresponding network device on a server where that service-component is installed.

See *Section 6.13.2, “Network Tags”* for more information on specific tags and their usage.

5.2.10.2 Networks

A network is part of a network-group.

Networks are fairly simple definitions. Each *network* defines the details of its VLAN, optional address details (CIDR, start and end address, gateway address), and which *network-group* it is a member of.

5.2.10.3 Interface Model

A server-role identifies an interface-model that describes how its network interfaces are to be configured and used.

Network groups are mapped onto specific network interfaces via an *interface-model*, which describes the network devices that need to be created (bonds, ovs-bridges, etc.) and their properties.

An *interface-model* acts like a template; it can define how some or all of the *network-groups* are to be mapped for a particular combination of physical NICs. However, it is the *service-components* on each server that determine which *network-groups* are required and hence which interfaces and *networks* will be configured. This means that *interface-models* can be shared between different *server-roles*. For example, an API role and a database role may share an interface model even though they may have different disk models and they will require a different subset of the *network-groups*.

Within an *interface-model*, physical ports are identified by a device name, which in turn is resolved to a physical port on a server basis via a *nic-mapping*. To allow different physical servers to share an *interface-model*, the *nic-mapping* is defined as a property of each *server*.

The `interface-model` can also used to describe how network devices are to be configured for use with DPDK, SR-IOV, and PCI Passthrough.

5.2.10.4 NIC Mapping

When a *server* has more than a single physical network port, a *nic-mapping* is required to unambiguously identify each port. Standard Linux mapping of ports to interface names at the time of initial discovery (for example, `eth0`, `eth1`, `eth2`, ...) is not uniformly consistent from server to server, so a mapping of PCI bus address to interface name is instead.

NIC mappings are also used to specify the device type for interfaces that are to be used for SR-IOV or PCI Passthrough. Each SUSE OpenStack Cloud release includes the data for the supported device types.

5.2.10.5 Firewall Configuration

The configuration processor uses the details it has about which networks and ports *service-components* use to create a set of firewall rules for each server. The model allows additional user-defined rules on a per *network-group* basis.

5.2.11 Configuration Data

Configuration Data is used to provide settings which have to be applied in a specific context, or where the data needs to be verified against or merged with other values in the input model.

For example, when defining a Neutron provider network to be used by Octavia, the network needs to be included in the routing configuration generated by the Configuration Processor.

6 Configuration Objects

6.1 Cloud Configuration

The top-level cloud configuration file, `cloudConfig.yml`, defines some global values for SUSE OpenStack Cloud, as described in the table below.

The snippet below shows the start of the control plane definition file.

```
---
  product:
    version: 2

  cloud:
    name: entry-scale-kvm

    hostname-data:
        host-prefix: ardana
        member-prefix: -m

    ntp-servers:
        - "ntp-server1"

    # dns resolving configuration for your site
    dns-settings:
      nameservers:
        - name-server1

    firewall-settings:
        enable: true
        # log dropped packets
        logging: true

    audit-settings:
       audit-dir: /var/audit
       default: disabled
       enabled-services:
         - keystone
```

Key	Value Description
name	An administrator-defined name for the cloud

Key	Value Description
hostname-data (optional)	Provides control over some parts of the generated names (see) Consists of two values: • host-prefix - default is to use the cloud name (above) • member-prefix - default is "-m"
ntp-servers (optional)	A list of external NTP servers your cloud has access to. If specified by name then the names need to be resolvable via the external DNS nameservers you specify in the next section. All servers running the "ntp-server" component will be configured to use these external NTP servers.
dns-settings (optional)	DNS configuration data that will be applied to all servers. See example configuration for a full list of values.
smtp-settings (optional)	SMTP client configuration data that will be applied to all servers. See example configurations for a full list of values.
firewall-settings (optional)	Used to enable/disable the firewall feature and to enable/disable logging of dropped packets. The default is to have the firewall enabled.
audit-settings (optional)	Used to enable/disable the production of audit data from services. The default is to have audit disabled for all services.

6.2 Control Plane

The snippet below shows the start of the control plane definition file.

```
---
  product:
     version: 2

  control-planes:
     - name: control-plane-1
       control-plane-prefix: cp1
       region-name: region0
       failure-zones:
         - AZ1
         - AZ2
         - AZ3
       configuration-data:
         - NEUTRON-CONFIG-CP1
         - OCTAVIA-CONFIG-CP1
       common-service-components:
         - logging-producer
         - monasca-agent
         - freezer-agent
         - stunnel
         - lifecycle-manager-target
       clusters:
         - name: cluster1
           cluster-prefix: c1
           server-role: CONTROLLER-ROLE
           member-count: 3
           allocation-policy: strict
           service-components:
             - lifecycle-manager
             - ntp-server
             - swift-ring-builder
             - mysql
             - ip-cluster
             ...

       resources:
         - name: compute
           resource-prefix: comp
           server-role: COMPUTE-ROLE
           allocation-policy: any
           min-count: 0
           service-components:
              - ntp-client
              - nova-compute
              - nova-compute-kvm
              - neutron-l3-agent
              ...
```

Key	Value Description
name	This name identifies the control plane. This value is used to persist server allocations *Section 7.3, “Persisted Data”* and cannot be changed once servers have been allocated.

Key	Value Description
control-plane-prefix (optional)	The control-plane-prefix is used as part of the hostname (see *Section 7.2, "Name Generation"*). If not specified, the control plane name is used.
region-name	This name identifies the Keystone region within which services in the control plane will be registered. In SUSE OpenStack Cloud, multiple regions are not supported. Only `Region0` is valid. For clouds consisting of multiple control planes, this attribute should be omitted and the regions object should be used to set the region name (`Region0`).
uses (optional)	Identifies the services this control will consume from other control planes (see *Section 6.2.3, "Multiple Control Planes"*).
load-balancers (optional)	A list of load balancer definitions for this control plane (see *Section 6.2.4, "Load Balancer Definitions in Control Planes"*). For a multi control-plane cloud load balancers must be defined in each control-plane. For a single control-plane cloud they may be defined either in the control plane or as part of a network group.
common-service-components (optional)	This lists a set of service components that run on all servers in the control plane (clusters and resource pools).
failure-zones (optional)	A list of *server-group* names that servers for this control plane will be allocated from. If no failure-zones are specified, only servers

Key	Value Description
	not associated with a *server-group* will be used. (See *Section 5.2.9.1, "Server Groups and Failure Zones"* for a description of server-groups as failure zones.)
configuration-data (optional)	A list of configuration data settings to be used for services in this control plane (see *Section 5.2.11, "Configuration Data"*).
clusters	A list of clusters for this control plane (see *Section 6.2.1, " Clusters"*).
resources	A list of resource groups for this control plane (see *Section 6.2.2, "Resources"*).

6.2.1 Clusters

Key	Value Description
name	Cluster and resource names must be unique within a control plane. This value is used to persist server allocations (see *Section 7.3, "Persisted Data"*) and cannot be changed once servers have been allocated.
cluster-prefix (optional)	The cluster prefix is used in the hostname (see *Section 7.2, "Name Generation"*). If not supplied then the cluster name is used.
server-role	This can either be a string (for a single role) or a list of roles. Only servers matching one of the specified *server-roles* will be allocated to this cluster. (see *Section 5.2.4, "Server Roles"* for a description of server roles)

Key	Value Description
service-components	The list of *service-components* to be deployed on the servers allocated for the cluster. (The common-service-components for the control plane are also deployed.)
member-count min-count max-count (all optional)	Defines the number of servers to add to the cluster. The number of servers that can be supported in a cluster depends on the services it is running. For example MariaDB and RabbitMQ can only be deployed on clusters on 1 (non-HA) or 3 (HA) servers. Other services may support different sizes of cluster. If min-count is specified, then at least that number of servers will be allocated to the cluster. If min-count is not specified it defaults to a value of 1. If max-count is specified, then the cluster will be limited to that number of servers. If max-count is not specified then all servers matching the required role and failure-zones will be allocated to the cluster. Specifying member-count is equivalent to specifying min-count and max-count with the same value.
failure-zones (optional)	A list of *server-groups* that servers will be allocated from. If specified, it overrides the list of values specified for the control-plane. If not specified, the control-plane value is used. (see *Section 5.2.9.1, "Server Groups and Failure Zones"* for a description of server groups as failure zones).

Key	Value Description
allocation-policy (optional)	Defines how failure zones will be used when allocating servers. **strict**: Server allocations will be distributed across all specified failure zones. (if max-count is not a whole number, an exact multiple of the number of zones, then some zones may provide one more server than other zones) **any**: Server allocations will be made from any combination of failure zones. The default allocation-policy for a cluster is *strict*.
configuration-data (optional)	A list of configuration-data settings that will be applied to the services in this cluster. The values for each service will be combined with any values defined as part of the configuration-data list for the control-plane. If a value is specified by settings in both lists, the value defined here takes precedence.

6.2.2 Resources

Key	Value Description
name	The name of this group of resources. Cluster names and resource-node names must be unique within a control plane. Additionally, clusters and resources cannot share names within a control-plane.

Key	Value Description
	This value is used to persist server allocations (see *Section 7.3, "Persisted Data"*) and cannot be changed once servers have been allocated.
resource-prefix	The resource-prefix is used in the name generation. (see *Section 7.2, "Name Generation"*)
server-role	This can either be a string (for a single role) or a list of roles. Only servers matching one of the specified *server-roles* will be allocated to this resource group. (see *Section 5.2.4, "Server Roles"* for a description of server roles).
service-components	The list of *service-components* to be deployed on the servers in this resource group. (The common-service-components for the control plane are also deployed.)
member-count min-count max-count (all optional)	Defines the number of servers to add to the cluster. The number of servers that can be supported in a cluster depends on the services it is running. For example MariaDB and RabbitMQ can only be deployed on clusters on 1 (non-HA) or 3 (HA) servers. Other services may support different sizes of cluster. If min-count is specified, then at least that number of servers will be allocated to the cluster. If min-count is not specified it defaults to a value of 1.

Key	Value Description
	If max-count is specified, then the cluster will be limited to that number of servers. If max-count is not specified then all servers matching the required role and failure-zones will be allocated to the cluster. Specifying member-count is equivalent to specifying min-count and max-count with the same value.
failure-zones (optional)	A list of *server-groups* that servers will be allocated from. If specified, it overrides the list of values specified for the control-plane. If not specified, the control-plane value is used. (see *Section 5.2.9.1, “Server Groups and Failure Zones”* for a description of server groups as failure zones).
allocation-policy (optional)	Defines how failure zones will be used when allocating servers. **strict**: Server allocations will be distributed across all specified failure zones. (if max-count is not a whole number, an exact multiple of the number of zones, then some zones may provide one more server than other zones) **any**: Server allocations will be made from any combination of failure zones. The default allocation-policy for resources is *any*.
configuration-data (optional)	A list of configuration-data settings that will be applied to the services in this cluster. The values for each service will be combined with any values defined as part of the configuration-data list for the control-plane. If a value

Key	Value Description
	is specified by settings in both lists, the value defined here takes precedence.

6.2.3 Multiple Control Planes

The dependencies between service components (for example, Nova needs MariaDB and Keystone API) is defined as part of the service definitions provide by SUSE OpenStack Cloud, the control-planes define how those dependencies will be met. For clouds consisting of multiple control-planes, the relationship between services in different control planes is defined by a `uses` attribute in its control-plane object. Services will always use other services in the same control-plane before looking to see if the required service can be provided from another control-plane. For example, a service component in control-plane `cp-2` (for example, nova-api) might use service components from control-plane `cp-shared` (for example, keystone-api).

```
control-planes:
    - name: cp-2
      uses:
        - from: cp-shared
          service-components:
            - any
```

Key	Value Description
from	The name of the control-plane providing services which may be consumed by this control-plane.
service-components	A list of service components from the specified control-plane which may be consumed by services in this control-plane. The reserved keyword `any` indicates that any service component from the specified control-plane may be consumed by services in this control-plane.

6.2.4 Load Balancer Definitions in Control Planes

Starting in SUSE OpenStack Cloud 8, a load-balancer may be defined within a control-plane object, and referenced by name from a network-groups object. The following example shows load balancer `extlb` defined in control-plane `cp1` and referenced from the EXTERNAL-API network group. See section Load balancers for a complete description of load balance attributes.

```
network-groups:
    - name: EXTERNAL-API
      load-balancers:
        - extlb

  control-planes:
    - name: cp1
      load-balancers:
        - provider: ip-cluster
          name: extlb
          external-name:
          tls-components:
            - default
          roles:
            - public
          cert-file: cp1-extlb-cert
```

6.3 Load Balancers

Load balancers may be defined as part of a network-group object, or as part of a control-plane object. When a load-balancer is defined in a control-plane, it must be referenced by name only from the associated network-group object.

For clouds consisting of multiple control planes, load balancers must be defined as part of a control-plane object. This allows different load balancer configurations for each control plane.

In either case, a load-balancer definition has the following attributes:

```
load-balancers:
        - provider: ip-cluster
          name: extlb
          external-name:

          tls-components:
            - default
          roles:
            - public
        cert-file: cp1-extlb-cert
```

Key	Value Description
name	An administrator defined name for the load balancer. This name is used to make the association from a network-group.
provider	The service component that implements the load balancer. Currently only `ip-cluster` (ha-proxy) is supported. Future releases will provide support for external load balancers.
roles	The list of endpoint roles that this load balancer provides (see below). Valid roles are `public`, `internal`, and `admin`. To ensure separation of concerns, the role `public` cannot be combined with any other role. See Load Balancers for an example of how the role provides endpoint separation.
components (optional)	The list of service-components for which the load balancer provides a non-encrypted virtual IP address for.
tls-components (optional)	The list of service-components for which the load balancer provides TLS-terminated virtual IP addresses for.
external-name (optional)	The name to be registered in Keystone for the publicURL. If not specified, the virtual IP address will be registered. Note that this value cannot be changed after the initial deployment.
cert-file (optional)	The name of the certificate file to be used for tls endpoints. If not specified, a file name will be constructed using the format *`CP-NAME`*`-`*`LB-NAME`*`-cert`, where *`CP-NAME`* is the con-

Key	Value Description
	trol-plane name and *LB-NAME* is the load-balancer name.

6.4 Regions

The regions configuration object is used to define how a set of services from one or more control-planes are mapped into Openstack regions (entries within the Keystone catalog). In SUSE OpenStack Cloud, multiple regions are not supported. Only `Region0` is valid.

Within each region a given service is provided by one control plane, but the set of services in the region may be provided by multiple control planes.

Key	Value Description
name	The name of the region in the Keystone service catalog.
includes	A list of services to include in this region, broken down by the control planes providing the services.

Key	Value Description
control-plane	A control-plane name.
services	A list of service names. This list specifies the services from this control-plane to be included in this region. The reserved keyword `all` may be used when all services from the control-plane are to be included.

6.5 Servers

The *servers* configuration object is used to list the available servers for deploying the cloud.

Optionally, it can be used as an input file to the operating system installation process, in which case some additional fields (identified below) will be necessary.

```
---
  product:
    version: 2

  baremetal:
    subnet: 192.168.10.0
    netmask: 255.255.255.0

  servers:
    - id: controller1
      ip-addr: 192.168.10.3
      role: CONTROLLER-ROLE
      server-group: RACK1
      nic-mapping: HP-DL360-4PORT
      mac-addr: b2:72:8d:ac:7c:6f
      ilo-ip: 192.168.9.3
      ilo-password: password
      ilo-user: admin

    - id: controller2
      ip-addr: 192.168.10.4
      role: CONTROLLER-ROLE
      server-group: RACK2
      nic-mapping: HP-DL360-4PORT
      mac-addr: 8a:8e:64:55:43:76
      ilo-ip: 192.168.9.4
      ilo-password: password
      ilo-user: admin
```

Key	Value Description
id	An administrator-defined identifier for the server. IDs must be unique and are used to track server allocations. (see *Section 7.3, "Persisted Data"*).
ip-addr	The IP address is used by the configuration processor to install and configure the service components on this server. This IP address must be within the range of a *network* defined in this model. When the servers file is being used for operating system installation, this IP address will be assigned to the node by the installation process, and the associated *network* must be an untagged VLAN.
hostname (optional)	The value to use for the hostname of the server. If specified this will be used to set the hostname value of the server which will in turn be reflected in systems

Key	Value Description
	such as Nova, Monasca, etc. If not specified the hostname will be derived based on where the server is used and the network defined to provide hostnames.
role	Identifies the *server-role* of the server.
nic-mapping	Name of the *nic-mappings* entry to apply to this server. (See *Section 6.12, "NIC Mappings"*.)
server-group (optional)	Identifies the *server-groups* entry that this server belongs to. (see *Section 5.2.9, "Server Groups"*)
boot-from-san (optional)	Must be set to true is the server needs to be configured to boot from SAN storage. Default is False
fcoe-interfaces (optional)	A list of network devices that will be used for accessing FCoE storage. This is only needed for devices that present as native FCoE, not devices such as Emulex which present as a FC device.
ansible-options (optional)	A string of additional variables to be set when defining the server as a host in Ansible. For example, `ansible_ssh_port=5986`
mac-addr (optional)	Needed when the servers file is being used for operating system installation. This identifies the MAC address on the server that will be used to network install the operating system.
kopt-extras (optional)	Provides additional command line arguments to be passed to the booting network kernel. For example, `vga=769` sets the video mode for the install to low resolution which can be useful for remote console users.
ilo-ip (optional)	Needed when the servers file is being used for operating system installation. This provides the IP address of the power management (for example, IPMI, iLO) subsystem.
ilo-user (optional)	Needed when the servers file is being used for operating system installation. This provides the user name of the power management (for example, IPMI, iLO) subsystem.

Key	Value Description
ilo-password (optional)	Needed when the servers file is being used for operating system installation. This provides the user password of the power management (for example, IPMI, iLO) subsystem.
ilo-extras (optional)	Needed when the servers file is being used for operating system installation. Additional options to pass to ipmitool. For example, this may be required if the servers require additional IPMI addressing parameters.
moonshot (optional)	Provides the node identifier for HPE Moonshot servers, for example, `c4n1` where c4 is the cartridge and n1 is node.
hypervisor-id (optional)	This attribute serves two purposes: it indicates that this server is a virtual machine (VM), and it specifies the server id of the Cloud Lifecycle Manager hypervisor that will host the VM.
ardana-hypervisor (optional)	When set to True, this attribute identifies a server as a Cloud Lifecycle Manager hypervisor. A Cloud Lifecycle Manager hypervisor is a server that may be used to host other servers that are themselves virtual machines. Default value is `False`.

6.6 Server Groups

The server-groups configuration object provides a mechanism for organizing servers and networks into a hierarchy that can be used for allocation and network resolution.

```
---
  product:
     version: 2

     - name: CLOUD
        server-groups:
         - AZ1
         - AZ2
         - AZ3
        networks:
         - EXTERNAL-API-NET
         - EXTERNAL-VM-NET
         - GUEST-NET
         - MANAGEMENT-NET
```

```
    #
    # Create a group for each failure zone
    #
    - name: AZ1
      server-groups:
        - RACK1

    - name: AZ2
      server-groups:
        - RACK2

    - name: AZ3
      server-groups:
        - RACK3

    #
    # Create a group for each rack
    #
    - name: RACK1
    - name: RACK2
    - name: RACK3
```

Key	Value Description
name	An administrator-defined name for the server group. The name is used to link server-groups together and to identify server-groups to be used as failure zones in a *control-plane*. (see *Section 6.2, “Control Plane”*)
server-groups (optional)	A list of server-group names that are nested below this group in the hierarchy. Each server group can only be listed in one other server group (that is in a strict tree topology).
networks (optional)	A list of network names (see *Section 5.2.10.2, “Networks”*). See *Section 5.2.9.2, “Server Groups and Networks”* for a description of how networks are matched to servers via server groups.

6.7 Server Roles

The server-roles configuration object is a list of the various server roles that you can use in your cloud. Each server role is linked to other configuration objects:

- Disk model (*Section 6.8, “ Disk Models”*)
- Interface model (*Section 6.11, “Interface Models”*)
- Memory model (*Section 6.9, “Memory Models”*)
- CPU model (*Section 6.10, “ CPU Models”*)

Server roles are referenced in the servers (see *Section 6.7, “Server Roles”*) configuration object above.

```
---
  product:
     version: 2

  server-roles:

     - name: CONTROLLER-ROLE
       interface-model: CONTROLLER-INTERFACES
       disk-model: CONTROLLER-DISKS

     - name: COMPUTE-ROLE
       interface-model: COMPUTE-INTERFACES
       disk-model: COMPUTE-DISKS
       memory-model: COMPUTE-MEMORY
       cpu-model: COMPUTE-CPU
```

Key	**Value Description**
name	An administrator-defined name for the role.
interface-model	The name of the *interface-model* to be used for this server-role. Different server-roles can use the same interface-model.
disk-model	The name of the *disk-model* to use for this server-role.

Key	Value Description
	Different server-roles can use the same disk-model.
memory-model (optional)	The name of the *memory-model* to use for this server-role. Different server-roles can use the same memory-model.
cpu-model (optional)	The name of the *cpu-model* to use for this server-role. Different server-roles can use the same cpu-model.

6.8 Disk Models

The disk-models configuration object is used to specify how the directly attached disks on the server should be configured. It can also identify which service or service component consumes the disk, for example, Swift object server, and provide service-specific information associated with the disk. It is also used to specify disk sizing information for virtual machine servers.

Disks can be used as raw devices or as logical volumes and the disk model provides a configuration item for each.

If the operating system has been installed by the SUSE OpenStack Cloud installation process then the root disk will already have been set up as a volume-group with a single logical-volume. This logical-volume will have been created on a partition identified, symbolically, in the configuration files as `/dev/sda_root`. This is due to the fact that different BIOS systems (UEFI, Legacy) will result in different partition numbers on the root disk.

```
---
  product:
     version: 2

  disk-models:
  - name: SES-DISKS

    volume-groups:
       - ...
    device-groups:
       - ...
    vm-size:
       ...
```

Key	Value Description
name	The name of the disk-model that is referenced from one or more server-roles.
volume-groups	A list of volume-groups to be configured (see below). There must be at least one volume-group describing the root file system.
device-groups (optional)	A list of device-groups (see below)

6.8.1 Volume Groups

The *volume-groups* configuration object is used to define volume groups and their constituent logical volumes.

Note that volume-groups are not exact analogs of device-groups. A volume-group specifies a set of physical volumes used to make up a volume-group that is then subdivided into multiple logical volumes.

The SUSE OpenStack Cloud operating system installation automatically creates a volume-group name "ardana-vg" on the first drive in the system. It creates a "root" logical volume there. The volume-group can be expanded by adding more physical-volumes (see examples). In addition, it is possible to create more logical-volumes on this volume-group to provide dedicated capacity for different services or file system mounts.

```
   volume-groups:
     - name: ardana-vg
       physical-volumes:
         - /dev/sda_root

       logical-volumes:
         - name: root
           size: 35%
           fstype: ext4
           mount: /
```

```
        - name: log
          size: 50%
          mount: /var/log
          fstype: ext4
          mkfs-opts: -O large_file

        - ...

    - name: vg-comp
      physical-volumes:
        - /dev/sdb
      logical-volumes:
        - name: compute
          size: 95%
          mount: /var/lib/nova
          fstype: ext4
          mkfs-opts: -O large_file
```

Key	Value Descriptions
name	The name that will be assigned to the volume-group
physical-volumes	A list of physical disks that make up the volume group. As installed by the SUSE OpenStack Cloud operating system install process, the volume group "ardana-vg" will use a large partition (sda_root) on the first disk. This can be expanded by adding additional disk(s).
logical-volumes	A list of logical volume devices to create from the above named volume group.
name	The name to assign to the logical volume.
size	The size, expressed as a percentage of the entire volume group capacity, to assign to the logical volume.
fstype (optional)	The file system type to create on the logical volume. If none specified, the volume is not formatted.

Key	Value Descriptions
mkfs-opts (optional)	Options, for example, `-O large_file` to pass to the mkfs command.
mode (optional)	The `mode` changes the root file system mode bits, which can be either a symbolic representation or an octal number representing the bit pattern for the new mode bits.
mount (optional)	Mount point for the file system.
consumer attributes (optional, consumer dependent)	These will vary according to the service consuming the device group. The examples section provides sample content for the different services.

Important

Multipath storage should be listed as the corresponding `/dev/mapper/mpathX`

6.8.2 Device Groups

The device-groups configuration object provides the mechanism to make the whole of a physical disk available to a service.

Key	Value Descriptions
name	An administrator-defined name for the device group.
devices	A list of named devices to be assigned to this group. There must be at least one device in the group. Multipath storage should be listed as the corresponding `/dev/mapper/mpathXf`

Key	Value Descriptions
consumer	Identifies the name of one of the storage services (for example, one of the following: Swift, Cinder, etc.) that will consume the disks in this device group.
consumer attributes	These will vary according to the service consuming the device group. The examples section provides sample content for the different services.

6.9 Memory Models

The memory-models configuration object describes details of the optional configuration of Huge Pages. It also describes the amount of memory to be allocated for virtual machine servers.

The memory-model allows the number of pages of a particular size to be configured at the server level or at the numa-node level.

The following example would configure:

- five 2 MB pages in each of numa nodes 0 and 1
- three 1 GB pages (distributed across all numa nodes)
- six 2 MB pages (distributed across all numa nodes)

```
memory-models:
    - name: COMPUTE-MEMORY-NUMA
      default-huge-page-size: 2M
      huge-pages:
        - size: 2M
          count: 5
          numa-node: 0
        - size: 2M
          count: 5
          numa-node: 1
        - size: 1G
          count: 3
        - size: 2M
          count: 6
    - name: VIRTUAL-CONTROLLER-MEMORY
      vm-size:
        ram: 6G
```

Key	Value Description
name	The name of the memory-model that is referenced from one or more server-roles.
default-huge-page-size (optional)	The default page size that will be used is specified when allocating huge pages. If not specified, the default is set by the operating system.
huge-pages	A list of huge page definitions (see below).

6.9.1 Huge Pages

Key	Value Description
size	The page size in kilobytes, megabytes, or gigabytes specified as *n*X where: *n* is an integer greater than zero X is one of "K", "M" or "G"
count	The number of pages of this size to create (must be greater than zero).
numa-node (optional)	If specified the pages will be created in the memory associated with this numa node. If not specified the pages are distributed across numa nodes by the operating system.

6.10 CPU Models

The `cpu-models` configuration object describes how CPUs are assigned for use by service components such as Nova (for VMs) and Open vSwitch (for DPDK), and whether or not those CPUs are isolated from the general kernel SMP balancing and scheduling algorithms. It also describes the number of vCPUs for virtual machine servers.

```
---
  product:
     version: 2

  cpu-models:
    - name: COMPUTE-CPU
      assignments:
        - components:
            - nova-compute-kvm
          cpu:
            - processor-ids: 0-1,3,5-7
              role: vm
        - components:
            - openvswitch
          cpu:
            - processor-ids: 4,12
              isolate: False
              role: eal
            - processor-ids: 2,10
              role: pmd
    - name: VIRTUAL-CONTROLLER-CPU
      vm-size:
         vcpus: 4
```

cpu-models

Key	Value Description
name	An administrator-defined name for the cpu model.
assignments	A list of CPU assignments .

6.10.1 CPU Assignments

assignments

Key	Value Description
components	A list of components to which the CPUs will be assigned.
cpu	A list of CPU usage objects (see *Section 6.10.2, “CPU Usage”* below).

6.10.2 CPU Usage

cpu

Key	Value Description
processor-ids	A list of CPU IDs as seen by the operating system.
isolate (optional)	A Boolean value which indicates if the CPUs are to be isolated from the general kernel SMP balancing and scheduling algorithms. The specified processor IDs will be configured in the Linux kernel isolcpus parameter. The default value is True.
role	A role within the component for which the CPUs will be used.

6.10.3 Components and Roles in the CPU Model

Component	Role	Description
nova-compute-kvm	vm	The specified processor IDs will be configured in the Nova vcpu_pin_set option.
openvswitch	eal	The specified processor IDs will be configured in the Open vSwitch DPDK EAL -c (coremask) option. Refer to the DPDK documentation for details.
	pmd	The specified processor IDs will be configured in the Open vSwitch pmd-cpu-mask option. Refer to the Open vSwitch documentation and the ovs-vswitchd.conf.db man page for details.

6.11 Interface Models

The interface-models configuration object describes how network interfaces are bonded and the mapping of network groups onto interfaces. Interface devices are identified by name and mapped to a particular physical port by the *nic-mapping* (see *Section 5.2.10.4, “NIC Mapping”*).

```
---
  product:
     version: 2

  interface-models:
     - name: INTERFACE_SET_CONTROLLER
       network-interfaces:
          - name: BONDED_INTERFACE
            device:
              name: bond0
            bond-data:
              provider: linux
              devices:
                - name: hed3
                - name: hed4
              options:
                mode: active-backup
                miimon: 200
                primary: hed3
            network-groups:
               - EXTERNAL_API
               - EXTERNAL_VM
               - GUEST

          - name: UNBONDED_INTERFACE
            device:
               name: hed0
            network-groups:
               - MGMT


       fcoe-interfaces:
          - name: FCOE_DEVICES
            devices:
              - eth7
              - eth8


     - name: INTERFACE_SET_DPDK
       network-interfaces:
```

```
       - name: BONDED_DPDK_INTERFACE
         device:
           name: bond0
         bond-data:
           provider: openvswitch
           devices:
             - name: dpdk0
             - name: dpdk1
           options:
             mode: active-backup
         network-groups:
            - GUEST
       - name: UNBONDED_DPDK_INTERFACE
         device:
            name: dpdk2
         network-groups:
            - PHYSNET2
    dpdk-devices:
      - devices:
          - name: dpdk0
          - name: dpdk1
          - name: dpdk2
            driver: igb_uio
        components:
          - openvswitch
        eal-options:
          - name: socket-mem
            value: 1024,0
          - name: n
            value: 2
        component-options:
          - name: n-dpdk-rxqs
            value: 64
```

Key	Value Description
name	An administrator-defined name for the interface model.
network-interfaces	A list of network interface definitions.

Key	Value Description
fcoe-interfaces (optional): *Section 6.11.2, "fcoe-interfaces"*	A list of network interfaces that will be used for Fibre Channel over Ethernet (FCoE). This is only needed for devices that present as a native FCoE device, not cards such as Emulex which present FCoE as a FC device.
dpdk-devices (optional)	A list of DPDK device definitions.

! Important

The devices must be "raw" device names, not names controlled via a nic-mapping.

6.11.1 network-interfaces

The network-interfaces configuration object has the following attributes:

Key	Value Description
name	An administrator-defined name for the interface
device	A dictionary containing the network device name (as seen on the associated server) and associated properties (see *Section 6.11.1.1, "network-interfaces device"* for details).
network-groups (optional if forced-network-groups is defined)	A list of one or more *network-groups* (see *Section 6.13, "Network Groups"*) containing *networks* (see *Section 6.14, "Networks"*) that can be accessed via this interface. Networks in these groups will only be configured if there is at least one *service-component* on the server which matches the list of component-endpoints defined in the *network-group*.

Key	Value Description
forced-network-groups (optional if network-groups is defined)	A list of one or more *network-groups* (see *Section 6.13, "Network Groups"*) containing *networks* (see *Section 6.14, "Networks"*) that can be accessed via this interface. Networks in these groups are always configured on the server.
passthrough-network-groups (optional)	A list of one or more network-groups (see *Section 6.13, "Network Groups"*) containing networks (see *Section 6.14, "Networks"*) that can be accessed by servers running as virtual machines on an Cloud Lifecycle Manager hypervisor server. Networks in these groups are not configured on the Cloud Lifecycle Manager hypervisor server unless they also are specified in the `network-groups` or `forced-network-groups` attributes.

6.11.1.1 network-interfaces device

network-interfaces device

The network-interfaces device configuration object has the following attributes:

Key	Value Description
name	When configuring a bond, this is used as the bond device name - the names of the devices to be bonded are specified in the bond-data section. If the interface is not bonded, this must be the name of the device specified by the nic-mapping (see NIC Mapping).

Key	Value Description
vf-count (optional)	Indicates that the interface is to be used for SR-IOV. The value is the number of virtual functions to be created. The associated device specified by the nic-mapping must have a valid nice-device-type. vf-count cannot be specified on bonded interfaces Interfaces used for SR-IOV must be associated with a network with `tagged-vlan: false`.
sriov-only (optional)	Only valid when vf-count is specified. If set to true then the interface is to be used for virtual functions only and the physical function will not be used. The default value is False.
pci-pt (optional)	If set to true then the interface is used for PCI passthrough. The default value is False.

6.11.2 fcoe-interfaces

The fcoe-interfaces configuration object has the following attributes:

Key	Value Description
name	An administrator-defined name for the group of FCOE interfaces
devices	A list of network devices that will be configured for FCOE

Key	Value Description
	Entries in this must be the name of a device specified by the nic-mapping (see *Section 6.12, “NIC Mappings”*).

6.11.3 dpdk-devices

The dpdk-devices configuration object has the following attributes:

Key	Value Descriptions
devices	A list of network devices to be configured for DPDK. See *Section 6.11.3.1, “ dpdk-devices devices”*.
eal-options	A list of key-value pairs that may be used to set DPDK Environmental Abstraction Layer (EAL) options. Refer to the DPDK documentation for details. Note that the cpu-model should be used to specify the processor IDs to be used by EAL for this component. The EAL coremask (`-c`) option will be set automatically based on the information in the cpu-model, and so should not be specified here. See *Section 6.10, “ CPU Models”*.
component-options	A list of key-value pairs that may be used to set component-specific configuration options.

6.11.3.1 dpdk-devices devices

The devices configuration object within dpdk-devices has the following attributes:

Key	Value Descriptions
name	The name of a network device to be used with DPDK. The device names must be the logical-name specified by the nic-mapping (see *Section 6.12, "NIC Mappings"*).
driver (optional)	Defines the userspace I/O driver to be used for network devices where the native device driver does not provide userspace I/O capabilities. The default value is `igb_uio`.

6.11.3.2 DPDK component-options for the openvswitch component

The following options are supported for use with the openvswitch component:

Name	Value Descriptions
n-dpdk-rxqs	Number of rx queues for each DPDK interface. Refer to the Open vSwitch documentation and the `ovs-vswitchd.conf.db` man page for details.

Note that the cpu-model should be used to define the CPU affinity of the Open vSwitch PMD (Poll Mode Driver) threads. The Open vSwitch `pmd-cpu-mask` option will be set automatically based on the information in the cpu-model. See *Section 6.10, " CPU Models"*.

6.12 NIC Mappings

The *nic-mappings* configuration object is used to ensure that the network device name used by the operating system always maps to the same physical device. A *nic-mapping* is associated to a *server* in the server definition file. Devices should be named `hedN` to avoid name clashes with any other devices configured during the operating system install as well as any interfaces that are not being managed by SUSE OpenStack Cloud, ensuring that all devices on a baremetal machine are specified in the file. An excerpt from `nic_mappings.yml` illustrates:

```
---
  product:
    version: 2

  nic-mappings:

    - name: HP-DL360-4PORT
      physical-ports:
        - logical-name: hed1
          type: simple-port
          bus-address: "0000:07:00.0"

        - logical-name: hed2
          type: simple-port
          bus-address: "0000:08:00.0"
          nic-device-type: '8086:10fb'

        - logical-name: hed3
          type: multi-port
          bus-address: "0000:09:00.0"
          port-attributes:
              port-num: 0

        - logical-name: hed4
          type: multi-port
          bus-address: "0000:09:00.0"
          port-attributes:
              port-num: 1
```

Each entry in the *nic-mappings* list has the following attributes:

Key	Value Description
name	An administrator-defined name for the mapping. This name may be used in a server definition (see *Section 6.5, "Servers"*) to apply the mapping to that server.
physical-ports	A list containing device name to address mapping information.

Each entry in the *physical-ports* list has the following attributes:

Key	Value Description
logical-name	The network device name that will be associated with the device at the specified *bus-address*. The logical-name specified here can be used as a device name in network interface model definitions. (See *Section 6.11, "Interface Models"*.)
type	The type of port. SUSE OpenStack Cloud 8 supports "simple-port" and "multi-port". Use "simple-port" if your device has a unique bus-address. Use "multi-port" if your hardware requires a "port-num" attribute to identify a single port on a multi-port device. An examples of such a device is: - Mellanox Technologies MT26438 [ConnectX VPI PCIe 2.0 5GT/s - IB QDR / 10GigE Virtualization+]
bus-address	PCI bus address of the port. Enclose the bus address in quotation marks so yaml does not misinterpret the embedded colon (`:`) characters. See *Book "Installing with Cloud Lifecycle Manager", Chapter 2 "Pre-Installation Checklist"* for details on how to determine this value.
port-attributes (required if type is `multi-port`)	Provides a list of attributes for the physical port. The current implementation supports only one attribute, "port-num". Multi-port devices share a bus-address. Use the "port-num" attribute to identify which physical port on the multi-port device to map. See *Book "Installing with Cloud Lifecycle Manager", Chapter 2*

Key	Value Description
	"Pre-Installation Checklist" for details on how to determine this value.
nic-device-type (optional)	Specifies the PCI vendor ID and device ID of the port in the format of *VENDOR_ID:DEVICE_ID*, for example, `8086:10fb`.

6.13 Network Groups

Network-groups define the overall network topology, including where service-components connect, what load balancers are to be deployed, which connections use TLS, and network routing. They also provide the data needed to map Neutron's network configuration to the physical networking.

Note

The name of the "MANAGEMENT" *network-group* cannot be changed. It must be upper case. Every SUSE OpenStack Cloud requires this network group in order to be valid.

```
---
  product:
     version: 2

  network-groups:

     - name: EXTERNAL-API
       hostname-suffix: extapi

       load-balancers:
         - provider: ip-cluster
           name: extlb
           external-name:

           tls-components:
             - default
           roles:
            - public
           cert-file: my-public-entry-scale-kvm-cert
```

```
    - name: EXTERNAL-VM
      tags:
        - neutron.l3_agent.external_network_bridge

    - name: GUEST
      hostname-suffix: guest
      tags:
        - neutron.networks.vxlan

    - name: MANAGEMENT
      hostname-suffix: mgmt
      hostname: true

      component-endpoints:
        - default

      routes:
        - default

      load-balancers:
        - provider: ip-cluster
          name: lb
          components:
            - default
          roles:
            - internal
            - admin

      tags:
        - neutron.networks.vlan:
            provider-physical-network: physnet1
```

Key	Value Description
name	An administrator-defined name for the network group. The name is used to make references from other parts of the input model.
component-endpoints (optional)	The list of *service-components* that will bind to or need direct access to networks in this network-group.
hostname (optional)	If set to true, the name of the address associated with a network in this group will be used to set the hostname of the server.

Key	Value Description
hostname-suffix (optional)	If supplied, this string will be used in the name generation (see *Section 7.2, "Name Generation"*). If not specified, the name of the network-group will be used.
load-balancers (optional)	A list of load balancers to be configured on networks in this network-group. Because load balances need a virtual IP address, any network group that contains a load balancer can only have one network associated with it. For clouds consisting of a single control plane, a load balancer may be fully defined within a `network-group` object. See Load balancer definitions in network groups. Starting in SUSE OpenStack Cloud 8, a load balancer may be defined within a `control-plane` object and referenced by name from a `network-group` object. See *Section 6.13.1, "Load Balancer Definitions in Network Groups"* in control planes.
routes (optional)	A list of *network-groups* that networks in this group provide access to via their gateway. This can include the value `default` to define the default route. A network group with no services attached to it can be used to define routes to external networks. The name of a Neutron provide network defined via configuration-data (see *Section 6.16.2.1, "neutron-provider-networks"*) can also be included in this list.

Key	Value Description
tags (optional)	A list of network tags. Tags provide the linkage between the physical network configuration and the Neutron network configuration. Starting in SUSE OpenStack Cloud 8, network tags may be defined as part of a Neutron `configuration-data` object rather than as part of a `network-group` object (see *Section 6.16.2, “Neutron Configuration Data”*).
mtu (optional)	Specifies the MTU value required for networks in this network group If not specified a default value of 1500 is used. See *Section 6.13.3, “MTU (Maximum Transmission Unit)”* on how MTU settings are applied to interfaces when there are multiple tagged networks on the same interface.

Important

`hostname` **must** be set to `true` for one, and only one, of your network groups.

A load balancer definition has the following attributes:

Key	Value Description
name	An administrator-defined name for the load balancer.
provider	The service component that implements the load balancer. Currently only `ip-cluster` (ha-proxy) is supported. Future releases will provide support for external load balancers.
roles	The list of endpoint roles that this load balancer provides (see below). Valid roles are "public", "internal", and "admin'. To ensure

Key	Value Description
	separation of concerns, the role "public" cannot be combined with any other role. See *Section 5.2.10.1.1, “Load Balancers”* for an example of how the role provides endpoint separation.
components (optional)	The list of *service-components* for which the load balancer provides a non-encrypted virtual IP address for.
tls-components (optional)	The list of *service-components* for which the load balancer provides TLS-terminated virtual IP addresses for. In SUSE OpenStack Cloud, TLS is supported both for internal and public endpoints.
external-name (optional)	The name to be registered in Keystone for the publicURL. If not specified, the virtual IP address will be registered. Note that this value cannot be changed after the initial deployment.
cert-file (optional)	The name of the certificate file to be used for TLS endpoints.

6.13.1 Load Balancer Definitions in Network Groups

In a cloud consisting of a single control-plane, a `load-balancer` may be fully defined within a `network-groups` object as shown in the examples above. See section *Section 6.3, “Load Balancers”* for a complete description of load balancer attributes.

Starting in SUSE OpenStack Cloud 8, a `load-balancer` may be defined within a `control-plane` object in which case the network-group provides just a list of load balancer names as shown below. See section *Section 6.3, “Load Balancers”* definitions in control planes.

```
network-groups:

    - name: EXTERNAL-API
      hostname-suffix: extapi
```

```
     load-balancers:
       - lb-cp1
       - lb-cp2
```

The same load balancer name can be used in multiple control-planes to make the above list simpler.

6.13.2 Network Tags

SUSE OpenStack Cloud supports a small number of network tags which may be used to convey information between the input model and the service components (currently only Neutron uses network tags). A network tag consists minimally of a tag name; but some network tags have additional attributes.

TABLE 6.1: NEUTRON.NETWORKS.VXLAN

Tag	Value Description
neutron.networks.vxlan	This tag causes Neutron to be configured to use VxLAN as the underlay for tenant networks. The associated network group will carry the VxLAN traffic.
tenant-vxlan-id-range (optional)	Used to specify the VxLAN identifier range in the format “*MIN-ID*:*MAX-ID*”. The default range is “1001:65535”. Enclose the range in quotation marks. Multiple ranges can be specified as a comma-separated list.

Example using the default ID range:

```
tags:
  - neutron.networks.vxlan
```

Example using a user-defined ID range:

```
tags:
  - neutron.networks.vxlan:
      tenant-vxlan-id-range: “1:20000”
```

Example using multiple user-defined ID range:

```
tags:
  - neutron.networks.vxlan:
      tenant-vxlan-id-range: “1:2000,3000:4000,5000:6000”
```

TABLE 6.2: NEUTRON.NETWORKS.VLAN

Tag	Value Description
neutron.networks.vlan	This tag causes Neutron to be configured for provider VLAN networks, and optionally to use VLAN as the underlay for tenant networks. The associated network group will carry the VLAN traffic. This tag can be specified on multiple network groups. However, this tag does not cause any Neutron networks to be created, that must be done in Neutron after the cloud is deployed.
provider-physical-network	The provider network name. This is the name to be used in the Neutron API for the *provider:physical_network* parameter of network objects.
tenant-vlan-id-range (optional)	This attribute causes Neutron to use VLAN for tenant networks; omit this attribute if you are using provider VLANs only. It specifies the VLAN ID range for tenant networks, in the format “*MIN-ID*:*MAX-ID*”. Enclose the range in quotation marks. Multiple ranges can be specified as a comma-separated list.

Example using a provider vlan only (may be used with tenant VxLAN):

```
tags:
  - neutron.networks.vlan:
      provider-physical-network: physnet1
```

Example using a tenant and provider VLAN:

```
tags:
  - neutron.networks.vlan:
      provider-physical-network: physnet1
      tenant-vlan-id-range: “30:50,100:200”
```

TABLE 6.3: NEUTRON.NETWORKS.FLAT

Tag	Value Description
neutron.networks.flat	This tag causes Neutron to be configured for provider flat networks. The associated network group will carry the traffic. This tag can be specified on multiple network groups. However, this tag does not cause any Neutron networks to be created, that must be done in Neutron after the cloud is deployed.
provider-physical-network	The provider network name. This is the name to be used in the Neutron API for the *provider:physical_network* parameter of network objects. When specified on multiple network groups, the name must be unique for each network group.

Example using a provider flat network:

```
tags:
  - neutron.networks.flat:
      provider-physical-network: flatnet1
```

TABLE 6.4: NEUTRON.L3_AGENT.EXTERNAL_NETWORK_BRIDGE

Tag	Value Description
neutron.l3_agent.external_network_bridge	This tag causes the Neutron L3 Agent to be configured to use the associated network group as the Neutron external network for floating IP addresses. A CIDR **should not** be defined for the associated physical network, as that will cause addresses from that network to be configured in the hypervisor. When this tag is used, provider networks cannot be used as external networks. However, this tag does not cause a Neutron external networks to be created, that must be done in Neutron after the cloud is deployed.

Example using neutron.l3_agent.external_network_bridge:

```
tags:
  - neutron.l3_agent.external_network_bridge
```

6.13.3 MTU (Maximum Transmission Unit)

A network group may optionally specify an MTU for its networks to use. Because a network-interface in the interface-model may have a mix of one untagged-vlan network group and one or more tagged-vlan network groups, there are some special requirements when specifying an MTU on a network group.

If the network group consists of untagged-vlan network(s) then its specified MTU must be greater than or equal to the MTU of any tagged-vlan network groups which are co-located on the same network-interface.

For example consider a network group with untagged VLANs, NET-GROUP-1, which is going to share (via a Network Interface definition) a device (eth0) with two network groups with tagged VLANs: NET-GROUP-2 (ID=201, MTU=1550) and NET-GROUP-3 (ID=301, MTU=9000).

The device (eth0) must have an MTU which is large enough to accommodate the VLAN in NET-GROUP-3. Since NET-GROUP-1 has untagged VLANS it will also be using this device and so it must also have an MTU of 9000, which results in the following configuration.

```
    +eth0 (9000)   <------ this MTU comes from NET-GROUP-1
    | |
    | |----+ vlan201@eth0 (1550)
    \------+ vlan301@eth0 (9000)
```

Where an interface is used only by network groups with tagged VLANs the MTU of the device or bond will be set to the highest MTU value in those groups.

For example if bond0 is configured to be used by three network groups: NET-GROUP-1 (ID=101, MTU=3000), NET-GROUP-2 (ID=201, MTU=1550) and NET-GROUP-3 (ID=301, MTU=9000).

Then the resulting configuration would be:

```
    +bond0 (9000)   <------ because of NET-GROUP-3
    | | |
    | | |--+vlan101@bond0 (3000)
    | |----+vlan201@bond0 (1550)
    |------+vlan301@bond0 (9000)
```

6.14 Networks

A network definition represents a physical L3 network used by the cloud infrastructure. Note that these are different from the network definitions that are created/configured in Neutron, although some of the networks may be used by Neutron.

```
---
   product:
     version: 2

   networks:
     - name: NET_EXTERNAL_VM
       vlanid: 102
       tagged-vlan: true
       network-group: EXTERNAL_VM

     - name: NET_GUEST
       vlanid: 103
       tagged-vlan: true
       cidr: 10.1.1.0/24
       gateway-ip: 10.1.1.1
       network-group: GUEST

     - name: NET_MGMT
       vlanid: 100
       tagged-vlan: false
       cidr: 10.2.1.0/24
       addresses:
       - 10.2.1.10-10.2.1.20
       - 10.2.1.24
       - 10.2.1.30-10.2.1.36
       gateway-ip: 10.2.1.1
       network-group: MGMT
```

Key	Value Description
name	The name of this network. The network *name* may be used in a server-group definition (see *Section 6.6, “Server Groups”*) to specify a particular network from within a network-group to be associated with a set of servers.
network-group	The name of the associated network group.

Key	Value Description
vlanid (optional)	The IEEE 802.1Q VLAN Identifier, a value in the range 1 through 4094. A *vlanid* must be specified when *tagged-vlan* is true.
tagged-vlan (optional)	May be set to `true` or `false`. If true, packets for this network carry the *vlanid* in the packet header; such packets are referred to as VLAN-tagged frames in IEEE 1Q.
cidr (optional)	The IP subnet associated with this network.
addresses (optional)	A list of IP addresses or IP address ranges (specified as *`START_ADDRESS_RANGE-END_ADDRESS_RANGE`* from which server addresses may be allocated. The default value is the first host address within the CIDR (for example, the `.1` address). The `addresses` parameter provides more flexibility than the `start-address` and `end-address` parameters and so is the preferred means of specifying this data.
start-address (optional) (deprecated)	An IP address within the *CIDR* which will be used as the start of the range of IP addresses from which server addresses may be allocated. The default value is the first host address within the *CIDR* (for example, the .1 address).
end-address (optional) (deprecated)	An IP address within the *CIDR* which will be used as the end of the range of IP addresses from which server addresses may be allocated. The default value is the last host address within the *CIDR* (for example, the .254 address of a /24). This parameter is deprecated

Key	Value Description
	in favor of the new `addresses` parameter. This parameter may be removed in a future release.
gateway-ip (optional)	The IP address of the gateway for this network. Gateway addresses must be specified if the associated *network-group* provides routes.

6.15 Firewall Rules

The configuration processor will automatically generate "allow" firewall rules for each server based on the services deployed and block all other ports. The firewall rules in the input model allow the customer to define additional rules for each network group.

Administrator-defined rules are applied after all rules generated by the Configuration Processor.

```
---
  product:
     version: 2

  firewall-rules:

     - name: PING
       network-groups:
       - MANAGEMENT
       - GUEST
       - EXTERNAL-API
       rules:
       # open ICMP echo request (ping)
       - type: allow
         remote-ip-prefix:  0.0.0.0/0
         # icmp type
         port-range-min: 8
         # icmp code
         port-range-max: 0
         protocol: icmp
```

Key	Value Description
name	An administrator-defined name for the group of rules.

Key	Value Description
network-groups	A list of *network-group* names that the rules apply to. A value of "all" matches all network-groups.
rules	A list of rules. Rules are applied in the order in which they appear in the list, apart from the control provided by the "final" option (see above). The order between sets of rules is indeterminate.

6.15.1 Rule

Each rule in the list takes the following parameters (which match the parameters of a Neutron security group rule):

Key	Value Description
type	Must `allow`
remote-ip-prefix	Range of remote addresses in CIDR format that this rule applies to.
port-range-min port-range-max	Defines the range of ports covered by the rule. Note that if the protocol is `icmp` then port-range-min is the ICMP type and port-range-max is the ICMP code.
protocol	Must be one of `tcp`, `udp`, or `icmp`.

6.16 Configuration Data

Configuration data allows values to be passed into the model to be used in the context of a specific control plane or cluster. The content and format of the data is service specific.

```
---
  product:
    version: 2
```

```
configuration-data:
  - name:  NEUTRON-CONFIG-CP1
    services:
      - neutron
    data:
      neutron_provider_networks:
      - name: OCTAVIA-MGMT-NET
        provider:
          - network_type: vlan
            physical_network: physnet1
            segmentation_id: 106
        cidr: 172.30.1.0/24
        no_gateway:  True
        enable_dhcp: True
        allocation_pools:
          - start: 172.30.1.10
            end: 172.30.1.250
        host_routes:
          # route to MANAGEMENT-NET-1
          - destination: 192.168.245.0/24
            nexthop:  172.30.1.1

      neutron_external_networks:
      - name: ext-net
        cidr: 172.31.0.0/24
        gateway: 172.31.0.1
        provider:
          - network_type: vlan
            physical_network: physnet1
            segmentation_id: 107
        allocation_pools:
          - start: 172.31.0.2
            end: 172.31.0.254

    network-tags:
      - network-group: MANAGEMENT
        tags:
          - neutron.networks.vxlan
          - neutron.networks.vlan:
              provider-physical-network: physnet1
      - network-group: EXTERNAL-VM
        tags:
          - neutron.l3_agent.external_network_bridge
```

Key	Value Description
name	An administrator-defined name for the set of configuration data.
services	A list of services that the data applies to. Note that these are service names (for example, `neutron`, `octavia`, etc.) not service-component names (`neutron-server`, `octavia-api`, etc.).
data	A service specific data structure (see below).
network-tags (optional, Neutron-only)	A list of network tags. Tags provide the linkage between the physical network configuration and the Neutron network configuration. Starting in SUSE OpenStack Cloud 8, network tags may be defined as part of a Neutron `configuration-data` object rather than as part of a `network-group` object.

6.16.1 Neutron network-tags

Key	Value Description
network-group	The name of the network-group with which the tags are associated.
tags	A list of network tags. Tags provide the linkage between the physical network configuration and the Neutron network configuration. See section Network Tags.

6.16.2 Neutron Configuration Data

Key	Value Description
neutron-provider-networks	A list of provider networks that will be created in Neutron.
neutron-external-networks	A list of external networks that will be created in Neutron. These networks will have the “router:external” attribute set to True.

6.16.2.1 neutron-provider-networks

Key	Value Description
name	The name for this network in Neutron. This name must be distinct from the names of any Network Groups in the model to enable it to be included in the “routes” value of a network group.
provider	Details of network to be created - network_type - physical_network - segmentation_id These values are passed as `--provider:` options to the Neutron `net-create` command
cidr	The CIDR to use for the network. This is passed to the Neutron `subnet-create` command.
shared (optional)	A Boolean value that specifies if the network can be shared.

Key	Value Description
	This value is passed to the Neutron `net-create` command.
allocation_pools (optional)	A list of start and end address pairs that limit the set of IP addresses that can be allocated for this network. These values are passed to the Neutron `subnet-create` command.
host_routes (optional)	A list of routes to be defined for the network. Each route consists of a `destination` in cidr format and a `nexthop` address. These values are passed to the Neutron `subnet-create` command.
gateway_ip (optional)	A gateway address for the network. This value is passed to the Neutron `subnet-create` command.
no_gateway (optional)	A Boolean value indicating that the gateway should not be distributed on this network. This is translated into the `no-gateway` option to the Neutron `subnet-create` command
enable_dhcp (optional)	A Boolean value indicating that DHCP should be enabled. The default if not specified is to not enable DHCP. This value is passed to the Neutron `subnet-create` command.

6.16.2.2 neutron-external-networks

Key	Value Description
name	The name for this network in Neutron. This name must be distinct from the names of any Network Groups in the model to enable it to be included in the “routes” value of a network group.
provider (optional)	The provider attributes are specified when using Neutron provider networks as external networks. Provider attributes should not be specified when the external network is configured with the `neutron.l3_agent.external_network_bridge`. Standard provider network attributes may be specified: - network_type - physical_network - segmentation_id These values are passed as `--provider:` options to the Neutron `net-create` command
cidr	The CIDR to use for the network. This is passed to the Neutron `subnet-create` command.
allocation_pools (optional)	A list of start and end address pairs that limit the set of IP addresses that can be allocated for this network. These values are passed to the Neutron `subnet-create` command.
gateway (optional)	A gateway address for the network.

Key	Value Description
	This value is passed to the Neutron `subnet-create` command.

6.16.3 Octavia Configuration Data

```
---
  product:
    version: 2

  configuration-data:
    - name: OCTAVIA-CONFIG-CP1
      services:
        - octavia
      data:
        amp_network_name: OCTAVIA-MGMT-NET
```

Key	Value Description
amp_network_name	The name of the Neutron provider network that Octavia will use for management access to load balancers.

6.16.4 Ironic Configuration Data

```
---
  product:
    version: 2

  configuration-data:
    - name:  IRONIC-CONFIG-CP1
      services:
        - ironic
      data:
        cleaning_network: guest-network
        enable_node_cleaning: true
        enable_oneview: false

        oneview_manager_url:
        oneview_username:
```

```
        oneview_encrypted_password:
        oneview_allow_insecure_connections:
        tls_cacert_file:
        enable_agent_drivers: true
```

Refer to the documentation on configuring Ironic for details of the above attributes.

6.16.5 Swift Configuration Data

```
---
  product:
    version: 2

  configuration-data:
  - name: SWIFT-CONFIG-CP1
    services:
      - swift
    data:
      control_plane_rings:
        swift-zones:
          - id: 1
            server-groups:
              - AZ1
          - id: 2
            server-groups:
              - AZ2
          - id: 3
            server-groups:
              - AZ3
        rings:
          - name: account
            display-name: Account Ring
            min-part-hours: 16
            partition-power: 12
            replication-policy:
              replica-count: 3

          - name: container
            display-name: Container Ring
            min-part-hours: 16
            partition-power: 12
            replication-policy:
              replica-count: 3

          - name: object-0
            display-name: General
            default: yes
            min-part-hours: 16
            partition-power: 12
            replication-policy:
              replica-count: 3
```

Refer to the documentation on *Section 11.10, “Understanding Swift Ring Specifications”* for details of the above attributes.

6.17 Pass Through

Through pass_through definitions, certain configuration values can be assigned and used.

```
product:
  version: 2

pass-through:
  global:
    esx_cloud: true
  servers:
      data:
        vmware:
          cert_check: false
          vcenter_cluster: Cluster1
          vcenter_id: BC9DED4E-1639-481D-B190-2B54A2BF5674
          vcenter_ip: 10.1.200.41
          vcenter_port: 443
          vcenter_username: administrator@vsphere.local
          id: 7d8c415b541ca9ecf9608b35b32261e6c0bf275a
```

Key	Value Description
global	These values will be used at the cloud level.
servers	These values will be assigned to a specific server(s) using the server-id.

7 Other Topics

7.1 Services and Service Components

Type	Service	Service Components
Compute		

Virtual Machine Provisioning	nova	`nova-api`
		`nova-compute`
		`nova-compute-hyperv`
		`nova-compute-ironic`
		`nova-compute-kvm`
		`nova-conductor`
		`nova-console-auth`
		`nova-esx-compute-proxy`
		`nova-metadata`
		`nova-novncproxy`
		`nova-scheduler`
		`nova-scheduler-ironic`
		`nova-placement-api`

Bare Metal Provisioning	ironic	`ironic-api` `ironic-conductor`
Networking		

Networking	neutron	`infoblox-ipam-agent`
		`neutron-dhcp-agent`
		`neutron-l2gateway-agent`
		`neutron-l3-agent`
		`neutron-lbaas-agent`
		`neutron-lbaasv2-agent`
		`neutron-metadata-agent`
		`neutron-ml2-plugin`
		`neutron-openvswitch-agent`
		`neutron-ovsvapp-agent`
		`neutron-server`
		`neutron-sriov-nic-agent`
		`neutron-vpn-agent`

Network Load Balancer	octavia	`octavia-api` `octavia-health-manager`

Type	Service	Service Components

Domain Name Service (DNS)	designate	`designate-api`
		`designate-central`
		`designate-mdns`
		`designate-mdns-external`
		`designate-pool-manager`
		`designate-zone-manager`

Storage		

Block Storage	cinder	`cinder-api`
		`cinder-backup`
		`cinder-scheduler`
		`cinder-volume`

Object Storage	swift	`swift-account`
		`swift-common`
		`swift-container`
		`swift-object`
		`swift-proxy`
		`swift-ring-builder`
		`swift-rsync`

Image		
Image Management	glance	`glance-api` `glance-registry`
Security		
Key Management	barbican	`barbican-api` `barbican-worker`
Identity and Authentication	keystone	`keystone-api`
Orchestration		

Orchestration	heat	`heat-api`
		`heat-api-cfn`
		`heat-api-cloudwatch`
		`heat-engine`

Operations		

Type	Service	Service Components

Telemetry	ceilometer	`ceilometer-agent-notification`
		`ceilometer-api`
		`ceilometer-common`
		`ceilometer-polling`

Backup and Recovery	freezer	`freezer-agent` `freezer-api`

Cloud Lifecycle Manager	ardana	`ardana-ux-services`
		`lifecycle-manager`
		`lifecycle-manager-target`

Dashboard	horizon	`horizon`

Centralized Logging	logging	`logging-api`
		`logging-producer`
		`logging-rotate`
		`logging-server`

Monitoring	monasca	`monasca-agent`
		`monasca-api`
		`monasca-dashboard`
		`monasca-liveness-check`
		`monasca-notifier`
		`monasca-persister`
		`monasca-threshold`
		`monasca-transform`

Operations Console	operations	`ops-console-web`
Openstack Functional Test Suite	tempest	`tempest`
Foundation		

OpenStack Clients	clients	`barbican-client`
		`ceilometer-client`
		`cinder-client`
		`designate-client`
		`glance-client`
		`heat-client`
		`ironic-client`

Type	Service	Service Components
		keystone-client monasca-client neutron-client nova-client openstack-client swift-client
Supporting Services	foundation	apache2 bind bind-ext influxdb ip-cluster kafka memcached mysql ntp-client ntp-server openvswitch powerdns powerdns-ext rabbitmq spark storm cassandra zookeeper

7.2 Name Generation

Names are generated by the configuration processor for all allocated IP addresses. A server connected to multiple networks will have multiple names associated with it. One of these may be assigned as the hostname for a server via the network-group configuration (see *Section 6.12, “NIC Mappings”*). Names are generated from data taken from various parts of the input model as described in the following sections.

Clusters

Names generated for servers in a cluster have the following form:

```
CLOUD-CONTROL-PLANE-CLUSTERMEMBER-PREFIXMEMBER_ID-NETWORK
```

Example: `ardana-cp1-core-m1-mgmt`

Name	Description
CLOUD	Comes from the hostname-data section of the *cloud* object (see *Section 6.1, "Cloud Configuration"*)
CONTROL-PLANE	is the *control-plane* prefix or name (see *Section 6.2, "Control Plane"*)
CLUSTER	is the *cluster-prefix* name (see *Section 6.2.1, " Clusters"*)
member-prefix	comes from the hostname-data section of the *cloud* object (see *Section 6.1, "Cloud Configuration"*)
member_id	is the ordinal within the cluster, generated by the configuration processor as servers are allocated to the cluster
network	comes from the *hostname-suffix* of the network group to which the network belongs (see *Section 6.12, "NIC Mappings"*).

Resource Nodes

Names generated for servers in a resource group have the following form:

```
CLOUD-CONTROL-PLANE-RESOURCE-PREFIXMEMBER_ID-NETWORK
```

Example: `ardana-cp1-comp0001-mgmt`

Name	Description
CLOUD	comes from the hostname-data section of the *cloud* object (see *Section 6.1, "Cloud Configuration"*).
CONTROL-PLANE	is the *control-plane* prefix or name (see *Section 6.2, "Control Plane"*).

Name	Description
RESOURCE-PREFIX	is the *resource-prefix* value name (see *Section 6.2.2, "Resources"*).
MEMBER_ID	is the ordinal within the cluster, generated by the configuration processor as servers are allocated to the cluster, padded with leading zeroes to four digits.
NETWORK	comes from the *hostname-suffix* of the network group to which the network belongs to (see *Section 6.12, "NIC Mappings"*)

7.3 Persisted Data

The configuration processor makes allocation decisions on servers and IP addresses which it needs to remember between successive runs so that if new servers are added to the input model they do not disrupt the previously deployed allocations.

To allow users to make multiple iterations of the input model before deployment SUSE OpenStack Cloud will only persist data when the administrator confirms that they are about to deploy the results via the "ready-deployment" operation. To understand this better, consider the following example:

Imagine you have completed your SUSE OpenStack Cloud deployment with servers A, B, and C and you want to add two new compute nodes by adding servers D and E to the input model.

When you add these to the input model and re-run the configuration processor it will read the persisted data for A, B, and C and allocate D and E as new servers. The configuration processor now has allocation data for A, B, C, D, and E -- which it keeps in a staging area (actually a special branch in Git) until we get confirmation that the configuration processor has done what you intended and you are ready to deploy the revised configuration.

If you notice that the role of E is wrong and it became a Swift node instead of a Nova node you need to be able to change the input model and re-run the configuration processor. This is fine because the allocations of D and E have not been confirmed, and so the configuration processor will re-read the data about A, B, C and re-allocate D and E now to the correct clusters, updating the persisted data in the staging area.

You can loop though this as many times as needed. Each time, the configuration processor is processing the deltas to what is deployed, not the results of the previous run. When you are ready to use the results of the configuration processor, you run `ready-deployment.yml` which commits the data in the staging area into the persisted data. The next run of the configuration processor will then start from the persisted data for A, B, C, D, and E.

7.3.1 Persisted Server Allocations

Server allocations are persisted by the administrator-defined server ID (see *Section 6.5, “Servers”*), and include the control plane, cluster/resource name, and ordinal within the cluster or resource group.

To guard against data loss, the configuration processor persists server allocations even when the server ID no longer exists in the input model -- for example, if a server was removed accidentally and the configuration processor allocated a new server to the same ordinal, then it would be very difficult to recover from that situation.

The following example illustrates the behavior:

A cloud is deployed with four servers with IDs of A, B, C, and D that can all be used in a resource group with `min-size=0` and `max-size=3`. At the end of this deployment they persisted state is as follows:

ID	Control Plane	Resource Group	Ordinal	State	Deployed As
A	ccp	compute	1	Allocated	mycloud-ccp-comp0001
B	ccp	compute	2	Allocated	mycloud-ccp-comp0002
C	ccp	compute	3	Allocated	mycloud-ccp-comp0003
D				Available	

(In this example server D has not been allocated because the group is at its max size, and there are no other groups that required this server)

If server B is removed from the input model and the configuration processor is re-run, the state is changed to:

ID	Control Plane	Resource Group	Ordinal	State	Deployed As
A	ccp	compute	1	Allocated	mycloud-ccp-comp0001
B	ccp	compute	2	Deleted	
C	ccp	compute	3	Allocated	mycloud-ccp-comp0003
D	ccp	compute	4	Allocated	mycloud-ccp-comp0004

The details associated with server B are still retained, but the configuration processor will not generate any deployment data for this server. Server D has been added to the group to meet the minimum size requirement but has been given a different ordinal and hence will get different names and IP addresses than were given to server B.

If server B is added back into the input model the resulting state will be:

ID	Control Plane	Resource Group	Ordinal	State	Deployed As
A	ccp	compute	1	Allocated	mycloud-ccp-comp0001
B	ccp	compute	2	Deleted	
C	ccp	compute	3	Allocated	mycloud-ccp-comp0003
D	ccp	compute	4	Allocated	mycloud-ccp-comp0004

The configuration processor will issue a warning that server B cannot be returned to the compute group because it would exceed the max-size constraint. However, because the configuration processor knows that server B is associated with this group it will not allocate it to any other group that could use it, since that might lead to data loss on that server.

If the max-size value of the group was increased, then server B would be allocated back to the group, with its previous name and addresses (mycloud-cp1-compute0002).

Note that the configuration processor relies on the server ID to identify a physical server. If the ID value of a server is changed the configuration processor will treat it as a new server. Conversely, if a different physical server is added with the same ID as a deleted server the configuration processor will assume that it is the original server being returned to the model.

You can force the removal of persisted data for servers that are no longer in the input model by running the configuration processor with the remove_deleted_servers option, like below:

```
ardana > cd ~/openstack/ardana/ansible
ardana > ansible-playbook -i hosts/localhost config-processor-run.yml \
-e remove_deleted_servers="y"
```

7.3.2 Persisted Address Allocations

The configuration processor persists IP address allocations by the generated name (see *Section 7.2, "Name Generation"* for how names are generated). As with servers. once an address has been allocated that address will remain allocated until the configuration processor is explicitly told that it is no longer required. The configuration processor will generate warnings for addresses that are persisted but no longer used.

You can remove persisted address allocations that are no longer used in the input model by running the configuration processor with the free_unused_addresses option, like below:

```
ardana > cd ~/openstack/ardana/ansible
ardana > ansible-playbook -i hosts/localhost config-processor-run.yml \
-e free_unused_addresses="y"
```

7.4 Server Allocation

The configuration processor allocates servers to a cluster or resource group in the following sequence:

1. Any *servers* that are persisted with a state of "allocated" are first returned to the *cluster* or *resource group*. Such servers are always allocated even if this contradicts the cluster size, failure-zones, or list of server roles since it is assumed that these servers are actively deployed.
2. If the *cluster* or *resource group* is still below its minimum size, then any *servers* that are persisted with a state of "deleted", but where the server is now listed in the input model (that is, the server was removed but is now back), are added to the group providing they meet the *failure-zone* and *server-role* criteria. If they do not meet the criteria then a warning is given and the *server* remains in a deleted state (that is, it is still not allocated to any other cluster or group). These *servers* are not part of the current deployment, and so you must resolve any conflicts before they can be redeployed.
3. If the *cluster* or *resource group* is still below its minimum size, the configuration processor will allocate additional *servers* that meet the *failure-zone* and *server-role* criteria. If the allocation policy is set to "strict" then the failure zones of servers already in the cluster or resource group are not considered until an equal number of servers has been allocated from each zone.

7.5 Server Network Selection

Once the configuration processor has allocated a *server* to a *cluster* or *resource group* it uses the information in the associated *interface-model* to determine which *networks* need to be configured. It does this by:

1. Looking at the *service-components* that are to run on the server (from the *control-plane* definition)
2. Looking to see which *network-group* each of those components is attached to (from the *network-groups* definition)
3. Looking to see if there are any *network-tags* related to a *service-component* running on this server, and if so, adding those *network-groups* to the list (also from the *network-groups* definition)

4. Looking to see if there are any *network-groups* that the *interface-model* says should be forced onto the server

5. It then searches the *server-group* hierarchy (as described in *Section 5.2.9.2, “Server Groups and Networks”*) to find a *network* in each of the *network-groups* it needs to attach to

If there is no *network* available to a server, either because the *interface-model* does not include the required *network-group*, or there is no *network* from that group in the appropriate part of the *server-groups* hierarchy, then the configuration processor will generate an error.

The configuration processor will also generate an error if the *server* address does not match any of the networks it will be connected to.

7.6 Network Route Validation

Once the configuration processor has allocated all of the required *servers* and matched them to the appropriate *networks*, it validates that all *service-components* have the required network routes to other *service-components*.

It does this by using the data in the services section of the input model which provides details of which *service-components* need to connect to each other. This data is not configurable by the administrator; however, it is provided as part of the SUSE OpenStack Cloud release.

For each *server*, the configuration processor looks at the list of *service-components* it runs and determines the network addresses of every other *service-component* it needs to connect to (depending on the service, this might be a virtual IP address on a load balancer or a set of addresses for the service).

If the target address is on a *network* that this *server* is connected to, then there is no routing required. If the target address is on a different *network*, then the Configuration Processor looks at each *network* the server is connected to and looks at the routes defined in the corresponding *network-group*. If the *network-group* provides a route to the *network-group* of the target address, then that route is considered valid.

Networks within the same *network-group* are always considered as routed to each other; *networks* from different *network-groups* must have an explicit entry in the `routes` stanza of the *network-group* definition. Routes to a named *network-group* are always considered before a "default" route.

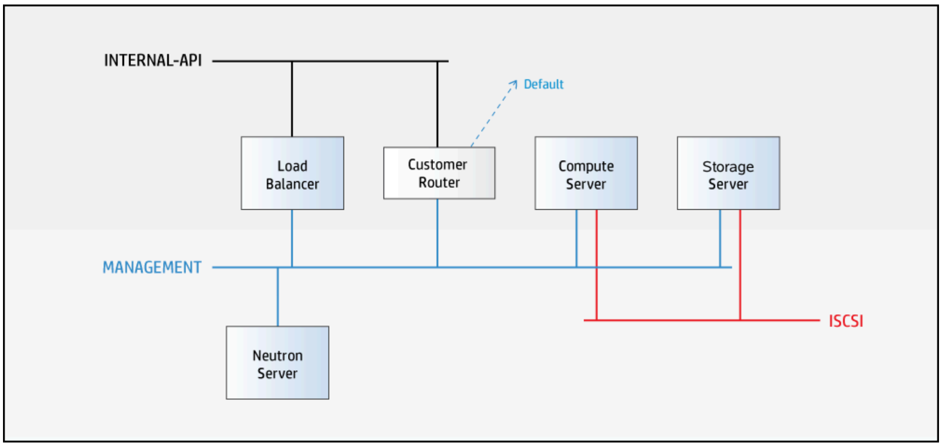
A warning is given for any routes which are using the "default" route since it is possible that the user did not intend to route this traffic. Such warning can be removed by adding the appropriate *network-group* to the list of routes.

The configuration processor provides details of all routes between networks that it is expecting to be configured in the `info/route_info.yml` file.

To illustrate how network routing is defined in the input model, consider the following example:

A compute server is configured to run nova-compute which requires access to the Neutron API servers and a block storage service. The Neutron API servers have a virtual IP address provided by a load balancer in the INTERNAL-API network-group and the storage service is connected to the ISCSI network-group. Nova-compute itself is part of the set of components attached by default to the MANAGEMENT network-group. The intention is to have virtual machines on the compute server connect to the block storage via the ISCSI network.

The physical network is shown below:



The corresponding entries in the *network-groups* are:

```
- name: INTERNAL-API
  hostname-suffix: intapi

  load-balancers:
     - provider: ip-cluster
       name: lb
       components:
         - default
       roles:
         - internal
         - admin

     - name: MANAGEMENT
```

```
      hostname-suffix: mgmt
      hostname: true

      component-endpoints:
        - default

      routes:
        - INTERNAL-API
        - default

    - name: ISCSI
      hostname-suffix: iscsi

      component-endpoints:
         - storage service
```

And the *interface-model* for the compute server looks like this:

```
- name: INTERFACE_SET_COMPUTE
  network-interfaces:
    - name: BOND0
      device:
         name: bond0
      bond-data:
         options:
            mode: active-backup
            miimon: 200
            primary: hed5
         provider: linux
         devices:
            - name: hed4
            - name: hed5
      network-groups:
        - MANAGEMENT
        - ISCSI
```

When validating the route from nova-compute to the Neutron API, the configuration processor will detect that the target address is on a network in the INTERNAL-API network group, and that the MANAGEMENT network (which is connected to the compute server) provides a route to this network, and thus considers this route valid.

When validating the route from nova-compute to a storage service, the configuration processor will detect that the target address is on a network in the ISCSInetwork group. However, because there is no service component on the compute server connected to the ISCSI network (according to the network-group definition) the ISCSI network will not have been configured on the compute server (see *Section 7.5, "Server Network Selection"*. The configuration processor will detect that the MANAGEMENT network-group provides a "default" route and thus considers the route as valid (it is, of course, valid to route ISCSI traffic). However, because this is using the default route, a warning will be issued:

```
#   route-generator-2.0       WRN: Default routing used between networks
The following networks are using a 'default' route rule. To remove this warning
either add an explicit route in the source network group or force the network to
attach in the interface model used by the servers.
  MANAGEMENT-NET-RACK1 to ISCSI-NET
    ardana-ccp-comp0001
  MANAGEMENT-NET-RACK 2 to ISCSI-NET
    ardana-ccp-comp0002
  MANAGEMENT-NET-RACK 3 to SCSI-NET
    ardana-ccp-comp0003
```

To remove this warning, you can either add ISCSI to the list of routes in the MANAGEMENT network group (routed ISCSI traffic is still a valid configuration) or force the compute server to attach to the ISCSI network-group by adding it as a forced-network-group in the interface-model, like this:

```
  - name: INTERFACE_SET_COMPUTE
      network-interfaces:
        - name: BOND0
          device:
            name: bond0
          bond-data:
            options:
               mode: active-backup
               miimon: 200
               primary: hed5
            provider: linux
            devices:
               - name: hed4
               - name: hed5
          network-groups:
             - MANAGEMENT
          forced-network-groups:
             - ISCSI
```

With the attachment to the ISCSI network group forced, the configuration processor will attach the compute server to a network in that group and validate the route as either being direct or between networks in the same network-group.

The generated `route_info.yml` file will include entries such as the following, showing the routes that are still expected to be configured between networks in the MANAGEMENT network group and the INTERNAL-API network group.

```
MANAGEMENT-NET-RACK1:
   INTERNAL-API-NET:
      default: false
      used_by:
         nova-compute:
            neutron-server:
            - ardana-ccp-comp0001
 MANAGEMENT-NET-RACK2:
   INTERNAL-API-NET:
      default: false
      used_by:
        nova-compute:
          neutron-server:
          - ardana-ccp-comp0003
```

7.7 Configuring Neutron Provider VLANs

Neutron provider VLANs are networks that map directly to an 802.1Q VLAN in the cloud provider's physical network infrastructure. There are four aspects to a provider VLAN configuration:

- Network infrastructure configuration (for example, the top-of-rack switch)
- Server networking configuration (for compute nodes and Neutron network nodes)
- Neutron configuration file settings
- Creation of the corresponding network objects in Neutron

The physical network infrastructure must be configured to convey the provider VLAN traffic as tagged VLANs to the cloud compute nodes and Neutron network nodes. Configuration of the physical network infrastructure is outside the scope of the SUSE OpenStack Cloud 8 software.

SUSE OpenStack Cloud 8 automates the server networking configuration and the Neutron configuration based on information in the cloud definition. To configure the system for provider VLANs, specify the `neutron.networks.vlan` tag with a `provider-physical-network` attribute on one or more *network-groups* as described in *Section 6.13.2, "Network Tags"*. For example (some attributes omitted for brevity):

```
network-groups:

  - name: NET_GROUP_A
    tags:
      - neutron.networks.vlan:
            provider-physical-network: physnet1

  - name: NET_GROUP_B
    tags:
      - neutron.networks.vlan:
            provider-physical-network: physnet2
```

A *network-group* is associated with a server network interface via an *interface-model* as described in *Section 6.11, "Interface Models"*. For example (some attributes omitted for brevity):

```
interface-models:
   - name: INTERFACE_SET_X
     network-interfaces:
       - device:
             name: bond0
         network-groups:
           - NET_GROUP_A
       - device:
             name: hed3
         network-groups:
           - NET_GROUP_B
```

A *network-group* used for provider VLANs may contain only a single SUSE OpenStack Cloud *network*, because that VLAN must span all compute nodes and any Neutron network nodes/controllers (that is, it is a single L2 segment). The SUSE OpenStack Cloud *network* must be defined with `tagged-vlan: false`, otherwise a Linux VLAN network interface will be created. For example:

```
networks:
   - name: NET_A
     tagged-vlan: false
     network-group: NET_GROUP_A
   - name: NET_B
     tagged-vlan: false
     network-group: NET_GROUP_B
```

When the cloud is deployed, SUSE OpenStack Cloud 8 will create the appropriate bridges on the servers, and set the appropriate attributes in the Neutron configuration files (for example, bridge_mappings).

After the cloud has been deployed, create Neutron network objects for each provider VLAN using the Neutron CLI:

```
tux > sudo neutron net-create --provider:network_type vlan \
--provider:physical_network PHYSNET1 --provider:segmentation_id 101 MYNET101
```

```
tux > sudo neutron net-create --provider:network_type vlan \
--provider:physical_network PHYSNET2 --provider:segmentation_id 234 MYNET234
```

7.8 Standalone Cloud Lifecycle Manager

All the example configurations use a “deployer-in-the-cloud” scenario where the first controller is also the deployer/Cloud Lifecycle Manager. If you want to use a standalone Cloud Lifecycle Manager, you need to add the relevant details in `control_plane.yml`, `servers.yml` and related configuration files. Detailed instructions are available at *Section 12.1, “Using a Dedicated Cloud Lifecycle Manager Node”*.

8 Configuration Processor Information Files

In addition to producing all of the data needed to deploy and configure the cloud, the configuration processor also creates a number of information files that provide details of the resulting configuration.

These files can be found in `~/openstack/my_cloud/info` after the first configuration processor run. This directory is also rebuilt each time the Configuration Processor is run.

Most of the files are in YAML format, allowing them to be used in further automation tasks if required.

File	Provides details of
`address_info.yml`	IP address assignments on each network. See *Section 8.1, "address_info.yml"*
`firewall_info.yml`	All ports that are open on each network by the firewall configuration. Can be used if you want to configure an additional firewall in front of the API network, for example. See *Section 8.2, "firewall_info.yml"*
`route_info.yml`	Routes that need to be configured between networks. See *Section 8.3, "route_info.yml"*
`server_info.yml`	How servers have been allocated, including their network configuration. Allows details of a server to be found from its ID. See *Section 8.4, "server_info.yml"*
`service_info.yml`	Details of where components of each service are deployed. See *Section 8.5, "service_info.yml"*
`control_plane_topology.yml`	Details the structure of the cloud from the perspective of each control-plane. See *Section 8.6, "control_plane_topology.yml"*
`network_topology.yml`	Details the structure of the cloud from the perspective of each control-plane. See *Section 8.7, "network_topology.yml"*

File	Provides details of
`region_topology.yml`	Details the structure of the cloud from the perspective of each region. See *Section 8.8, "region_topology.yml"*
`service_topology.yml`	Details the structure of the cloud from the perspective of each service. See *Section 8.9, "service_topology.yml"*
`private_data_metadata_ccp.yml`	Details the secrets that are generated by the configuration processor – the names of the secrets, along with the service(s) that use each secret and a list of the clusters on which the service that consumes the secret is deployed. See *Section 8.10, "private_data_metadata_ccp.yml"*
`password_change.yml`	Details the secrets that have been changed by the configuration processor – information for each secret is the same as for `private_data_metadata_ccp.yml`. See *Section 8.11, "password_change.yml"*
`explain.txt`	An explanation of the decisions the configuration processor has made when allocating servers and networks. See *Section 8.12, "explain.txt"*
`CloudDiagram.txt`	A pictorial representation of the cloud. See *Section 8.13, "CloudDiagram.txt"*

The examples are taken from the `entry-scale-kvm` example configuration.

8.1 address_info.yml

This file provides details of all the IP addresses allocated by the Configuration Processor:

```
NETWORK GROUPS
```

```
   LIST OF NETWORKS
      IP ADDRESS
         LIST OF ALIASES
```

Example:

```
EXTERNAL-API:
   EXTERNAL-API-NET:
      10.0.1.2:
         - ardana-cp1-c1-m1-extapi
      10.0.1.3:
         - ardana-cp1-c1-m2-extapi
      10.0.1.4:
         - ardana-cp1-c1-m3-extapi
      10.0.1.5:
         - ardana-cp1-vip-public-SWF-PRX-extapi
         - ardana-cp1-vip-public-FRE-API-extapi
         - ardana-cp1-vip-public-GLA-API-extapi
         - ardana-cp1-vip-public-HEA-ACW-extapi
         - ardana-cp1-vip-public-HEA-ACF-extapi
         - ardana-cp1-vip-public-NEU-SVR-extapi
         - ardana-cp1-vip-public-KEY-API-extapi
         - ardana-cp1-vip-public-MON-API-extapi
         - ardana-cp1-vip-public-HEA-API-extapi
         - ardana-cp1-vip-public-NOV-API-extapi
         - ardana-cp1-vip-public-CND-API-extapi
         - ardana-cp1-vip-public-CEI-API-extapi
         - ardana-cp1-vip-public-SHP-API-extapi
         - ardana-cp1-vip-public-OPS-WEB-extapi
         - ardana-cp1-vip-public-HZN-WEB-extapi
         - ardana-cp1-vip-public-NOV-VNC-extapi
EXTERNAL-VM:
   EXTERNAL-VM-NET: {}
GUEST:
   GUEST-NET:
      10.1.1.2:
         - ardana-cp1-c1-m1-guest
      10.1.1.3:
         - ardana-cp1-c1-m2-guest
      10.1.1.4:
         - ardana-cp1-c1-m3-guest
      10.1.1.5:
         - ardana-cp1-comp0001-guest
MANAGEMENT:
...
```

8.2 firewall_info.yml

This file provides details of all the network ports that will be opened on the deployed cloud. Data is ordered by network. If you want to configure an external firewall in front of the External API network, then you would need to open the ports listed in that section.

```
NETWORK NAME
   List of:
      PORT
      PROTOCOL
      LIST OF IP ADDRESSES
      LIST OF COMPONENTS
```

Example:

```
EXTERNAL-API:
-   addresses:
    - 10.0.1.5
    components:
    - horizon
    port: '443'
    protocol: tcp
-   addresses:
    - 10.0.1.5
    components:
    - keystone-api
    port: '5000'
    protocol: tcp
```

Port 443 (tcp) is open on network EXTERNAL-API for address 10.0.1.5 because it is used by Horizon

Port 5000 (tcp) is open on network EXTERNAL-API for address 10.0.1.5 because it is used by Keystone API

8.3 route_info.yml

This file provides details of routes between networks that need to be configured. Available routes are defined in the input model as part of the *network-groups* data; this file shows which routes will actually be used. SUSE OpenStack Cloud will reconfigure routing rules on the servers, you must configure the corresponding routes within your physical network. Routes must be configured to be symmetrical -- only the direction in which a connection is initiated is captured in this file.

Note that simple models may not require any routes, with all servers being attached to common L3 networks. The following example is taken from the `tech-preview/mid-scale-kvm` example.

```
  SOURCE-NETWORK-NAME
      TARGET-NETWORK-NAME
           default:   TRUE IF THIS IS THIS THE RESULT OF A "DEFAULT" ROUTE RULE
           used_by:
                SOURCE-SERVICE
                     TARGET-SERVICE
                     LIST OF HOSTS USING THIS ROUTE
```

Example:

```
MANAGEMENT-NET-RACK1:
    INTERNAL-API-NET:
         default: false
         used_by:
            ceilometer-client:
            ceilometer-api:
            - ardana-cp1-mtrmon-m1
            keystone-api:
            - ardana-cp1-mtrmon-m1
     MANAGEMENT-NET-RACK2:
         default: false
         used_by:
            cinder-backup:
            rabbitmq:
            - ardana-cp1-core-m1
```

A route is required from network **MANAGEMENT-NET-RACK1** to network **INTERNAL-API-NET** so that **ceilometer-client** can connect to **ceilometer-api** from server **ardana-cp1-mtrmon-m1** and to **keystone-api** from the same server.

A route is required from network **MANAGEMENT-NET-RACK1** to network **MANAGEMENT-NET-RACK2** so that **cinder-backup** can connect to **rabbitmq** from server **ardana-cp1-core-m1**

8.4 server_info.yml

This file provides details of how servers have been allocated by the Configuration Processor. This provides the easiest way to find where a specific physical server (identified by `server-id`) is being used.

```
   SERVER-ID
         failure-zone: FAILURE ZONE THAT THE SERVER WAS ALLOCATED FROM
         hostname: HOSTNAME OF THE SERVER
```

```
       net_data: NETWORK CONFIGURATION
       state:  "allocated" | "available"
```

Example:

```
 controller1:
       failure-zone: AZ1
       hostname: ardana-cp1-c1-m1-mgmt
       net_data:
            BOND0:
                 EXTERNAL-API-NET:
                     addr: 10.0.1.2
                     tagged-vlan: true
                     vlan-id: 101
                 EXTERNAL-VM-NET:
                     addr: null
                     tagged-vlan: true
                     vlan-id: 102
                 GUEST-NET:
                     addr: 10.1.1.2
                     tagged-vlan: true
                     vlan-id: 103
                 MANAGEMENT-NET:
                     addr: 192.168.10.3
                     tagged-vlan: false
                     vlan-id: 100
       state: allocated
```

8.5 service_info.yml

This file provides details of how services are distributed across the cloud.

```
CONTROL-PLANE
    SERVICE
        SERVICE COMPONENT
             LIST OF HOSTS
```

Example:

```
control-plane-1:
      neutron:
           neutron-client:
              - ardana-cp1-c1-m1-mgmt
              - ardana-cp1-c1-m2-mgmt
              - ardana-cp1-c1-m3-mgmt
```

```
             neutron-dhcp-agent:
                - ardana-cp1-c1-m1-mgmt
                - ardana-cp1-c1-m2-mgmt
                - ardana-cp1-c1-m3-mgmt
             neutron-l3-agent:
                 - ardana-cp1-comp0001-mgmt
             neutron-lbaasv2-agent:
                 - ardana-cp1-comp0001-mgmt
        ...
```

8.6 control_plane_topology.yml

This file provides details of the topology of the cloud from the perspective of each control plane:

```
control_planes:
  CONTROL-PLANE-NAME
      load-balancers:
         LOAD-BALANCER-NAME:
             address:  IP ADDRESS OF VIP
             cert-file:  NAME OF CERT FILE
             external-name: NAME TO USED FOR ENDPOINTS
             network: NAME OF THE NETWORK THIS LB IS CONNECTED TO
             network_group: NAME OF THE NETWORK GROUP THIS LB IS CONNECT TO
             provider: SERVICE COMPONENT PROVIDING THE LB
             roles:  LIST OF ROLES OF THIS LB
             services:
                SERVICE-NAME:
                    COMPONENT-NAME:
                        aliases:
                           ROLE:  NAME IN /etc/hosts
                        host-tls:  BOOLEAN, TRUE IF CONNECTION FROM LB USES TLS
                        hosts:  LIST OF HOSTS FOR THIS SERVICE
                        port:  PORT USED FOR THIS COMPONENT
                        vip-tls: BOOLEAN, TRUE IF THE VIP TERMINATES TLS
      clusters:
          CLUSTER-NAME
              failure-zones:
                 FAILURE-ZONE-NAME:
                    LIST OF HOSTS
              services:
                 SERVICE NAME:
                     components:
                        LIST OF SERVICE COMPONENTS
                     regions:
                        LIST OF REGION NAMES
      resources:
         RESOURCE-NAME:
             AS FOR CLUSTERS ABOVE
```

Example:

```
    control_planes:
    control-plane-1:
        clusters:
            cluster1:
                failure_zones:
                    AZ1:
                    - ardana-cp1-c1-m1-mgmt
                    AZ2:
                    - ardana-cp1-c1-m2-mgmt
                    AZ3:
                    - ardana-cp1-c1-m3-mgmt
                services:
                    barbican:
                        components:
                        - barbican-api
                        - barbican-worker
                        regions:
                        - region1
                                                    …
        load-balancers:
            extlb:
                address: 10.0.1.5
                cert-file: my-public-entry-scale-kvm-cert
                external-name: ''
                network: EXTERNAL-API-NET
                network-group: EXTERNAL-API
                provider: ip-cluster
                roles:
                - public
                services:
                    barbican:
                        barbican-api:
                            aliases:
                                public: ardana-cp1-vip-public-KEYMGR-API-extapi
                            host-tls: true
                            hosts:
                            - ardana-cp1-c1-m1-mgmt
                            - ardana-cp1-c1-m2-mgmt
                            - ardana-cp1-c1-m3-mgmt
                            port: '9311'
                            vip-tls: true
```

8.7 network_topology.yml

This file provides details of the topology of the cloud from the perspective of each network_group:

```
network-groups:
  NETWORK-GROUP-NAME:
      NETWORK-NAME:
          control-planes:
              CONTROL-PLANE-NAME:
                  clusters:
                     CLUSTER-NAME:
                         servers:
                            ARDANA-SERVER-NAME: ip address
                         vips:
                            IP ADDRESS: load balancer name
                  resources:
                     RESOURCE-GROUP-NAME:
                         servers:
                            ARDANA-SERVER-NAME: ip address
```

Example:

```
   network_groups:
    EXTERNAL-API:
        EXTERNAL-API-NET:
            control_planes:
                control-plane-1:
                    clusters:
                        cluster1:
                            servers:
                                ardana-cp1-c1-m1: 10.0.1.2
                                ardana-cp1-c1-m2: 10.0.1.3
                                ardana-cp1-c1-m3: 10.0.1.4
                            vips:
                                10.0.1.5: extlb
    EXTERNAL-VM:
        EXTERNAL-VM-NET:
            control_planes:
                control-plane-1:
                    clusters:
                        cluster1:
                            servers:
                                ardana-cp1-c1-m1: null
                                ardana-cp1-c1-m2: null
                                ardana-cp1-c1-m3: null
                    resources:
                        compute:
                            servers:
                                ardana-cp1-comp0001: null
```

8.8 region_topology.yml

This file provides details of the topology of the cloud from the perspective of each region. In SUSE OpenStack Cloud, multiple regions are not supported. Only `Region0` is valid.

```
regions:
  REGION-NAME:
      control-planes:
          CONTROL-PLANE-NAME:
              services:
                 SERVICE-NAME:
                     LIST OF SERVICE COMPONENTS
```

Example:

```
regions:
    region0:
        control-planes:
            control-plane-1:
                services:
                    barbican:
                    - barbican-api
                    - barbican-worker
                    ceilometer:
                    - ceilometer-common
                    - ceilometer-agent-notification
                    - ceilometer-api
                    - ceilometer-polling
                    cinder:
                    - cinder-api
                    - cinder-volume
                    - cinder-scheduler
                    - cinder-backup
```

8.9 service_topology.yml

This file provides details of the topology of the cloud from the perspective of each service:

```
services:
    SERVICE-NAME:
        components:
            COMPONENT-NAME:
                control-planes:
                    CONTROL-PLANE-NAME:
                        clusters:
                            CLUSTER-NAME:
                                LIST OF SERVERS
                        resources:
                            RESOURCE-GROUP-NAME:
                                LIST OF SERVERS
                        regions:
                            LIST OF REGIONS
```

Example:

```
services:
    freezer:
        components:
            freezer-agent:
                control_planes:
                    control-plane-1:
                        clusters:
                            cluster1:
                            - ardana-cp1-c1-m1-mgmt
                            - ardana-cp1-c1-m2-mgmt
                            - ardana-cp1-c1-m3-mgmt
                        regions:
                        - region1
                        resources:
                            compute:
                            - ardana-cp1-comp0001-mgmt
                        regions:
                        - region1
```

8.10 private_data_metadata_ccp.yml

This file provide details of the secrets that are generated by the configuration processor. The details include:

- The names of each secret
- Metadata about each secret. This is a list where each element contains details about each `component` service that uses the secret.

  - The `component` service that uses the secret, and if applicable the service that this component "consumes" when using the secret
  - The list of clusters on which the `component` service is deployed
  - The control plane `cp` on which the services are deployed
- A version number (the model version number)

```
SECRET
    METADATA
         LIST OF METADATA
             CLUSTERS
                 LIST OF CLUSTERS
             COMPONENT
             CONSUMES
             CONTROL-PLANE
     VERSION
```

For example:

```
barbican_admin_password:
    metadata:
    -   clusters:
        - cluster1
        component: barbican-api
        cp: ccp
    version: '2.0'
keystone_swift_password:
    metadata:
    -   clusters:
        - cluster1
        component: swift-proxy
        consumes: keystone-api
        cp: ccp
    version: '2.0'
metadata_proxy_shared_secret:
    metadata:
    -   clusters:
        - cluster1
        component: nova-metadata
        cp: ccp
    -   clusters:
        - cluster1
        - compute
        component: neutron-metadata-agent
        cp: ccp
```

```
version: '2.0'
…
```

8.11 password_change.yml

This file provides details equivalent to those in private_data_metadata_ccp.yml for passwords which have been changed from their original values, using the procedure outlined in the SUSE OpenStack Cloud documentation

8.12 explain.txt

This file provides details of the server allocation and network configuration decisions the configuration processor has made. The sequence of information recorded is:

- Any service components that are automatically added
- Allocation of servers to clusters and resource groups
- Resolution of the network configuration for each server
- Resolution of the network configuration of each load balancer

Example:

```
Add required services to control plane control-plane-1
======================================================
control-plane-1: Added nova-metadata required by nova-api
control-plane-1: Added swift-common required by swift-proxy
control-plane-1: Added swift-rsync required by swift-account

Allocate Servers for control plane control-plane-1
==================================================

cluster: cluster1
-----------------
  Persisted allocation for server 'controller1' (AZ1)
  Persisted allocation for server 'controller2' (AZ2)
  Searching for server with role ['CONTROLLER-ROLE'] in zones: set(['AZ3'])
  Allocated server 'controller3' (AZ3)

resource: compute
-----------------
  Persisted allocation for server 'compute1' (AZ1)
  Searching for server with role ['COMPUTE-ROLE'] in zones: set(['AZ1', 'AZ2', 'AZ3'])
```

```
        Resolve Networks for Servers
        ============================
        server: ardana-cp1-c1-m1
        ------------------------
          add EXTERNAL-API for component ip-cluster
          add MANAGEMENT for component ip-cluster
          add MANAGEMENT for lifecycle-manager (default)
          add MANAGEMENT for ntp-server (default)
          ...
          add MANAGEMENT for swift-rsync (default)
          add GUEST for tag neutron.networks.vxlan (neutron-openvswitch-agent)
          add EXTERNAL-VM for tag neutron.l3_agent.external_network_bridge (neutron-vpn-agent)
          Using persisted address 10.0.1.2 for server ardana-cp1-c1-m1 on network EXTERNAL-API-NET
          Using address 192.168.10.3 for server ardana-cp1-c1-m1 on network MANAGEMENT-NET
          Using persisted address 10.1.1.2 for server ardana-cp1-c1-m1 on network GUEST-NET

        …
        Define load balancers
        =====================

        Load balancer: extlb
        --------------------
          Using persisted address 10.0.1.5 for vip extlb ardana-cp1-vip-extlb-extapi on network EXTERNAL-API-NET
          Add nova-api for roles ['public'] due to 'default'
          Add glance-api for roles ['public'] due to 'default'
          ...

        Map load balancers to providers
        ===============================

        Network EXTERNAL-API-NET
        ------------------------
          10.0.1.5: ip-cluster nova-api roles: ['public'] vip-port: 8774 host-port: 8774
          10.0.1.5: ip-cluster glance-api roles: ['public'] vip-port: 9292 host-port: 9292
          10.0.1.5: ip-cluster keystone-api roles: ['public'] vip-port: 5000 host-port: 5000
          10.0.1.5: ip-cluster swift-proxy roles: ['public'] vip-port: 8080 host-port: 8080
          10.0.1.5: ip-cluster monasca-api roles: ['public'] vip-port: 8070 host-port: 8070
          10.0.1.5: ip-cluster heat-api-cfn roles: ['public'] vip-port: 8000 host-port: 8000
          10.0.1.5: ip-cluster ops-console-web roles: ['public'] vip-port: 9095 host-port: 9095
          10.0.1.5: ip-cluster heat-api roles: ['public'] vip-port: 8004 host-port: 8004
          10.0.1.5: ip-cluster nova-novncproxy roles: ['public'] vip-port: 6080 host-port: 6080
          10.0.1.5: ip-cluster neutron-server roles: ['public'] vip-port: 9696 host-port: 9696
          10.0.1.5: ip-cluster heat-api-cloudwatch roles: ['public'] vip-port: 8003 host-port: 8003
          10.0.1.5: ip-cluster ceilometer-api roles: ['public'] vip-port: 8777 host-port: 8777
          10.0.1.5: ip-cluster freezer-api roles: ['public'] vip-port: 9090 host-port: 9090
          10.0.1.5: ip-cluster horizon roles: ['public'] vip-port: 443 host-port: 80
          10.0.1.5: ip-cluster cinder-api roles: ['public'] vip-port: 8776 host-port: 8776
```

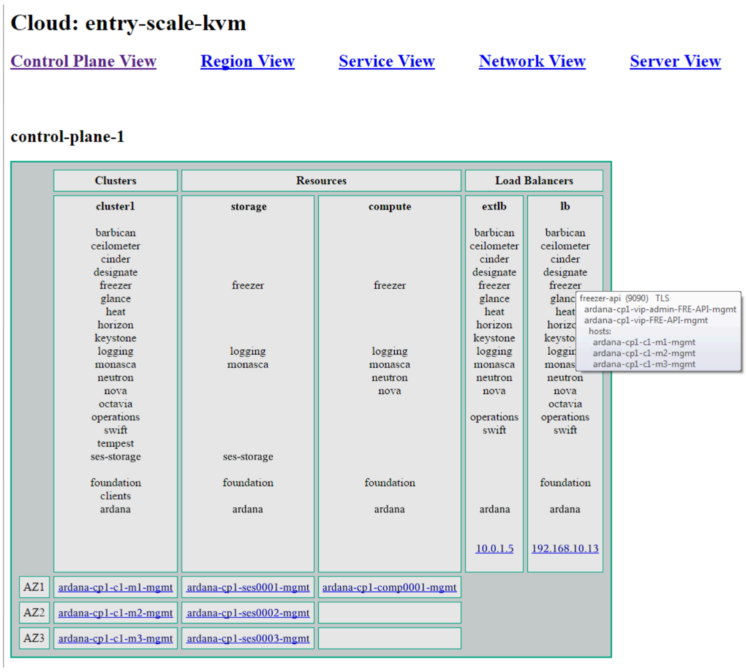
8.13 CloudDiagram.txt

This file provides a pictorial representation of the cloud. Although this file is still produced, it is superseded by the HTML output described in the following section.

8.14 HTML Representation

An HTML representation of the cloud can be found in `~/openstack/my_cloud/html` after the first Configuration Processor run. This directory is also rebuilt each time the Configuration Processor is run. These files combine the data in the input model with allocation decisions made by the Configuration processor to allow the configured cloud to be viewed from a number of different perspectives.

Most of the entries on the HTML pages provide either links to other parts of the HTML output or additional details via hover text.

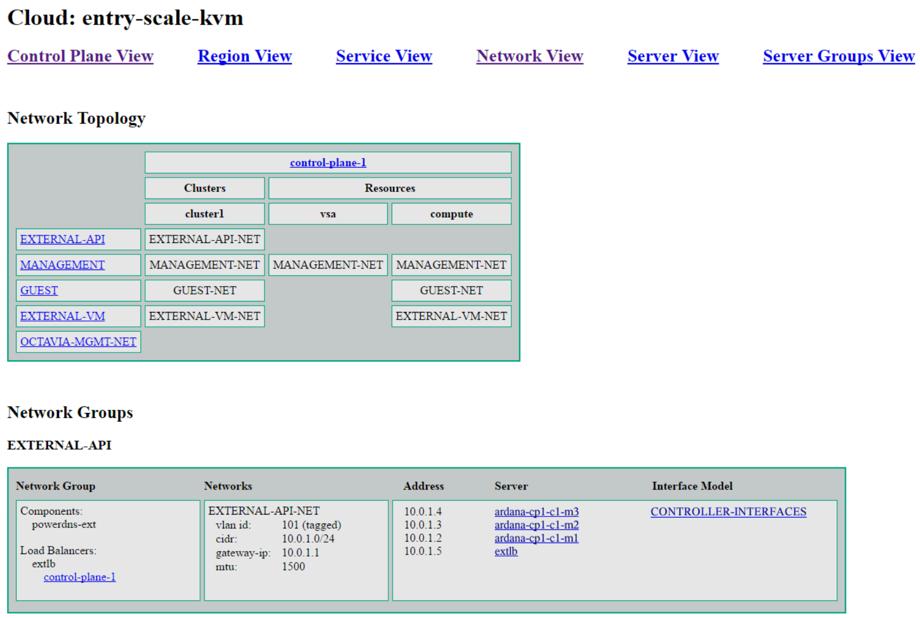


Cloud: entry-scale-kvm

control-plane-1

	Clusters	Resources		Load Balancers	
	cluster1	**storage**	**compute**	**extlb**	**lb**
	barbican ceilometer cinder designate freezer glance heat horizon keystone logging monasca neutron nova octavia operations swift tempest ses-storage foundation clients ardana	freezer logging monasca ses-storage foundation ardana	freezer logging monasca neutron nova foundation ardana	barbican ceilometer cinder designate freezer glance heat horizon keystone logging monasca neutron nova operations swift ardana 10.0.1.5	barbican ceilometer cinder designate freezer glanc heat horizo keysto loggi monas neutron nova octavia operations swift foundation ardana 192.168.10.13
AZ1	ardana-cp1-c1-m1-mgmt	ardana-cp1-ses0001-mgmt	ardana-cp1-comp0001-mgmt		
AZ2	ardana-cp1-c1-m2-mgmt	ardana-cp1-ses0002-mgmt			
AZ3	ardana-cp1-c1-m3-mgmt	ardana-cp1-ses0003-mgmt			

```
freezer-api (9090) TLS
 ardana-cp1-vip-admin-FRE-API-mgmt
 ardana-cp1-vip-FRE-API-mgmt
  hosts:
   ardana-cp1-c1-m1-mgmt
   ardana-cp1-c1-m2-mgmt
   ardana-cp1-c1-m3-mgmt
```



Cloud: entry-scale-kvm

Control Plane View   Region View   Service View   Network View   Server View   Server Groups View

Network Topology

	control-plane-1		
	Clusters	Resources	
	cluster1	vsa	compute
EXTERNAL-API	EXTERNAL-API-NET		
MANAGEMENT	MANAGEMENT-NET	MANAGEMENT-NET	MANAGEMENT-NET
GUEST	GUEST-NET		GUEST-NET
EXTERNAL-VM	EXTERNAL-VM-NET		EXTERNAL-VM-NET
OCTAVIA-MGMT-NET			

Network Groups

EXTERNAL-API

Network Group	Networks	Address	Server	Interface Model
Components: powerdns-ext Load Balancers: extlb control-plane-1	EXTERNAL-API-NET vlan id: 101 (tagged) cidr: 10.0.1.0/24 gateway-ip: 10.0.1.1 mtu: 1500	10.0.1.4 10.0.1.3 10.0.1.2 10.0.1.5	ardana-cp1-c1-m3 ardana-cp1-c1-m2 ardana-cp1-c1-m1 extlb	CONTROLLER-INTERFACES

9 Example Configurations

The SUSE OpenStack Cloud 8 system ships with a collection of pre-qualified example configurations. These are designed to help you to get up and running quickly with a minimum number of configuration changes.

The SUSE OpenStack Cloud input model allows a wide variety of configuration parameters that can, at first glance, appear daunting. The example configurations are designed to simplify this process by providing pre-built and pre-qualified examples that need only a minimum number of modifications to get started.

9.1 SUSE OpenStack Cloud Example Configurations

This section briefly describes the various example configurations and their capabilities. It also describes in detail, for the entry-scale-kvm example, how you can adapt the input model to work in your environment.

The following pre-qualified examples are shipped with SUSE OpenStack Cloud 8:

Name	Location
Section 9.3.1, "Entry-Scale Cloud"	`~/openstack/examples/entry-scale-kvm`
Section 9.3.2, "Entry Scale Cloud with Metering and Monitoring Services"	`~/openstack/examples/entry-scale-kvm-mml`
Section 9.4.1, "Single-Region Entry-Scale Cloud with a Mix of KVM and ESX Hypervisors"	`~/openstack/examples/entry-scale-esx-kvm`
Section 9.4.2, "Single-Region Entry-Scale Cloud with Metering and Monitoring Services, and a Mix of KVM and ESX Hypervisors"	`~/openstack/examples/entry-scale-esx-kvm-mml`
Section 9.5.1, "Entry-scale Swift Model"	`~/openstack/examples/entry-scale-swift`
Section 9.6.1, "Entry-Scale Cloud with Ironic Flat Network"	`~/openstack/examples/entry-scale-ironic-flat-network`
Section 9.6.2, "Entry-Scale Cloud with Ironic Multi-Tenancy"	`~/openstack/examples/entry-scale-ironic-multi-tenancy`

Name	Location
Section 9.3.3, “Single-Region Mid-Size Model”	`~/openstack/examples/mid-scale-kvm`

The entry-scale systems are designed to provide an entry-level solution that can be scaled from a small number of nodes to a moderately high node count (approximately 100 compute nodes, for example).

In the mid-scale model, the cloud control plane is subdivided into a number of dedicated service clusters to provide more processing power for individual control plane elements. This enables a greater number of resources to be supported (compute nodes, Swift object servers). This model also shows how a segmented network can be expressed in the SUSE OpenStack Cloud model.

9.2 Alternative Configurations

In SUSE OpenStack Cloud 8 there are alternative configurations that we recommend for specific purposes and this section we will outline them.

- *Section 12.1, “Using a Dedicated Cloud Lifecycle Manager Node”*
- *Section 12.2, “Configuring SUSE OpenStack Cloud without DVR”*
- *Section 12.3, “Configuring SUSE OpenStack Cloud with Provider VLANs and Physical Routers Only”*
- *Section 12.4, “Considerations When Installing Two Systems on One Subnet”*

The Ironic multi-tenancy feature uses Neutron to manage the tenant networks. The interaction between Neutron and the physical switch is facilitated by Neutron's Modular Layer 2 (ML2) plugin. The Neutron ML2 plugin supports drivers to interact with various networks, as each vendor may have their own extensions. Those drivers are referred to as *Neutron ML2 mechanism drivers*, or simply *mechanism drivers*.

The Ironic multi-tenancy feature has been validated using OpenStack genericswitch mechanism driver. However, if the given physical switch requires a different mechanism driver, you must update the input model accordingly. To update the input model with a custom ML2 mechanism driver, specify the relevant information in the `multi_tenancy_switch_config:` section of the `data/ironic/ironic_config.yml` file.

9.3 KVM Examples

9.3.1 Entry-Scale Cloud

This example deploys an entry-scale cloud.

Control Plane

Cluster1 3 nodes of type `CONTROLLER-ROLE` run the core OpenStack services, such as Keystone, Nova API, Glance API, Neutron API, Horizon, and Heat API.

Cloud Lifecycle Manager

The Cloud Lifecycle Manager runs on one of the control-plane nodes of type `CONTROLLER-ROLE`. The IP address of the node that will run the Cloud Lifecycle Manager needs to be included in the `data/servers.yml` file.

Resource Nodes

- **Compute** One node of type `COMPUTE-ROLE` runs Nova Compute and associated services.
- **Object Storage** Minimal Swift resources are provided by the control plane.

Additional resource nodes can be added to the configuration.

Networking

This example requires the following networks:

- **IPMI** network connected to the lifecycle-manager and the IPMI ports of all servers.

Nodes require a pair of bonded NICs which are used by the following networks:

- **External API** The network for making requests to the cloud.
- **External VM** This network provides access to VMs via floating IP addresses.
- **Cloud Management** This network is used for all internal traffic between the cloud services. It is also used to install and configure the nodes. The network needs to be on an untagged VLAN.
- **Guest** The network that carries traffic between VMs on private networks within the cloud.

The EXTERNAL API network must be reachable from the EXTERNAL VM network for VMs to be able to make API calls to the cloud.

An example set of networks is defined in data/networks.yml. The file needs to be modified to reflect your environment.

The example uses the devices hed3 and hed4 as a bonded network interface for all services. The name given to a network interface by the system is configured in the file data/net_interfaces.yml. That file needs to be edited to match your system.

Local Storage

All servers should present a single OS disk, protected by a RAID controller. This disk needs to be at least 512 GB in capacity. In addition the example configures one additional disk depending on the role of the server:

- **Controllers** /dev/sdb and /dev/sdc are configured to be used by Swift.
- **Compute Servers** /dev/sdb is configured as an additional Volume Group to be used for VM storage

Additional disks can be configured for any of these roles by editing the corresponding data/disks_*.yml file

9.3.2 Entry Scale Cloud with Metering and Monitoring Services

This example deploys an entry-scale cloud that provides metering and monitoring services and runs the database and messaging services in their own cluster.

Control Plane

- **Cluster1** 2 nodes of type CONTROLLER-ROLE run the core OpenStack services, such as Keystone, Nova API, Glance API, Neutron API, Horizon, and Heat API.
- **Cluster2** 3 nodes of type MTRMON-ROLE, run the OpenStack services for metering and monitoring (for example, Ceilometer, Monasca and Logging).
- **Cluster3** 3 nodes of type DBMQ-ROLE that run clustered database and RabbitMQ services to support the cloud infrastructure. 3 nodes are required for high availability.

Cloud Lifecycle Manager

The Cloud Lifecycle Manager runs on one of the control-plane nodes of type `CONTROLLER-ROLE`. The IP address of the node that will run the Cloud Lifecycle Manager needs to be included in the `data/servers.yml` file.

Resource Nodes

- **Compute** 1 node of type `COMPUTE-ROLE` runs Nova Compute and associated services.
- **Object Storage** Minimal Swift resources are provided by the control plane.

Additional resource nodes can be added to the configuration.

Networking

This example requires the following networks:

- **IPMI** network connected to the lifecycle-manager and the IPMI ports of all servers.

Nodes require a pair of bonded NICs which are used by the following networks:

- **External API** The network for making requests to the cloud.
- **External VM** The network that provides access to VMs via floating IP addresses.
- **Cloud Management** This is the network that is used for all internal traffic between the cloud services. It is also used to install and configure the nodes. The network needs to be on an untagged VLAN.
- **Guest** The network that carries traffic between VMs on private networks within the cloud.

The `EXTERNAL API` network must be reachable from the `EXTERNAL VM` network for VMs to be able to make API calls to the cloud.

An example set of networks is defined in `data/networks.yml`. The file needs to be modified to reflect your environment.

The example uses the devices `hed3` and `hed4` as a bonded network interface for all services. The name given to a network interface by the system is configured in the file `data/net_interfaces.yml`. That file needs to be edited to match your system.

Local Storage

All servers should present a single OS disk, protected by a RAID controller. This disk needs to be at least 512 GB of capacity. In addition, the example configures one additional disk depending on the role of the server:

- **Core Controllers** `/dev/sdb` and `/dev/sdc` is configured to be used by Swift.
- **DBMQ Controllers** `/dev/sdb` is configured as an additional Volume Group to be used by the database and RabbitMQ.
- **Compute Servers** `/dev/sdb` is configured as an additional Volume Group to be used for VM storage.

Additional disks can be configured for any of these roles by editing the corresponding `data/disks_*.yml` file.

9.3.3 Single-Region Mid-Size Model

The mid-size model is intended as a template for a moderate sized cloud. The Control plane is made up of multiple server clusters to provide sufficient computational, network and IOPS capacity for a mid-size production style cloud.

Control Plane

- **Core Cluster** runs core OpenStack Services, such as Keystone, Nova API, Glance API, Neutron API, Horizon, and Heat API. Default configuration is two nodes of role type `CORE-ROLE`.
- **Metering and Monitoring Cluster** runs the OpenStack Services for metering and monitoring (for example, Ceilometer, Monasca and logging). Default configuration is three nodes of role type `MTRMON-ROLE`.
- **Database and Message Queue Cluster** runs clustered MariaDB and RabbitMQ services to support the Ardana cloud infrastructure. Default configuration is three nodes of role type `DBMQ-ROLE`. Three nodes are required for high availability.
- **Swift PAC Cluster** runs the Swift Proxy, Account and Container services. Default configuration is three nodes of role type `SWPAC-ROLE`.
- **Neutron Agent Cluster** Runs Neutron VPN (L3), DHCP, Metadata and OpenVswitch agents. Default configuration is two nodes of role type `NEUTRON-ROLE`.

Cloud Lifecycle Manager

The Cloud Lifecycle Manager runs on one of the control-plane nodes of type `CONTROLLER-ROLE`. The IP address of the node that will run the Cloud Lifecycle Manager needs to be included in the `data/servers.yml` file.

Resource Nodes

- **Compute** runs Nova Compute and associated services. Runs on nodes of role type `COMPUTE-ROLE`. This model lists 3 nodes. 1 node is the minimum requirement.
- **Object Storage** 3 nodes of type `SOWBJ-ROLE` run the Swift Object service. The minimum node count should match your Swift replica count.

The minimum node count required to run this model unmodified is 19 nodes. This can be reduced by consolidating services on the control plane clusters.

Networking

This example requires the following networks:

- **IPMI** network connected to the lifecycle-manager and the IPMI ports of all servers.

Nodes require a pair of bonded NICs which are used by the following networks:

- **External API** The network for making requests to the cloud.
- **Internal API** This network is used within the cloud for API access between services.
- **External VM** This network provides access to VMs via floating IP addresses.
- **Cloud Management** This network is used for all internal traffic between the cloud services. It is also used to install and configure the nodes. The network needs to be on an untagged VLAN.
- **Guest** The network that carries traffic between VMs on private networks within the cloud.
- **SWIFT** This network is used for internal Swift communications between the Swift nodes.

The `EXTERNAL API` network must be reachable from the `EXTERNAL VM` network for VMs to be able to make API calls to the cloud.
An example set of networks is defined in `data/networks.yml`. The file needs to be modified to reflect your environment.

The example uses the devices `hed3` and `hed4` as a bonded network interface for all services. The name given to a network interface by the system is configured in the file `data/net_interfaces.yml`. That file needs to be edited to match your system.

9.3.3.1 Adapting the Mid-Size Model to Fit Your Environment

The minimum set of changes you need to make to adapt the model for your environment are:

- Update `servers.yml` to list the details of your baremetal servers.
- Update the `networks.yml` file to replace network CIDRs and VLANs with site specific values.
- Update the `nic_mappings.yml` file to ensure that network devices are mapped to the correct physical port(s).
- Review the disk models (`disks_*.yml`) and confirm that the associated servers have the number of disks required by the disk model. The device names in the disk models might need to be adjusted to match the probe order of your servers. The default number of disks for the Swift nodes (3 disks) is set low on purpose to facilitate deployment on generic hardware. For production scale Swift the servers should have more disks. For example, 6 on SWPAC nodes and 12 on SWOBJ nodes. If you allocate more Swift disks then you should review the ring power in the Swift ring configuration. This is documented in the Swift section. Disk models are provided as follows:
  - DISK SET CONTROLLER: Minimum 1 disk
  - DISK SET DBMQ: Minimum 3 disks
  - DISK SET COMPUTE: Minimum 2 disks
  - DISK SET SWPAC: Minimum 3 disks
  - DISK SET SWOBJ: Minimum 3 disks
- Update the `netinterfaces.yml` file to match the server NICs used in your configuration. This file has a separate interface model definition for each of the following:
  - INTERFACE SET CONTROLLER
  - INTERFACE SET DBMQ

- INTERFACE SET SWPAC
- INTERFACE SET SWOBJ
- INTERFACE SET COMPUTE

9.4 ESX Examples

9.4.1 Single-Region Entry-Scale Cloud with a Mix of KVM and ESX Hypervisors

This example deploys a cloud which mixes KVM and ESX hypervisors.

Control Plane

Cluster1 3 nodes of type `CONTROLLER-ROLE` run the core OpenStack services, such as Keystone, Nova API, Glance API, Neutron API, Horizon, and Heat API.

Cloud Lifecycle Manager

The Cloud Lifecycle Manager runs on one of the control-plane nodes of type `CONTROLLER-ROLE`. The IP address of the node that will run the Cloud Lifecycle Manager needs to be included in the `data/servers.yml` file.

Resource Nodes

- Compute:
  - **KVM** runs Nova Computes and associated services. It runs on nodes of role type `COMPUTE-ROLE`.
  - **ESX** provides ESX Compute services. OS and software on this node is installed by user.

ESX Resource Requirements

1. User needs to supply vSphere server
2. User needs to deploy the ovsvapp network resources using the vSphere GUI (*Book “Installing with Cloud Lifecycle Manager”, Chapter 15 “Installing ESX Computes and OVSvAPP”, Section 15.8 “Configuring the Required Distributed vSwitches and Port Groups”, Sec-*

tion 15.8.2 “Creating ESXi MGMT DVS and Required Portgroup”) by running the `neutron-create-ovsvapp-resources.yml` playbook (*Book* “Installing with Cloud Lifecycle Manager”, *Chapter 15 “Installing ESX Computes and OVSvAPP”, Section 15.8 “Configuring the Required Distributed vSwitches and Port Groups”, Section 15.8.3 “Configuring OVSvApp Network Resources Using Ansible-Playbook”*) or via Python-Networking-vSphere (*Book* “Installing with Cloud Lifecycle Manager”, *Chapter 15 “Installing ESX Computes and OVSvAPP”, Section 15.8 “Configuring the Required Distributed vSwitches and Port Groups”, Section 15.8.4 “Configuring OVSVAPP Using Python-Networking-vSphere”*)
The following DVS and DVPGs need to be created and configured for each cluster in each ESX hypervisor that will host an OvsVapp appliance. The settings for each DVS and DVPG are specific to your system and network policies. A JSON file example is provided in the documentation, but it needs to be edited to match your requirements.

DVS	Port Groups assigned to DVS
MGMT	MGMT-PG, ESX-CONF-PG, GUEST-PG
TRUNK	TRUNK-PG

3. User needs to deploy ovsvapp appliance (`OVSVAPP-ROLE`) and nova-proxy appliance (`ESX-COMPUTE-ROLE`)

4. User needs to add required information related to compute proxy and OVSvApp Nodes

Networking

This example requires the following networks:

- **IPMI**network connected to the lifecycle-manager and the IPMI ports of all nodes, except the ESX hypervisors.

Nodes require a pair of bonded NICs which are used by the following networks:

- **External API** The network for making requests to the cloud.
- **External VM** The network that provides access to VMs via floating IP addresses.
- **Cloud Management** The network used for all internal traffic between the cloud services. It is also used to install and configure the nodes. The network needs to be on an untagged VLAN.

- **Guest** This network carries traffic between VMs on private networks within the cloud.
- **SES** This is the network that control-plane and compute-node clients use to talk to the external SUSE Enterprise Storage.
- **TRUNK** is the network that is used to apply security group rules on tenant traffic. It is managed by the cloud admin and is restricted to the vCenter environment.
- **ESX-CONF-NET** network is used only to configure the ESX compute nodes in the cloud. This network should be different from the network used with PXE to stand up the cloud control-plane.

This example's set of networks is defined in `data/networks.yml`. The file needs to be modified to reflect your environment.
The example uses the devices `hed3` and `hed4` as a bonded network interface for all services. The name given to a network interface by the system is configured in the file `data/net_interfaces.yml`. That file needs to be edited to match your system.

Local Storage

All servers should present a single OS disk, protected by a RAID controller. This disk needs to be at least 512 GB in capacity. In addition, the example configures additional disk depending on the node's role:

- **Controllers** `/dev/sdb` and `/dev/sdc` are configured to be used by Swift
- **Compute Servers** `/dev/sdb` is configured as an additional Volume Group to be used for VM storage

Additional disks can be configured for any of these roles by editing the corresponding `data/disks_*.yml` file.

9.4.2 Single-Region Entry-Scale Cloud with Metering and Monitoring Services, and a Mix of KVM and ESX Hypervisors

This example deploys a cloud which mixes KVM and ESX hypervisors, provides metering and monitoring services, and runs the database and messaging services in their own cluster.

Control Plane

- **Cluster1** 2 nodes of type `CONTROLLER-ROLE` run the core OpenStack services, such as Keystone, Nova API, Glance API, Neutron API, Horizon, and Heat API.
- **Cluster2** 3 nodes of type `MTRMON-ROLE`, run the OpenStack services for metering and monitoring (for example, Ceilometer, Monasca and Logging).
- **Cluster3** 3 nodes of type `DBMQ-ROLE`, run clustered database and RabbitMQ services to support the cloud infrastructure. 3 nodes are required for high availability.

Cloud Lifecycle Manager

The Cloud Lifecycle Manager runs on one of the control-plane nodes of type `CONTROLLER-ROLE`. The IP address of the node that will run the Cloud Lifecycle Manager needs to be included in the `data/servers.yml` file.

Resource Nodes

- Compute:
  - **KVM** runs Nova Computes and associated services. It runs on nodes of role type `COMPUTE-ROLE`.
  - **ESX** provides ESX Compute services. OS and software on this node is installed by user.

ESX Resource Requirements

1. User needs to supply vSphere server
2. User needs to deploy the ovsvapp network resources using the vSphere GUI or by running the `neutron-create-ovsvapp-resources.yml` playbook
   The following DVS and DVPGs need to be created and configured for each cluster in each ESX hypervisor that will host an OvsVapp appliance. The settings for each DVS and DVPG are specific to your system and network policies. A JSON file example is provided in the documentation, but it needs to be edited to match your requirements.
   - ESX-CONF (DVS and DVPG) connected to ovsvapp eth0 and compute-proxy eth0
   - MANAGEMENT (DVS and DVPG) connected to ovsvapp eth1, eth2, eth3 and compute-proxy eth1

3. User needs to deploy ovsvapp appliance (`OVSVAPP-ROLE`) and nova-proxy appliance (`ESX-COMPUTE-ROLE`)
4. User needs to add required information related to compute proxy and OVSvApp Nodes

Networking

This example requires the following networks:

- **IPMI**network connected to the lifecycle-manager and the IPMI ports of all nodes, except the ESX hypervisors.

Nodes require a pair of bonded NICs which are used by the following networks:

- **External API** The network for making requests to the cloud.
- **External VM** The network that provides access to VMs (via floating IP addresses).
- **Cloud Management** This network is used for all internal traffic between the cloud services. It is also used to install and configure the nodes. The network needs to be on an untagged VLAN.
- **Guest** This is the network that will carry traffic between VMs on private networks within the cloud.
- **TRUNK** is the network that will be used to apply security group rules on tenant traffic. It is managed by the cloud admin and is restricted to the vCenter environment.
- **ESX-CONF-NET** network is used only to configure the ESX compute nodes in the cloud. This network should be different from the network used with PXE to stand up the cloud control-plane.

This example's set of networks is defined in `data/networks.yml` . The file needs to be modified to reflect your environment.
The example uses the devices `hed3` and `hed4` as a bonded network interface for all services. The name given to a network interface by the system is configured in the file `data/net_interfaces.yml` . That file needs to be edited to match your system.

Local Storage

All servers should present a single OS disk, protected by a RAID controller. This disk needs to be at least 512 GB in capacity. In addition, the example configures additional disk depending on the node's role:

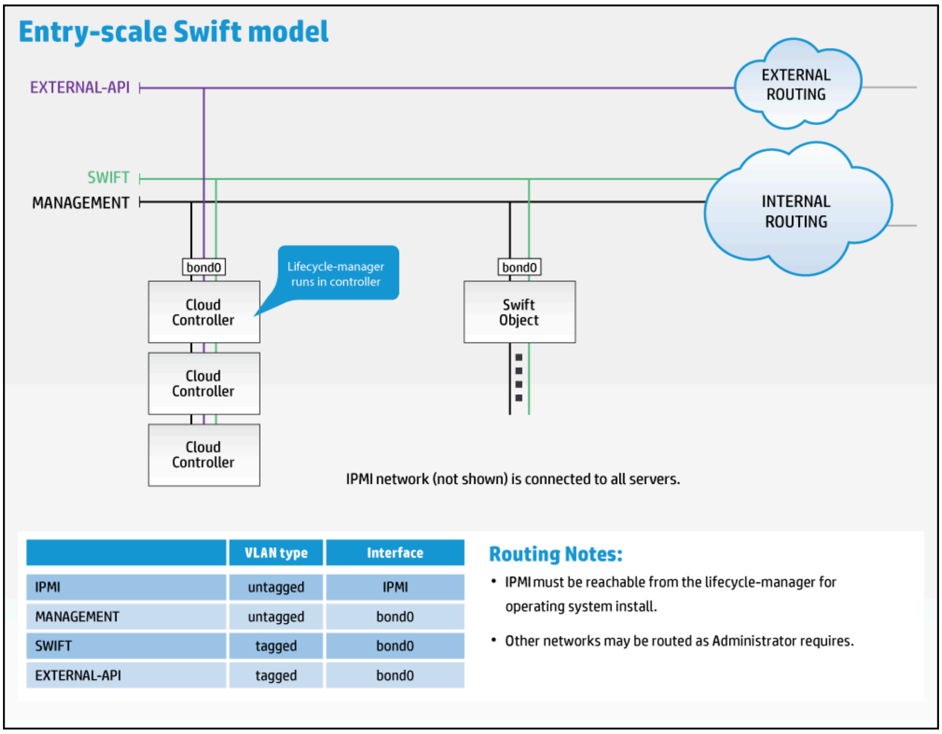
- **Controllers** `/dev/sdb` and `/dev/sdc` are configured to be used by Swift.
- **Compute Servers** `/dev/sdb` is configured as an additional Volume Group to be used for VM storage

Additional disks can be configured for any of these roles by editing the corresponding `data/disks_*.yml` file

9.5 Swift Examples

9.5.1 Entry-scale Swift Model

This example shows how SUSE OpenStack Cloud can be configured to provide a Swift-only configuration, consisting of three controllers and one or more Swift object servers.



IPMI network (not shown) is connected to all servers.

	VLAN type	Interface
IPMI	untagged	IPMI
MANAGEMENT	untagged	bond0
SWIFT	tagged	bond0
EXTERNAL-API	tagged	bond0

Routing Notes:

- IPMI must be reachable from the lifecycle-manager for operating system install.
- Other networks may be routed as Administrator requires.

The example requires the following networks:

- **External API** - The network for making requests to the cloud.
- **Swift** - The network for all data traffic between the Swift services.
- **Management** - This network that is used for all internal traffic between the cloud services, including node provisioning. This network must be on an untagged VLAN.

All of these networks are configured to be presented via a pair of bonded NICs. The example also enables provider VLANs to be configured in Neutron on this interface.

In the diagram "External Routing" refers to whatever routing you want to provide to allow users to access the External API. "Internal Routing" refers to whatever routing you want to provide to allow administrators to access the Management network.

If you are using SUSE OpenStack Cloud to install the operating system, then an IPMI network connected to the IPMI ports of all servers and routable from the Cloud Lifecycle Manager is also required for BIOS and power management of the node during the operating system installation process.

In the example the controllers use one disk for the operating system and two disks for Swift proxy and account storage. The Swift object servers use one disk for the operating system and four disks for Swift storage. These values can be modified to suit your environment.

These recommended minimums are based on the included with the base installation and are suitable only for demo environments. For production systems you will want to consider your capacity and performance requirements when making decisions about your hardware.

The `entry-scale-swift` example runs the Swift proxy, account and container services on the three controller servers. However, it is possible to extend the model to include the Swift proxy, account and container services on dedicated servers (typically referred to as the Swift proxy servers). If you are using this model, we have included the recommended Swift proxy servers specs in the table below.

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
Dedicated Cloud Lifecycle Man-	Lifecycle-manager	1	300 GB	8 GB	1 x 10 Gbit/s with PXE Support	8 CPU (64-bit) cores

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
ager (optional)						total (Intel x86_64)
Control Plane	Controller	3	• 1 x 600 GB (minimum) - operating system drive • 2 x 600 GB (minimum) - Swift account/container data drive	64 GB	2 x 10 Gbit/s with one PXE enabled port	8 CPU (64-bit) cores total (Intel x86_64)

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
Swift Object	swobj	3	If using x3 replication only: - 1 x 600 GB (minimum, see considerations at bottom of page for more details) If using Erasure Codes only or a mix of x3 repli-	32 GB (see considerations at bottom of page for more details)	2 x 10 Gbit/s with one PXE enabled port	8 CPU (64-bit) cores total (Intel x86_64)

Node Type	Role Name	Required Number	Server Hardware - Minimum Requirements and Recommendations			
			Disk	Memory	Network	CPU
			cation and Erasure Codes: - 6 x 600 GB (minimum, see considerations at bottom of page for more details)			
Swift Proxy, Account, and Container	swpac	3	2 x 600 GB (minimum, see considerations at bottom of page for more details)	64 GB (see considerations at bottom of page for more details)	2 x 10 Gbit/s with one PXE enabled port	8 CPU (64-bit) cores total (Intel x86_64)

Note

The disk speeds (RPM) chosen should be consistent within the same ring or storage policy. It is best to not use disks with mixed disk speeds within the same Swift ring.

Considerations for your Swift object and proxy, account, container servers RAM and disk capacity needs

Swift can have a diverse number of hardware configurations. For example, a Swift object server may have just a few disks (minimum of 6 for erasure codes) or up to 70 and beyond. The memory requirement needs to be increased as more disks are added. The general rule of thumb for memory needed is 0.5 GB per TB of storage. For example, a system with 24 hard drives at 8TB each, giving a total capacity of 192TB, should use 96GB of RAM. However, this does not work well for a system with a small number of small hard drives or a very large number of very large drives. So, if after calculating the memory given this guideline, if the answer is less than 32GB then go with 32GB of memory minimum and if the answer is over 256GB then use 256GB maximum, no need to use more memory than that.

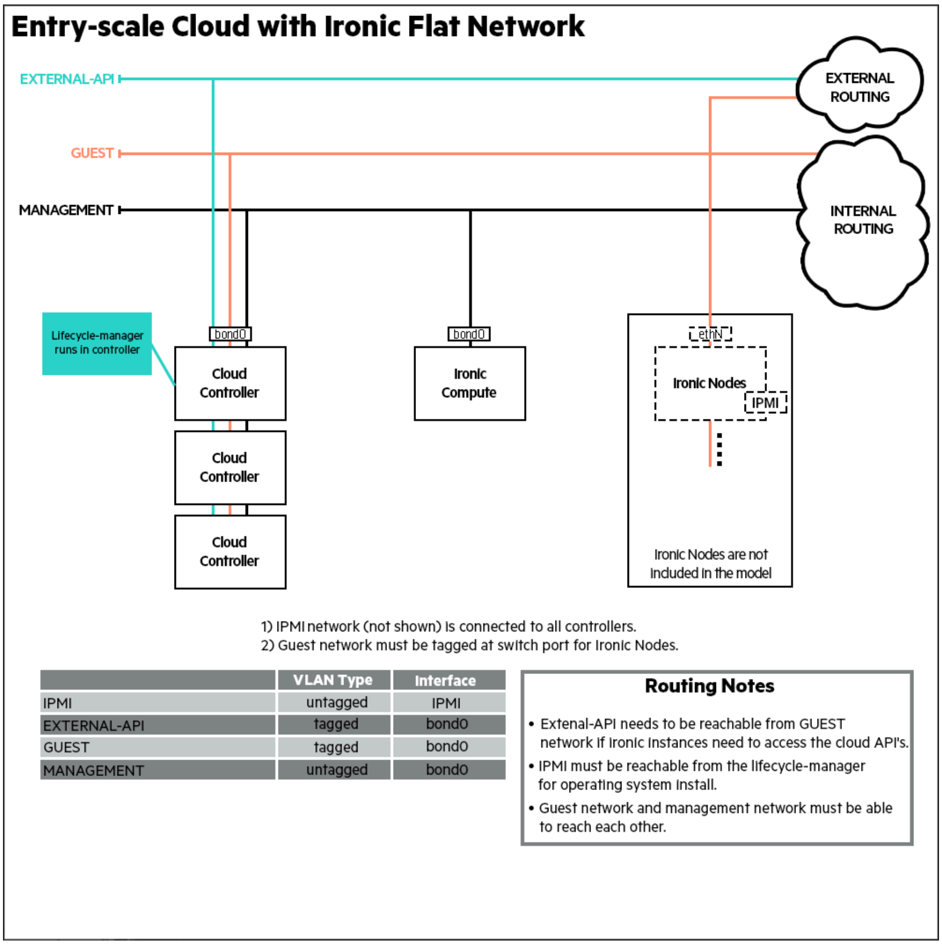
When considering the capacity needs for the Swift proxy, account, and container (PAC) servers, you should calculate 2% of the total raw storage size of your object servers to specify the storage required for the PAC servers. So, for example, if you were using the example we provided earlier and you had an object server setup of 24 hard drives with 8TB each for a total of 192TB and you had a total of 6 object servers, that would give a raw total of 1152TB. So you would take 2% of that, which is 23TB, and ensure that much storage capacity was available on your Swift proxy, account, and container (PAC) server cluster. If you had a cluster of three Swift PAC servers, that would be ~8TB each.

Another general rule of thumb is that if you are expecting to have more than a million objects in a container then you should consider using SSDs on the Swift PAC servers rather than HDDs.

9.6 Ironic Examples

9.6.1 Entry-Scale Cloud with Ironic Flat Network

This example deploys an entry scale cloud that uses the Ironic service to provision physical machines through the Compute services API.



1) IPMI network (not shown) is connected to all controllers.
2) Guest network must be tagged at switch port for Ironic Nodes.

	VLAN Type	Interface
IPMI	untagged	IPMI
EXTERNAL-API	tagged	bond0
GUEST	tagged	bond0
MANAGEMENT	untagged	bond0

Routing Notes

- Extenal-API needs to be reachable from GUEST network if Ironic instances need to access the cloud API's.
- IPMI must be reachable from the lifecycle-manager for operating system install.
- Guest network and management network must be able to reach each other.

Control Plane

Cluster1 3 nodes of type `CONTROLLER-ROLE` run the core OpenStack services, such as Keystone, Nova API, Glance API, Neutron API, Horizon, and Heat API.

Cloud Lifecycle Manager

The Cloud Lifecycle Manager runs on one of the control-plane nodes of type `CONTROLLER-ROLE`. The IP address of the node that will run the Cloud Lifecycle Manager needs to be included in the `data/servers.yml` file.

Resource Nodes

- **Ironic Compute** One node of type `IRONIC-COMPUTE-ROLE` runs nova-compute, nova-compute-ironic, and other supporting services.
- **Object Storage** Minimal Swift resources are provided by the control plane.

Networking

This example requires the following networks:

- **IPMI** network connected to the lifecycle-manager and the IPMI ports of all servers.

Nodes require a pair of bonded NICs which are used by the following networks:

- **External API** This is the network that users will use to make requests to the cloud.
- **Cloud Management** This is the network that will be used for all internal traffic between the cloud services. This network is also used to install and configure the nodes. The network needs to be on an untagged VLAN.
- **Guest** This is the flat network that will carry traffic between bare metal instances within the cloud. It is also used to PXE boot said bare metal instances and install the operating system selected by tenants.

The `EXTERNAL API` network must be reachable from the `GUEST` network for the bare metal instances to make API calls to the cloud.
An example set of networks is defined in `data/networks.yml`. The file needs to be modified to reflect your environment.
The example uses the devices `hed3` and `hed4` as a bonded network interface for all services. The name given to a network interface by the system is configured in the file `data/net_interfaces.yml`. That file needs to be modified to match your system.

Local Storage

All servers should present a single OS disk, protected by a RAID controller. This disk needs to be at least 512 GB in capacity. In addition the example configures one additional disk depending on the role of the server:

- **Controllers** `/dev/sdb` and `/dev/sdc` configured to be used by Swift.

Additional discs can be configured for any of these roles by editing the corresponding `data/disks_*.yml` file.

9.6.2 Entry-Scale Cloud with Ironic Multi-Tenancy

This example deploys an entry scale cloud that uses the Ironic service to provision physical machines through the Compute services API and supports multi tenancy.

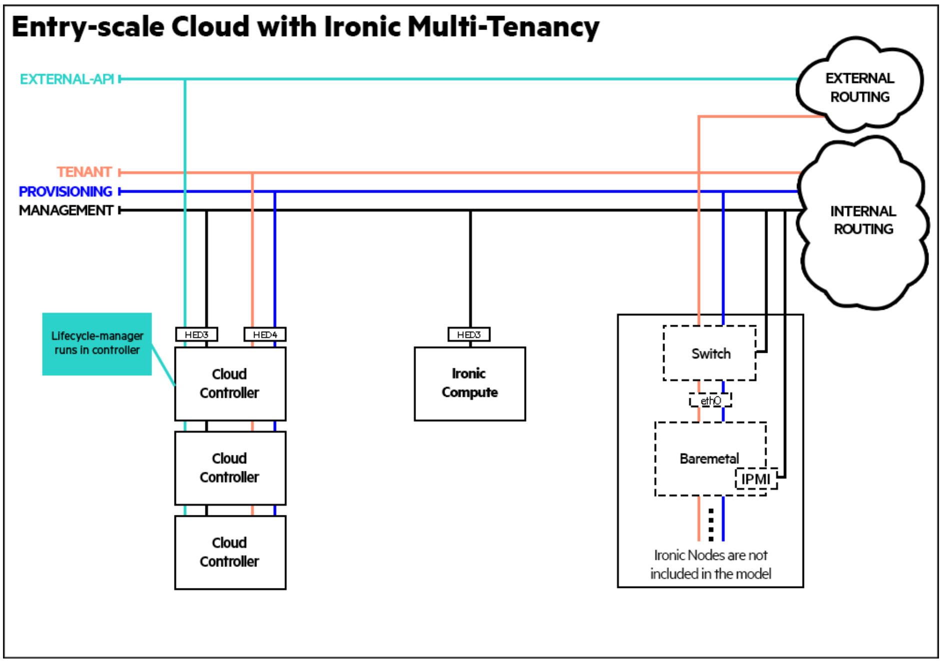


FIGURE 9.1: ENTRY-SCALE CLOUD WITH IRONIC MUTI-TENANCY

Control Plane

Cluster1 3 nodes of type `CONTROLLER-ROLE` run the core OpenStack services, such as Keystone, Nova API, Glance API, Neutron API, Horizon, and Heat API.

Cloud Lifecycle Manager

The Cloud Lifecycle Manager runs on one of the control-plane nodes of type `CONTROLLER-ROLE`. The IP address of the node that will run the Cloud Lifecycle Manager needs to be included in the `data/servers.yml` file.

Resource Nodes

- **Ironic Compute** One node of type `IRONIC-COMPUTE-ROLE` runs nova-compute, nova-compute-ironic, and other supporting services.
- **Object Storage** Minimal Swift Resources are provided by the control plane.

Networking

This example requires the following networks:

- **IPMI** network connected to the deployer and the IPMI ports of all nodes.
- **External API** network is used to make requests to the cloud.

- **Cloud Management** This is the network that will be used for all internal traffic between the cloud services. This network is also used to install and configure the controller nodes. The network needs to be on an untagged VLAN.
- **Provisioning** is the network used to PXE boot the Ironic nodes and install the operating system selected by tenants. This network needs to be tagged on the switch for control plane/Ironic compute nodes. For Ironic bare metal nodes, VLAN configuration on the switch will be set by Neutron driver.
- **Tenant VLANs** The range of VLAN IDs should be reserved for use by Ironic and set in the cloud configuration. It is configured as untagged on control plane nodes, therefore it cannot be combined with management network on the same network interface.

The following access should be allowed by routing/firewall:

- Access from Management network to IPMI. Used during cloud installation and during Ironic bare metal node provisioning.
- Access from Management network to switch management network. Used by neutron driver.
- The `EXTERNAL API` network must be reachable from the tenant networks if you want bare metal nodes to be able to make API calls to the cloud.

An example set of networks is defined in `data/networks.yml`. The file needs to be modified to reflect your environment.
The example uses `hed3` for Management and External API traffic, and `hed4` for provisioning and tenant network traffic. If you need to modify these assignments for your environment, they are defined in `data/net_interfaces.yml`.

Local Storage

All servers should present a single OS disk, protected by a RAID controller. This disk needs to be at least 512 GB in capacity. In addition the example configures one additional disk depending on the role of the server:

- **Controllers** `/dev/sdb` and `/dev/sdc` configured to be used by Swift.

Additional disks can be configured for any of these roles by editing the corresponding `data/disks_*.yml` file.

10 Modifying Example Configurations for Compute Nodes

This section contains detailed information about the Compute Node parts of the input model. For example input models, see *Chapter 9, Example Configurations*. For general descriptions of the input model, see *Section 6.14, “Networks”*.

Usually, the example models provide most of the data that is required to create a valid input model. However, before you start to deploy, you may want to customize an input model using the following information about Compute Nodes.

10.1 SLES Compute Nodes

`net_interfaces.yml`

```
- name: SLES-COMPUTE-INTERFACES
 network-interfaces:
   - name: BOND0
     device:
         name: bond0
     bond-data:
         options:
             mode: active-backup
             miimon: 200
             primary: hed1
         provider: linux
         devices:
             - name: hed1
             - name: hed2
     network-groups:
       - EXTERNAL-VM
       - GUEST
       - MANAGEMENT
```

`servers.yml`

```
    - id: compute1
      ip-addr: 10.13.111.15
      role: SLES-COMPUTE-ROLE
      server-group: RACK1
      nic-mapping: DL360p_G8_2Port
      mac-addr: ec:b1:d7:77:d0:b0
```

```
   ilo-ip: 10.12.13.14
   ilo-password: *********
   ilo-user: Administrator
   distro-id: sles12sp3-x86_64
```

server_roles.yml

```
- name: SLES-COMPUTE-ROLE
  interface-model: SLES-COMPUTE-INTERFACES
  disk-model: SLES-COMPUTE-DISKS
```

disk_compute.yml

```
 - name: SLES-COMPUTE-DISKS
  volume-groups:
   - name: ardana-vg
    physical-volumes:
     - /dev/sda_root

    logical-volumes:
    # The policy is not to consume 100% of the space of each volume group.
    # 5% should be left free for snapshots and to allow for some flexibility.
     - name: root
      size: 35%
      fstype: ext4
      mount: /
     - name: log
      size: 50%
      mount: /var/log
      fstype: ext4
      mkfs-opts: -O large_file
     - name: crash
      size: 10%
      mount: /var/crash
      fstype: ext4
      mkfs-opts: -O large_file

   - name: vg-comp
    # this VG is dedicated to Nova Compute to keep VM IOPS off the OS disk
    physical-volumes:
     - /dev/sdb
    logical-volumes:
     - name: compute
      size: 95%
      mount: /var/lib/nova
      fstype: ext4
      mkfs-opts: -O large_file
```

`control_plane.yml`

```
 control-planes:
  - name: control-plane-1
   control-plane-prefix: cp1
   region-name: region0
....
   resources:
    - name: sles-compute
     resource-prefix: sles-comp
     server-role: SLES-COMPUTE-ROLE
     allocation-policy: any
     min-count: 1
     service-components:
      - ntp-client
      - nova-compute
      - nova-compute-kvm
      - neutron-l3-agent
      - neutron-metadata-agent
      - neutron-openvswitch-agent
      - neutron-lbaasv2-agent
```

11 Modifying Example Configurations for Object Storage using Swift

This section contains detailed descriptions about the Swift-specific parts of the input model. For example input models, see *Chapter 9, Example Configurations*. For general descriptions of the input model, see *Section 6.14, "Networks"*. In addition, the Swift ring specifications are available in the `~/openstack/my_cloud/definition/data/swift/rings.yml` file.

Usually, the example models provide most of the data that is required to create a valid input model. However, before you start to deploy, you must do the following:

- Check the disk model used by your nodes and that all disk drives are correctly named and used as described in *Section 11.6, "Swift Requirements for Device Group Drives"*.
- Select an appropriate partition power for your rings. For more information, see *Section 11.10, "Understanding Swift Ring Specifications"*.

For further information, read these related pages:

11.1 Object Storage using Swift Overview

11.1.1 What is the Object Storage (Swift) Service?

The SUSE OpenStack Cloud Object Storage using Swift service leverages Swift which uses software-defined storage (SDS) layered on top of industry-standard servers using native storage devices. Swift presents an object paradigm, using an underlying set of disk drives. The disk drives are managed by a data structure called a "ring" and you can store, retrieve, and delete objects in containers using RESTful APIs.

SUSE OpenStack Cloud Object Storage using Swift provides a highly-available, resilient, and scalable storage pool for unstructured data. It has a highly-durable architecture, with no single point of failure. In addition, SUSE OpenStack Cloud includes the concept of cloud models, where the user can modify the cloud input model to provide the configuration required for their environment.

11.1.2 Object Storage (Swift) Services

A Swift system consists of a number of services:

- Swift-proxy provides the API for all requests to the Swift system.
- Account and container services provide storage management of the accounts and containers.
- Object services provide storage management for object storage.

These services can be co-located in a number of ways. The following general pattern exists in the example cloud models distributed in SUSE OpenStack Cloud:

- The swift-proxy, account, container, and object services run on the same (PACO) node type in the control plane. This is used for smaller clouds or where Swift is a minor element in a larger cloud. This is the model seen in most of the entry-scale models.
- The swift-proxy, account, and container services run on one (PAC) node type in a cluster in a control plane and the object services run on another (OBJ) node type in a resource pool. This deployment model, known as the Entry-Scale Swift model, is used in larger clouds or where a larger Swift system is in use or planned. See *Section 9.5.1, "Entry-scale Swift Model"* for more details.

The Swift storage service can be scaled both vertically (nodes with larger or more disks) and horizontally (more Swift storage nodes) to handle an increased number of simultaneous user connections and provide larger storage space.

Swift is configured through a number of YAML files in the SUSE OpenStack Cloud implementation of the OpenStack Object Storage (Swift) service. For more details on the configuration of the YAML files, see *Chapter 11, Modifying Example Configurations for Object Storage using Swift*.

11.2 Allocating Proxy, Account, and Container (PAC) Servers for Object Storage

A Swift proxy, account, and container (PAC) server is a node that runs the swift-proxy, swift-account and swift-container services. It is used to respond to API requests and to store account and container data. The PAC node does not store object data.

This section describes the procedure to allocate PAC servers during the **initial** deployment of the system.

11.2.1 To Allocate Swift PAC servers

Perform the following steps to allocate PAC servers:

- Verify if the example input model already contains a suitable server role. The server roles are usually described in the `data/server_roles.yml` file. If the server role is not described, you must add a suitable server role and allocate drives to store object data. For instructions, see *Section 11.4, "Creating Roles for Swift Nodes"* and *Section 11.5, "Allocating Disk Drives for Object Storage"*.
- Verify if the example input model has assigned a cluster to Swift proxy, account, container servers. It is usually mentioned in the `data/control_plane.yml` file. If the cluster is not assigned, then add a suitable cluster. For instructions, see *Section 11.7, "Creating a Swift Proxy, Account, and Container (PAC) Cluster"*.
- Identify the physical servers and their IP address and other detailed information.
  - You add these details to the servers list (usually in the `data/servers.yml` file).
  - As with all servers, you must also verify and/or modify the server-groups information (usually in `data/server_groups.yml`)

The only part of this process that is unique to Swift is the allocation of disk drives for use by the account and container rings. For instructions, see *Section 11.5, "Allocating Disk Drives for Object Storage"*.

11.3 Allocating Object Servers

A Swift object server is a node that runs the swift-object service (**only**) and is used to store object data. It does not run the swift-proxy, swift-account, or swift-container services.

This section describes the procedure to allocate a Swift object server during the **initial** deployment of the system.

11.3.1 To Allocate a Swift Object Server

Perform the following steps to allocate one or more Swift object servers:

- Verify if the example input model already contains a suitable server role. The server roles are usually described in the `data/server_roles.yml` file. If the server role is not described, you must add a suitable server role. For instructions, see *Section 11.4, "Creating Roles for Swift Nodes"*. While adding a server role for the Swift object server, you will also allocate drives to store object data. For instructions, see *Section 11.5, "Allocating Disk Drives for Object Storage"*.
- Verify if the example input model has a resource node assigned to Swift object servers. The resource nodes are usually assigned in the `data/control_plane.yml` file. If it is not assigned, you must add a suitable resource node. For instructions, see *Section 11.8, "Creating Object Server Resource Nodes"*.
- Identify the physical servers and their IP address and other detailed information. Add the details for the servers in either of the following YAML files and verify the server-groups information:
  - Add details in the servers list (usually in the `data/servers.yml` file).
  - As with all servers, you must also verify and/or modify the server-groups information (usually in the `data/server_groups.yml` file).

  The only part of this process that is unique to Swift is the allocation of disk drives for use by the object ring. For instructions, see *Section 11.5, "Allocating Disk Drives for Object Storage"*.

11.4 Creating Roles for Swift Nodes

To create roles for Swift nodes, you must edit the `data/server_roles.yml` file and add an entry to the server-roles list using the following syntax:

```
server-roles:
- name: PICK-A-NAME
  interface-model: SPECIFY-A-NAME
  disk-model: SPECIFY-A-NAME
```

The fields for server roles are defined as follows:

`name`	Specifies a name assigned for the role. In the following example, **SWOBJ-ROLE** is the role name.
`interface-model`	You can either select an existing interface model or create one specifically for Swift object servers. In the following example **SWOBJ-INTERFACES** is used. For more information, see *Section 11.9, "Understanding Swift Network and Service Requirements"*.
`disk-model`	You can either select an existing model or create one specifically for Swift object servers. In the following example **SWOBJ-DISKS** is used. For more information, see *Section 11.5, "Allocating Disk Drives for Object Storage"*.

```
server-roles:
- name: SWOBJ-ROLE
  interface-model: SWOBJ-INTERFACES
  disk-model: SWOBJ-DISKS
```

11.5 Allocating Disk Drives for Object Storage

The disk model describes the configuration of disk drives and their usage. The examples include several disk models. You must always review the disk devices before making any changes to the existing the disk model.

11.5.1 Making Changes to a Swift Disk Model

There are several reasons for changing the disk model:

- If you have additional drives available, you can add them to the devices list.
- If the disk devices listed in the example disk model have different names on your servers. This may be due to different hardware drives. Edit the disk model and change the device names to the correct names.
- If you prefer a different disk drive than the one listed in the model. For example, if `/dev/sdb` and `/dev/sdc` are slow hard drives and you have SDD drives available in `/dev/sdd` and `/dev/sde`. In this case, delete `/dev/sdb` and `/dev/sdc` and replace them with `/dev/sdd` and `/dev/sde`.

Note

Disk drives must not contain labels or file systems from a prior usage. For more information, see *Section 11.6, “Swift Requirements for Device Group Drives”*.

Tip

The terms **add** and **delete** in the document means editing the respective YAML files to add or delete the configurations/values.

Swift Consumer Syntax

The consumer field determines the usage of a disk drive or logical volume by Swift. The syntax of the consumer field is as follows:

```
consumer:
    name: swift
    attrs:
        rings:
        - name: RING-NAME
        - name: RING-NAME
        - etc...
```

The fields for consumer are defined as follows:

`name`	Specifies the service that uses the device group. A `name` field containing **swift** indicates that the drives or logical volumes are used by Swift.
`attrs`	Lists the rings that the devices are allocated to. It must contain a `rings` item.
`rings`	Contains a list of ring names. In the `rings` list, the `name` field is optional.

The following are the different configurations (patterns) of the proxy, account, container, and object services:

- Proxy, account, container, and object (PACO) run on same node type.
- Proxy, account, and container run on a node type (PAC) and the object services run on a dedicated object server (OBJ).

Note

The proxy service does not have any rings associated with it.

EXAMPLE 11.1: **PACO** - PROXY, ACCOUNT, CONTAINER, AND OBJECT RUN ON THE SAME NODE TYPE.

```
consumer:
    name: swift
    attrs:
        rings:
        - name: account
        - name: container
        - name: object-0
```

EXAMPLE 11.2: **PAC** - PROXY, ACCOUNT, AND CONTAINER RUN ON THE SAME NODE TYPE.

```
consumer:
    name: swift
    attrs:
        rings:
        - name: account
        - name: container
```

EXAMPLE 11.3: **OBJ** - DEDICATED OBJECT SERVER

> The following example shows two Storage Policies (object-0 and object-1). For more information, see *Section 11.11, “Designing Storage Policies”*.
>
> ```
> consumer:
>     name: swift
>     attrs:
>         rings:
>         - name: object-0
>         - name: object-1
> ```

Swift Device Groups

You may have several device groups if you have several different uses for different sets of drives.

The following example shows a configuration where one drive is used for account and container rings and the other drives are used by the object-0 ring:

```
device-groups:

- name: swiftpac
  devices:
  - name: /dev/sdb
  consumer:
      name: swift
      attrs:
      - name: account
      - name: container
  - name: swiftobj
    devices:
    - name: /dev/sdc
    - name: /dev/sde
    - name: /dev/sdf
    consumer:
       name: swift
       attrs:
           rings:
              - name: object-0
```

Swift Logical Volumes

Warning

Be careful while using logical volumes to store Swift data. The data remains intact during an upgrade, but will be lost if the server is reimaged. If you use logical volumes you must ensure that you only reimage one server at a time. This is to allow the data from the other replicas to be replicated back to the logical volume once the reimage is complete.

Swift can use a logical volume. To do this, ensure you meet the requirements listed in the table below:

• `mount` • `mkfs-opts` • `fstype`	Do not specify these attributes.
• `name` • `size`	Specify both of these attributes.
• `consumer`	This attribute must have a `name` field set to **swift**.

Note

When setting up Swift as a logical volume, the configuration processor will give a warning. This warning is normal and does not affect the configuration.

Following is an example of Swift logical volumes:

```
...
   - name: swift
     size: 50%
     consumer:
         name: swift
         attrs:
             rings:
             - name: object-0
```

```
            - name: object-1
```

11.6 Swift Requirements for Device Group Drives

To install and deploy, Swift requires that the disk drives listed in the devices list of the device-groups item in a disk model meet the following criteria (if not, the deployment will fail):

- The disk device must exist on the server. For example, if you add `/dev/sdX` to a server with only three devices, then the deploy process will fail.
- The disk device must be unpartitioned or have a single partition that uses the whole drive.
- The partition must not be labeled.
- The XFS file system must not contain a file system label.
- If the disk drive is already labeled as described above, the `swiftlm-drive-provision` process will assume that the drive has valuable data and will not use or modify the drive.

11.7 Creating a Swift Proxy, Account, and Container (PAC) Cluster

If you already have a cluster with the server-role `SWPAC-ROLE` there is no need to proceed through these steps.

11.7.1 Steps to Create a Swift Proxy, Account, and Container (PAC) Cluster

To create a cluster for Swift proxy, account, and container (PAC) servers, you must identify the control plane and node type/role:

1. In the `~/openstack/my_cloud/definition/data/control_plane.yml` file, identify the control plane that the PAC servers are associated with.
2. Next, identify the node type/role used by the Swift PAC servers. In the following example, `server-role` is set to **SWPAC-ROLE**.

Add an entry to the `clusters` item in the `control-plane` section.
Example:

```
control-planes:
    - name: control-plane-1
      control-plane-prefix: cp1

  . . .
  clusters:
  . . .
     - name: swpac1
       cluster-prefix: c2
       server-role: SWPAC-ROLE
       member-count: 3
       allocation-policy: strict
       service-components:
         - ntp-client
         - swift-ring-builder
         - swift-proxy
         - swift-account
         - swift-container
         - swift-client
```

Important

Do not change the name of the cluster `swpac` as it may conflict with an existing cluster. Use a name such as `swpac1`, `swpac2`, or `swpac3`.

3. If you have more than three servers available that have the `SWPAC-ROLE` assigned to them, you must change `member-count` to match the number of servers.
For example, if you have four servers with a role of `SWPAC-ROLE`, then the `member-count` should be 4.

11.7.2 Service Components

A Swift PAC server requires the following service components:

- ntp-client
- swift-proxy
- swift-account

- swift-container
- swift-ring-builder
- swift-client

11.8 Creating Object Server Resource Nodes

To create a resource node for Swift object servers, you must identify the control plane and node type/role:

- In the `data/control_plane.yml` file, identify the control plane that the object servers are associated with.
- Next, identify the node type/role used by the Swift object servers. In the following example, `server-role` is set to **SWOBJ-ROLE**:
  Add an entry to the `resources` item in the **control-plane**:

```
control-planes:
    - name: control-plane-1
      control-plane-prefix: cp1
      region-name: region1
  . . .
  resources:
  . . .
  - name: swobj
    resource-prefix: swobj
    server-role: SWOBJ-ROLE
    allocation-policy: strict
    min-count: 0
    service-components:
    - ntp-client
    - swift-object
```

Service Components

A Swift object server requires the following service components:

- `ntp-client`
- `swift-object`
- `swift-client` is optional; installs the `python-swiftclient` package on the server.

Resource nodes do not have a member count attribute. So the number of servers allocated with the **SWOBJ-ROLE** is the number of servers in the `data/servers.yml` file with a server role of **SWOBJ-ROLE**.

11.9 Understanding Swift Network and Service Requirements

This topic describes Swift's requirements for which service components must exist in the input model and how these relate to the network model. This information is useful if you are creating a cluster or resource node, or when defining the networks used by Swift. The network model allows many options and configurations. For smooth Swift operation, the following must be **true**:

- The following services must have a **direct** connection to the same network:
  - `swift-proxy`
  - `swift-account`
  - `swift-container`
  - `swift-object`
  - `swift-ring-builder`
- The `swift-proxy` service must have a **direct** connection to the same network as the `cluster-ip` service.
- The memcached service must be configured on a cluster of the control plane. In small deployments, it is convenient to run it on the same cluster as the horizon service. For larger deployments, with many nodes running the `swift-proxy` service, it is better to **co-locate** the `swift-proxy` and `memcached` services. The `swift-proxy` and `swift-container` services must have a **direct** connection to the same network as the `memcached` service.
- The `swift-proxy` and `swift-ring-builder` service must be **co-located** in the same cluster of the control plane.
- The `ntp-client` service must be **present** on all Swift nodes.

11.10 Understanding Swift Ring Specifications

In Swift, the ring is responsible for mapping data on particular disks. There is a separate ring for account databases, container databases, and each object storage policy, but each ring works similarly. The `swift-ring-builder` utility is used to build and manage rings. This utility uses a builder file to contain ring information and additional data required to build future rings. In SUSE OpenStack Cloud 8, you will use the cloud model to specify how the rings are configured and used. This model is used to automatically invoke the `swift-ring-builder` utility as part of the deploy process. (Normally, you will not run the `swift-ring-builder` utility directly.)

The rings are specified in the input model using the **configuration-data** key. The `configuration-data` in the `control-planes` definition is given a name that you will then use in the `swift_config.yml` file. If you have several control planes hosting Swift services, the ring specifications can use a shared `configuration-data` object, however it is considered best practice to give each Swift instance its own `configuration-data` object.

11.10.1 Ring Specifications in the Input Model

In most models, the ring-specification is mentioned in the `~/openstack/my_cloud/definition/data/swift/swift_config.yml` file. For example:

```
configuration-data:
  - name: SWIFT-CONFIG-CP1
    services:
      - swift
    data:
      control_plane_rings:
        swift-zones:
          - id: 1
            server-groups:
              - AZ1
          - id: 2
            server-groups:
              - AZ2
          - id: 3
            server-groups:
              - AZ3
        rings:
          - name: account
            display-name: Account Ring
            min-part-hours: 16
            partition-power: 12
            replication-policy:
              replica-count: 3

          - name: container
            display-name: Container Ring
            min-part-hours: 16
            partition-power: 12
            replication-policy:
              replica-count: 3

          - name: object-0
            display-name: General
            default: yes
            min-part-hours: 16
            partition-power: 12
            replication-policy:
              replica-count: 3
```

The above sample file shows that the rings are specified using the `configuration-data` object **SWIFT-CONFIG-CP1** and has three rings as follows:

- **Account ring**: You must always specify a ring called **account**. The account ring is used by Swift to store metadata about the projects in your system. In Swift, a Keystone project maps to a Swift account. The `display-name` is informational and not used.

- **Container ring**:You must always specify a ring called **container**. The `display-name` is informational and not used.

- **Object ring**: This ring is also known as a storage policy. You must always specify a ring called **object-0**. It is possible to have multiple object rings, which is known as *storage policies*. The `display-name` is the name of the storage policy and can be used by users of the Swift system when they create containers. It allows them to specify the storage policy that the container uses. In the example, the storage policy is called **General**. There are also two aliases for the storage policy name: `GeneralPolicy` and `AnotherAliasForGeneral`. In this example, you can use `General`, `GeneralPolicy`, or `AnotherAliasForGeneral` to refer to this storage policy. The aliases item is optional. The `display-name` is required.

- **Min-part-hours, partition-power, replication-policy** and **replica-count** are described in the following section.

11.10.2 Replication Ring Parameters

The ring parameters for traditional replication rings are defined as follows:

Parameter	Description
`replica-count`	Defines the number of copies of object created. Use this to control the degree of resiliency or availability. The `replica-count` is normally set to `3` (that means Swift will keep three copies of accounts, containers, or objects). As a best practice, do not set the value below `3`. To achieve higher resiliency, increase the value.
`min-part-hours`	Changes the value used to decide when a given partition can be moved. This is the number of hours that the **`swift-ring-builder`** tool will enforce between ring rebuilds. On a small system, this can be as low as `1` (one hour). The value can be different for each ring. In the example above, the `swift-ring-builder` will enforce a minimum of 16 hours between ring rebuilds. However, this time is system-dependent so you will be unable to determine the appropriate value for `min-part-hours` until you have more experience with your system. A value of `0` (zero) is not allowed. In prior releases, this parameter was called `min-part-time`. The older name is still supported, however do not specify both `min-part-hours` and `min-part-time` in the same files.

Parameter	Description
partition-power	The optimal value for this parameter is related to the number of disk drives that you allocate to Swift storage. As a best practice, you should use the same drives for both the account and container rings. In this case, the partition-power value should be the same. For more information, see *Section 11.10.4, "Selecting a Partition Power"*.
replication-policy	Specifies that a ring uses replicated storage. The duplicate copies of the object are created and stored on different disk drives. All replicas are identical. If one is lost or corrupted, the system automatically copies one of the remaining replicas to restore the missing replica.
default	The default value in the above sample file of ring-specification is set to **yes**, which means that the storage policy is enabled to store objects. For more information, see *Section 11.11, "Designing Storage Policies"*.

11.10.3 Erasure Coded Rings

In the cloud model, a ring-specification is mentioned in the ~/openstack/my_cloud/definition/data/swift/rings.yml file. A typical erasure coded ring in this file looks like this:

```
- name: object-1
  display-name: EC_ring
  default: no
  min-part-hours: 16
  partition-power: 12
  erasure-coding-policy:
    ec-type: jerasure_rs_vand
    ec-num-data-fragments: 10
    ec-num-parity-fragments: 4
    ec-object-segment-size: 1048576
```

The additional parameters are defined as follows:

Parameter	Description
ec-type	This is the particular erasure policy scheme that is being used. The supported ec_types in SUSE OpenStack Cloud 8 are: - `jerasure_rs_vand` => Vandermonde Reed-Solomon encoding, based on Jerasure
erasure-coding-policy	This line indicates that the object ring will be of type "erasure coding"
ec-num-data-fragments	This indicated the number of data fragments for an object in the ring.
ec-num-parity-fragments	This indicated the number of parity fragments for an object in the ring.
ec-object-segment-size	The amount of data that will be buffered up before feeding a segment into the encoder/decoder. The default value is 1048576.

When using an erasure coded ring, the number of devices in the ring must be greater than or equal to the total number of fragments of an object. For example, if you define an erasure coded ring with 10 data fragments and 4 parity fragments, there must be at least 14 (10+4) devices added to the ring.

When using erasure codes, for a PUT object to be successful it must store `ec_ndata + 1` fragment to achieve quorum. Where the number of data fragments (`ec_ndata`) is 10 then at least 11 fragments must be saved for the object PUT to be successful. The 11 fragments must be saved to different drives. To tolerate a single object server going down, say in a system with 3 object servers, each object server must have at least 6 drives assigned to the erasure coded storage policy. So with a single object server down, 12 drives are available between the remaining object servers. This allows an object PUT to save 12 fragments, one more than the minimum to achieve quorum.

Unlike replication rings, none of the erasure coded parameters may be edited after the initial creation. Otherwise there is potential for permanent loss of access to the data.

On the face of it, you would expect that an erasure coded configuration that uses a data to parity ratio of 10:4, that the data consumed storing the object is 1.4 times the size of the object just like the x3 replication takes x3 times the size of the data when storing the object. However, for erasure coding, this 10:4 ratio is not correct. The efficiency (that is how much storage is needed to store the object) is very poor for small objects and improves as the object size grows. However, the improvement is not linear. If all of your files are less than 32K in size, erasure coding will take more space to store than the x3 replication.

11.10.4 Selecting a Partition Power

When storing an object, the object storage system hashes the name. This hash results in a hit on a partition (so a number of different object names result in the same partition number). Generally, the partition is mapped to available disk drives. With a replica count of 3, each partition is mapped to three different disk drives. The hashing algorithm used hashes over a fixed number of partitions. The partition-power attribute determines the number of partitions you have.

Partition power is used to distribute the data uniformly across drives in a Swift nodes. It also defines the storage cluster capacity. You must set the partition power value based on the total amount of storage you expect your entire ring to use.

You should select a partition power for a given ring that is appropriate to the number of disk drives you allocate to the ring for the following reasons:

- If you use a high partition power and have a few disk drives, each disk drive will have thousands of partitions. With too many partitions, audit and other processes in the Object Storage system cannot walk the partitions in a reasonable time and updates will not occur in a timely manner.

- If you use a low partition power and have many disk drives, you will have tens (or maybe only one) partition on a drive. The Object Storage system does not use size when hashing to a partition - it hashes the name.
  With many partitions on a drive, a large partition is cancelled out by a smaller partition so the overall drive usage is similar. However, with very small numbers of partitions, the uneven distribution of sizes can be reflected in uneven disk drive usage (so one drive becomes full while a neighboring drive is empty).

An ideal number of partitions per drive is 100. If you know the number of drives, select a partition power that will give you approximately 100 partitions per drive. Usually, you install a system with a specific number of drives and add drives as needed. However, you cannot change the value of the partition power. Hence you must select a value that is a compromise between current and planned capacity.

Important

If you are installing a small capacity system and you need to grow to a very large capacity but you cannot fit within any of the ranges in the table, please seek help from Sales Engineering to plan your system.

There are additional factors that can help mitigate the fixed nature of the partition power:

- Account and container storage represents a small fraction (typically 1 percent) of your object storage needs. Hence, you can select a smaller partition power (relative to object ring partition power) for the account and container rings.
- For object storage, you can add additional storage policies (that is, another object ring). When you have reached capacity in an existing storage policy, you can add a new storage policy with a higher partition power (because you now have more disk drives in your system). This means that you can install your system using a small partition power appropriate to a small number of initial disk drives. Later, when you have many disk drives, the new storage policy can have a higher value appropriate to the larger number of drives.

However, when you continue to add storage capacity, existing containers will continue to use their original storage policy. Hence, the additional objects must be added to new containers to take advantage of the new storage policy.

Use the following table to select an appropriate partition power for each ring. The partition power of a ring cannot be changed, so it is important to select an appropriate value. This table is based on a replica count of 3. If your replica count is different, or you are unable to find your system in the table, then see *Section 11.10.4, “Selecting a Partition Power”* for information of selecting a partition power.

The table assumes that when you first deploy Swift, you have a small number of drives (the minimum column in the table), and later you add drives.

Note

- Use the total number of drives. For example, if you have three servers, each with two drives, the total number of drives is six.
- The lookup should be done separately for each of the account, container and object rings. Since account and containers represent approximately 1 to 2 percent of object storage, you will probably use fewer drives for the account and container rings (that is, you will have fewer proxy, account, and container (PAC) servers) so that your object rings may have a higher partition power.
- The largest anticipated number of drives imposes a limit in the minimum drives you can have. (For more information, see *Section 11.10.4, “Selecting a Partition Power”*.) This means that, if you anticipate significant growth, your initial system can be small, but under a certain limit. For example, if you determine that the maximum number of drives the system will grow to is 40,000, then use a partition power of 17 as listed in the table below. In addition, a minimum of 36 drives is required to build the smallest system with this partition power.
- The table assumes that disk drives are the same size. The actual size of a drive is not significant.

11.11 Designing Storage Policies

Storage policies enable you to differentiate the way objects are stored.

Reasons to use storage policies include the following:

- Different types or classes of disk drive
  You can use different drives to store various type of data. For example, you can use 7.5K RPM high-capacity drives for one type of data and fast SSD drives for another type of data.
- Different redundancy or availability needs
  You can define the redundancy and availability based on your requirement. You can use a replica count of 3 for "normal" data and a replica count of 4 for "critical" data.
- Growing of cluster capacity

If the storage cluster capacity grows beyond the recommended partition power as described in *Section 11.10, “Understanding Swift Ring Specifications”*.

- Erasure-coded storage and replicated storage
  If you use erasure-coded storage for some objects and replicated storage for other objects.

Storage policies are implemented on a per-container basis. If you want a non-default storage policy to be used for a new container, you can explicitly specify the storage policy to use when you create the container. You can change which storage policy is the default. However, this does not affect existing containers. Once the storage policy of a container is set, the policy for that container cannot be changed.

The disk drives used by storage policies can overlap or be distinct. If the storage policies overlap (that is, have disks in common between two storage policies), it is recommended to use the same set of disk drives for both policies. But in the case where there is a partial overlap in disk drives, because one storage policy receives many objects, the drives that are common to both policies must store more objects than drives that are only allocated to one storage policy. This can be appropriate for a situation where the overlapped disk drives are larger than the non-overlapped drives.

11.11.1 Specifying Storage Policies

There are two places where storage policies are specified in the input model:

- The attribute of the storage policy is specified in ring-specification in the `data/swift/rings.yml` file.
- When associating disk drives with specific rings in a disk model. This specifies which drives and nodes use the storage policy. In other word words, where data associated with a storage policy is stored.

A storage policy is specified similar to other rings. However, the following features are unique to storage policies:

- Storage policies are applicable to object rings only. The account or container rings cannot have storage policies.
- There is a format for the ring name: object-*`index`*, where index is a number in the range 0 to 9 (in this release). For example: object-0.

- The object-0 ring must always be specified.
- Once a storage policy is deployed, it should never be deleted. You can remove all disk drives for the storage policy, however the ring specification itself cannot be deleted.
- You can use the `display-name` attribute when creating a container to indicate which storage policy you want to use for that container.
- One of the storage policies can be the default policy. If you do not specify the storage policy then the object created in new container uses the default storage policy.
- If you change the default, only containers created later will have that changed default policy.

The following example shows three storage policies in use. Note that the third storage policy example is an erasure coded ring.

```
rings:
. . .
- name: object-0
  display-name: General
  default: no
  min-part-hours: 16
  partition-power: 12
  replication-policy:
      replica-count: 3
- name: object-1
  display-name: Data
  default: yes
  min-part-hours: 16
  partition-power: 20
  replication-policy:
      replica-count: 3
- name: object-2
  display-name: Archive
  default: no
  min-part-hours: 16
  partition-power: 20
  erasure-coded-policy:
    ec-type: jerasure_rs_vand
    ec-num-data-fragments: 10
    ec-num-parity-fragments: 4
    ec-object-segment-size: 1048576
```

11.12 Designing Swift Zones

The concept of Swift zones allows you to control the placement of replicas on different groups of servers. When constructing rings and allocating replicas to specific disk drives, Swift will, where possible, allocate replicas using the following hierarchy so that the greatest amount of resiliency is achieved by avoiding single points of failure:

- Swift will place each replica on a different disk drive within the same server.
- Swift will place each replica on a different server.
- Swift will place each replica in a different Swift zone.

If you have three servers and a replica count of three, it is easy for Swift to place each replica on a different server. If you only have two servers though, Swift will place two replicas on one server (different drives on the server) and one copy on the other server.

With only three servers there is no need to use the Swift zone concept. However, if you have more servers than your replica count, the Swift zone concept can be used to control the degree of resiliency. The following table shows how data is placed and explains what happens under various failure scenarios. In all cases, a replica count of three is assumed and that there are a total of six servers.

Number of Swift Zones	Replica Placement	Failure Scenarios	Details
One (all servers in the same zone)	Replicas are placed on different servers. For any given object, you have no control over which servers the replicas are placed on.	One server fails	You are guaranteed that there are two other replicas.
		Two servers fail	You are guaranteed that there is one remaining replica.
		Three servers fail	1/3 of the objects cannot be accessed. 2/3 of the objects have three replicas.
Two (three servers in each Swift zone)	Half the objects have two replicas in Swift	One Swift zone fails	You are guaranteed to have at least one

Number of Swift Zones	Replica Placement	Failure Scenarios	Details
	zone 1 with one replica in Swift zone The other objects are reversed, with one replica in Swift zone 1 and two replicas in Swift zone 2.		replica. Half the objects have two remaining replicas and the other half have a single replica.
Three (two servers in each Swift zone)	Each zone contains a replica. For any given object, there is a replica in each Swift zone.	One Swift zone fails	You are guaranteed to have two replicas of every object.
		Two Swift zones fail	You are guaranteed to have one replica of every object.

The following sections show examples of how to specify the Swift zones in your input model.

11.12.1 Using Server Groups to Specify Swift Zones

Swift zones are specified in the ring specifications using the server group concept. To define a Swift zone, you specify:

- An id - this is the Swift zone number
- A list of associated server groups

Server groups are defined in your input model. The example input models typically define a number of server groups. You can use these pre-defined server groups or create your own.

For example, the following three models use the example server groups `CLOUD`, `AZ1`, `AZ2` and `AZ3`. Each of these examples achieves the same effect – creating a single Swift zone.

```
ring-specifications:
              - region: region1
              swift-zones:
              - id: 1
              server-groups:
```

```
              - CLOUD
              rings:
              …
```

```
ring-specifications:
              - region: region1
              swift-zones:
              - id: 1
              server-groups:
              - AZ1
              - AZ2
              - AZ3
              rings:
              …
```

```
server-groups:
              - name: ZONE_ONE
              server-groups:
              - AZ1
              - AZ2
              - AZ3
              ring-specifications:
              - region: region1
              swift-zones:
              - id: 1
              server-groups:
              - ZONE_ONE
              rings:
              …
```

Alternatively, if you omit the `swift-zones` specification, a single Swift zone is used by default for all servers.

In the following example, three Swift zones are specified and mapped to the same availability zones that Nova uses (assuming you are using one of the example input models):

```
ring-specifications:
      - region: region1
      swift-zones:
      - id: 1
      server-groups:
      - AZ1
      - id: 2
      server-groups:
      - AZ2
      - id: 3
      server-groups:
```

```
      - AZ3
```

In this example, it shows a datacenter with four availability zones which are mapped to two Swift zones. This type of setup may be used if you had two buildings where each building has a duplicated network infrastructure:

```
ring-specifications:
      - region: region1
      swift-zones:
      - id: 1
      server-groups:
      - AZ1
      - AZ2
      - id: 2
      server-groups:
      - AZ3
      - AZ4
```

11.12.2 Specifying Swift Zones at Ring Level

Usually, you would use the same Swift zone layout for all rings in your system. However, it is possible to specify a different layout for a given ring. The following example shows that the account, container and object-0 rings have two zones, but the object-1 ring has a single zone.

```
ring-specifications:
        - region: region1
        swift-zones:
        - id: 1
        server-groups:
        - AZ1
        - id: 2
        server-groups:
        - AZ2
        rings
        - name: account
        …
        - name: container
        …
        - name: object-0
        …
        - name: object-1
        swift-zones:
        - id: 1
        server-groups:
```

```
        - CLOUD
        …
```

11.13 Customizing Swift Service Configuration Files

SUSE OpenStack Cloud 8 enables you to modify various Swift service configuration files. The following Swift service configuration files are located on the Cloud Lifecycle Manager in the `~/openstack/my_cloud/config/swift/` directory:

- `account-server.conf.j2`
- `container-reconciler.conf.j2`
- `container-server.conf.j2`
- `container-sync-realms.conf.j2`
- `object-expirer.conf.j2`
- `object-server.conf.j2`
- `proxy-server.conf.j2`
- `rsyncd.conf.j2`
- `swift.conf.j2`
- `swift-recon.j2`

There are many configuration options that can be set or changed, including **container rate limit** and **logging level**:

11.13.1 Configuring Swift Container Rate Limit

The Swift container rate limit allows you to limit the number of `PUT` and `DELETE` requests of an object based on the number of objects in a container. For example, suppose the `container_ratelimit_x = r` . It means that for containers of size `x`, limit requests per second to `r`.

To enable container rate limiting:

1. Log in to the Cloud Lifecycle Manager.

2. Edit the DEFAULT section of ~/openstack/my_cloud/config/swift/proxy-server.conf.j2:

   ```
   container_ratelimit_0 = 100
   container_ratelimit_1000000 = 100
   container_ratelimit_5000000 = 50
   ```

   This will set the PUT and DELETE object rate limit to 100 requests per second for containers with up to 1,000,000 objects. Also, the PUT and DELETE rate for containers with between 1,000,000 and 5,000,000 objects will vary linearly from between 100 and 50 requests per second as the container object count increases.

3. Commit your changes to git:

   ```
   ardana > cd ~/openstack/ardana/ansible
   ardana > git commit -m "COMMIT_MESSAGE" \
   ~/openstack/my_cloud/config/swift/proxy-server.conf.j2
   ```

4. Run the configuration processor:

   ```
   ardana > cd ~/openstack/ardana/ansible
   ardana > ansible-playbook -i hosts/localhost config-processor-run.yml
   ```

5. Create a deployment directory:

   ```
   ardana > cd ~/openstack/ardana/ansible
   ardana > ansible-playbook -i hosts/localhost ready-deployment.yml
   ```

6. Run the swift-reconfigure.yml playbook to reconfigure the Swift servers:

   ```
   ardana > cd ~/scratch/ansible/next/ardana/ansible
   ardana > ansible-playbook -i hosts/verb_hosts swift-reconfigure.yml
   ```

11.13.2 Configuring Swift Account Server Logging Level

By default the Swift logging level is set to INFO. As a best practice, do not set the log level to DEBUG for a long period of time. Use it for troubleshooting issues and then change it back to INFO.

Perform the following steps to set the logging level of the account-server to DEBUG:

1. Log in to the Cloud Lifecycle Manager.

2. Edit the `DEFAULT` section of `~/openstack/my_cloud/config/swift/account-server.conf.j2`:

   ```
   [DEFAULT] . . log_level = DEBUG
   ```

3. Commit your changes to git:

   ```
   ardana > cd ~/openstack/ardana/ansible
   ardana > git commit -m "COMMIT_MESSAGE" \
   ~/openstack/my_cloud/config/swift/account-server.conf.j2
   ```

4. Run the configuration processor:

   ```
   ardana > cd ~/openstack/ardana/ansible
   ardana > ansible-playbook -i hosts/localhost config-processor-run.yml
   ```

5. Create a deployment directory:

   ```
   ardana > cd ~/openstack/ardana/ansible
   ardana > ansible-playbook -i hosts/localhost ready-deployment.yml
   ```

6. Run the `swift-reconfigure.yml` playbook to reconfigure the Swift servers:

   ```
   ardana > cd ~/scratch/ansible/next/ardana/ansible
   ardana > ansible-playbook -i hosts/verb_hosts swift-reconfigure.yml
   ```

11.13.3 For More Information

For more information, see:

- *Book “Operations Guide”, Chapter 12 “Managing Monitoring, Logging, and Usage Reporting”, Section 12.2 “Centralized Logging Service”, Section 12.2.5 “Configuring Centralized Logging”*
- *Book “Operations Guide”, Chapter 12 “Managing Monitoring, Logging, and Usage Reporting”, Section 12.2 “Centralized Logging Service”*

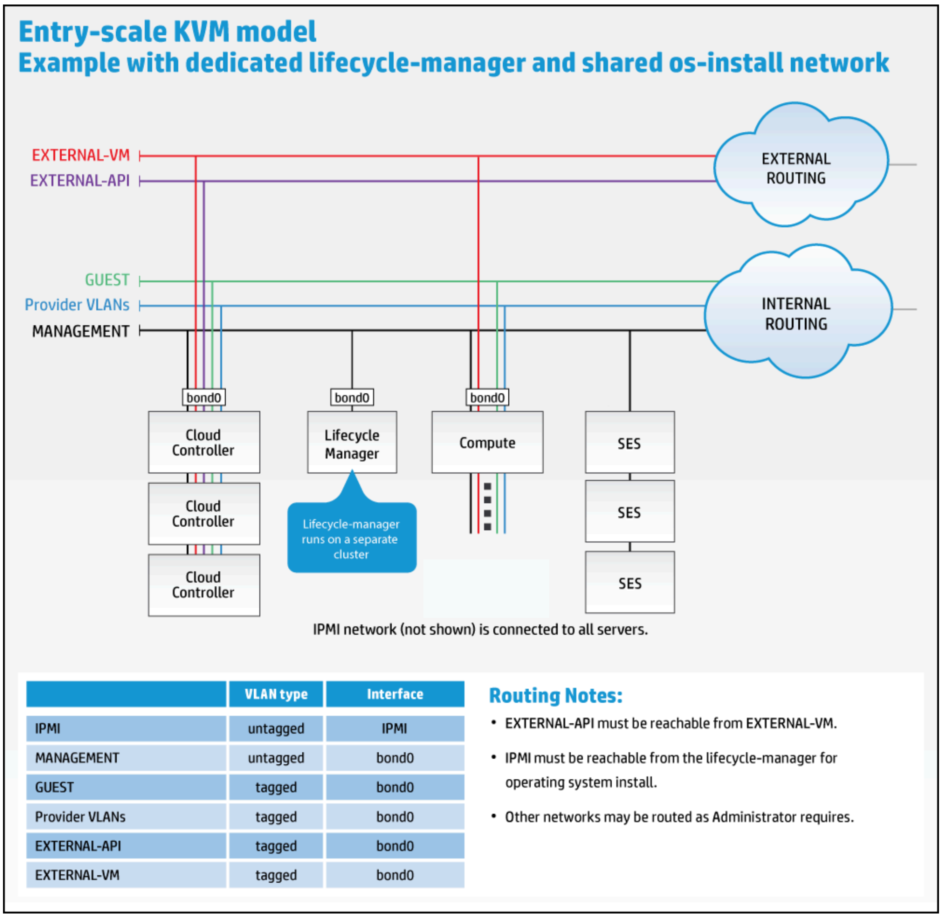
12 Alternative Configurations

In SUSE OpenStack Cloud 8 there are alternative configurations that we recommend for specific purposes.

12.1 Using a Dedicated Cloud Lifecycle Manager Node

All of the example configurations included host the Cloud Lifecycle Manager on the first Control Node. It is also possible to deploy this service on a dedicated node. One use case for wanting to run the dedicated Cloud Lifecycle Manager is to be able to test the deployment of different configurations without having to re-install the first server. Some administrators prefer the additional security of keeping all of the configuration data on a separate server from those that users of the cloud connect to (although all of the data can be encrypted and SSH keys can be password protected).

Here is a graphical representation of this setup:



	VLAN type	Interface
IPMI	untagged	IPMI
MANAGEMENT	untagged	bond0
GUEST	tagged	bond0
Provider VLANs	tagged	bond0
EXTERNAL-API	tagged	bond0
EXTERNAL-VM	tagged	bond0

Routing Notes:

- EXTERNAL-API must be reachable from EXTERNAL-VM.
- IPMI must be reachable from the lifecycle-manager for operating system install.
- Other networks may be routed as Administrator requires.

12.1.1 Specifying a dedicated Cloud Lifecycle Manager in your input model

To specify a dedicated Cloud Lifecycle Manager in your input model, make the following edits to your configuration files.

Important

The indentation of each of the input files is important and will cause errors if not done correctly. Use the existing content in each of these files as a reference when adding additional content for your Cloud Lifecycle Manager.

- Update `control_plane.yml` to add the Cloud Lifecycle Manager.
- Update `server_roles.yml` to add the Cloud Lifecycle Manager role.
- Update `net_interfaces.yml` to add the interface definition for the Cloud Lifecycle Manager.
- Create a `disks_lifecycle_manager.yml` file to define the disk layout for the Cloud Lifecycle Manager.
- Update `servers.yml` to add the dedicated Cloud Lifecycle Manager node.

`Control_plane.yml`: The snippet below shows the addition of a single node cluster into the control plane to host the Cloud Lifecycle Manager service. Note that, in addition to adding the new cluster, you also have to remove the Cloud Lifecycle Manager component from the `cluster1` in the examples:

```
clusters:
   - name: cluster0
     cluster-prefix: c0
     server-role: LIFECYCLE-MANAGER-ROLE
     member-count: 1
     allocation-policy: strict
     service-components:
       - lifecycle-manager
       - ntp-client
   - name: cluster1
     cluster-prefix: c1
     server-role: CONTROLLER-ROLE
     member-count: 3
     allocation-policy: strict
     service-components:
       - lifecycle-manager
       - ntp-server
       - tempest
```

This specifies a single node of role `LIFECYCLE-MANAGER-ROLE` hosting the Cloud Lifecycle Manager.

`Server_roles.yml`: The snippet below shows the insertion of the new server roles definition:

```
 server-roles:

    - name: LIFECYCLE-MANAGER-ROLE
      interface-model: LIFECYCLE-MANAGER-INTERFACES
      disk-model: LIFECYCLE-MANAGER-DISKS
      - name: CONTROLLER-ROLE
```

This defines a new server role which references a new interface-model and disk-model to be used when configuring the server.

`net-interfaces.yml`: The snippet below shows the insertion of the network-interface info:

```
    - name: LIFECYCLE-MANAGER-INTERFACES
      network-interfaces:
        - name: BOND0
          device:
             name: bond0
          bond-data:
             options:
                 mode: active-backup
                 miimon: 200
                 primary: hed3
             provider: linux
             devices:
                 - name: hed3
                 - name: hed4
          network-groups:
             - MANAGEMENT
```

This assumes that the server uses the same physical networking layout as the other servers in the example.

`disks_lifecycle_manager.yml`: In the examples, disk-models are provided as separate files (this is just a convention, not a limitation) so the following should be added as a new file named `disks_lifecycle_manager.yml`:

```
---
   product:
      version: 2

   disk-models:
   - name: LIFECYCLE-MANAGER-DISKS
     # Disk model to be used for Cloud Lifecycle Managers nodes
     # /dev/sda_root is used as a volume group for /, /var/log and /var/crash
     # sda_root is a templated value to align with whatever partition is really used
     # This value is checked in os config and replaced by the partition actually used
     # on sda e.g. sda1 or sda5

     volume-groups:
       - name: ardana-vg
         physical-volumes:
           - /dev/sda_root
```

```
        logical-volumes:
        # The policy is not to consume 100% of the space of each volume group.
        # 5% should be left free for snapshots and to allow for some flexibility.
           - name: root
             size: 80%
             fstype: ext4
             mount: /
           - name: crash
             size: 15%
             mount: /var/crash
             fstype: ext4
             mkfs-opts: -O large_file
         consumer:
               name: os
```

`Servers.yml`: The snippet below shows the insertion of an additional server used for hosting the Cloud Lifecycle Manager. Provide the address information here for the server you are running on, that is, the node where you have installed the SUSE OpenStack Cloud ISO.

```
  servers:
     # NOTE: Addresses of servers need to be changed to match your environment.
     #
     #       Add additional servers as required

     #Lifecycle-manager
     - id: lifecycle-manager
       ip-addr: YOUR IP ADDRESS HERE
       role: LIFECYCLE-MANAGER-ROLE
       server-group: RACK1
       nic-mapping: HP-SL230-4PORT
       mac-addr: 8c:dc:d4:b5:c9:e0
       # ipmi information is not needed

     # Controllers
     - id: controller1
       ip-addr: 192.168.10.3
       role: CONTROLLER-ROLE
```

Important

With a stand-alone deployer, the OpenStack CLI and other clients will not be installed automatically. You need to install OpenStack clients to get the desired OpenStack capabilities. For more information and installation instructions, consult *Book* “Installing with Cloud Lifecycle Manager”, *Chapter 28 “Installing OpenStack Clients”*.

12.2 Configuring SUSE OpenStack Cloud without DVR

By default in the KVM model, the Neutron service utilizes distributed routing (DVR). This is the recommended setup because it allows for high availability. However, if you would like to disable this feature, here are the steps to achieve this.

On your Cloud Lifecycle Manager, make the following changes:

1. In the `~/openstack/my_cloud/config/neutron/neutron.conf.j2` file, change the line below from:

   ```
   router_distributed = {{ router_distributed }}
   ```

   to:

   ```
   router_distributed = False
   ```

2. In the `~/openstack/my_cloud/config/neutron/ml2_conf.ini.j2` file, change the line below from:

   ```
   enable_distributed_routing = True
   ```

   to:

   ```
   enable_distributed_routing = False
   ```

3. In the `~/openstack/my_cloud/config/neutron/l3_agent.ini.j2` file, change the line below from:

   ```
   agent_mode = {{ neutron_l3_agent_mode }}
   ```

   to:

   ```
   agent_mode = legacy
   ```

4. In the `~/openstack/my_cloud/definition/data/control_plane.yml` file, remove the following values from the Compute resource `service-components` list:

   ```
   - neutron-l3-agent
      - neutron-metadata-agent
   ```

Warning

If you fail to remove the above values from the Compute resource service-components list from file `~/openstack/my_cloud/definition/data/control_plane.yml`, you will end up with routers (non_DVR routers) being deployed in the compute host, even though the lifecycle manager is configured for non_distributed routers.

5. Commit your changes to your local git repository:

```
ardana > cd ~/openstack/ardana/ansible
ardana > git add -A
ardana > git commit -m "My config or other commit message"
```

6. Run the configuration processor:

```
ardana > cd ~/openstack/ardana/ansible
ardana > ansible-playbook -i hosts/localhost config-processor-run.yml
```

7. Run the ready deployment playbook:

```
ardana > cd ~/openstack/ardana/ansible
ardana > ansible-playbook -i hosts/localhost ready-deployment.yml
```

8. Continue installation. More information on cloud deployments are available in the *Book "Installing with Cloud Lifecycle Manager", Chapter 7 "Overview"*

12.3 Configuring SUSE OpenStack Cloud with Provider VLANs and Physical Routers Only

Another option for configuring Neutron is to use provider VLANs and physical routers only, here are the steps to achieve this.

On your Cloud Lifecycle Manager, make the following changes:

1. In the `~/openstack/my_cloud/config/neutron/neutron.conf.j2` file, change the line below from:

```
router_distributed = {{ router_distributed }}
```

to:

```
router_distributed = False
```

2. In the ~/openstack/my_cloud/config/neutron/ml2_conf.ini.j2 file, change the line below from:

   ```
   enable_distributed_routing = True
   ```

   to:

   ```
   enable_distributed_routing = False
   ```

3. In the ~/openstack/my_cloud/config/neutron/dhcp_agent.ini.j2 file, change the line below from:

   ```
   enable_isolated_metadata = {{ neutron_enable_isolated_metadata }}
   ```

   to:

   ```
   enable_isolated_metadata = True
   ```

4. In the ~/openstack/my_cloud/definition/data/control_plane.yml file, remove the following values from the Compute resource service-components list:

   ```
   - neutron-l3-agent
     - neutron-metadata-agent
   ```

12.4 Considerations When Installing Two Systems on One Subnet

If you wish to install two separate SUSE OpenStack Cloud 8 systems using a single subnet, you will need to consider the following notes.

The ip_cluster service includes the keepalived daemon which maintains virtual IPs (VIPs) on cluster nodes. In order to maintain VIPs, it communicates between cluster nodes over the VRRP protocol.

A VRRP virtual routerid identifies a particular VRRP cluster and must be unique for a subnet. If you have two VRRP clusters with the same virtual routerid, causing a clash of VRRP traffic, the VIPs are unlikely to be up or pingable and you are likely to get the following signature in your `/etc/keepalived/keepalived.log`:

```
Dec 16 15:43:43 ardana-cp1-c1-m1-mgmt Keepalived_vrrp[2218]: ip address
  associated with VRID not present in received packet : 10.2.1.11
Dec 16 15:43:43 ardana-cp1-c1-m1-mgmt Keepalived_vrrp[2218]: one or more VIP
  associated with VRID mismatch actual MASTER advert
Dec 16 15:43:43 ardana-cp1-c1-m1-mgmt Keepalived_vrrp[2218]: bogus VRRP packet
  received on br-bond0 !!!
Dec 16 15:43:43 ardana-cp1-c1-m1-mgmt Keepalived_vrrp[2218]: VRRP_Instance(VI_2)
  ignoring received advertisment...
```

To resolve this issue, our recommendation is to install your separate SUSE OpenStack Cloud 8 systems with VRRP traffic on different subnets.

If this is not possible, you may also assign a unique routerid to your separate SUSE OpenStack Cloud 8 system by changing the `keepalived_vrrp_offset` service configurable. The routerid is currently derived using the `keepalived_vrrp_index` which comes from a configuration processor variable and the `keepalived_vrrp_offset`.

For example,

1. Log in to your Cloud Lifecycle Manager.

2. Edit your `~/openstack/my_cloud/config/keepalived/defaults.yml` file and change the value of the following line:

   ```
   keepalived_vrrp_offset: 0
   ```

   Change the off value to a number that uniquely identifies a separate vrrp cluster. For example:
   `keepalived_vrrp_offset: 0` for the 1st vrrp cluster on this subnet.
   `keepalived_vrrp_offset: 1` for the 2nd vrrp cluster on this subnet.
   `keepalived_vrrp_offset: 2` for the 3rd vrrp cluster on this subnet.

   ### Important

   You should be aware that the files in the `~/openstack/my_cloud/config/` directory are symlinks to the `~/openstack/ardana/ansible/` directory. For example:

   ```
   ardana > ls -al ~/openstack/my_cloud/config/keepalived/defaults.yml
   ```

```
lrwxrwxrwx 1 stack stack 55 May 24 20:38 /var/lib/ardana/openstack/my_cloud/config/
keepalived/defaults.yml ->
    ../../../ardana/ansible/roles/keepalived/defaults/main.yml
```

If you are using a tool like `sed` to make edits to files in this directory, you might break the symbolic link and create a new copy of the file. To maintain the link, you will need to force `sed` to follow the link:

```
ardana > sed -i --follow-symlinks \
  's$keepalived_vrrp_offset: 0$keepalived_vrrp_offset: 2$' \
  ~/openstack/my_cloud/config/keepalived/defaults.yml
```

Alternatively, directly edit the target of the link `~/openstack/ardana/ansible/roles/keepalived/defaults/main.yml`.

3. Commit your configuration to the Git repository (see *Book "Installing with Cloud Lifecycle Manager", Chapter 10 "Using Git for Configuration Management"*), as follows:

```
ardana > cd ~/openstack/ardana/ansible
ardana > git add -A
ardana > git commit -m "changing Admin password"
```

4. Run the configuration processor with this command:

```
ardana > cd ~/openstack/ardana/ansible
ardana > ansible-playbook -i hosts/localhost config-processor-run.yml
```

5. Use the playbook below to create a deployment directory:

```
ardana > cd ~/openstack/ardana/ansible
ardana > ansible-playbook -i hosts/localhost ready-deployment.yml
```

6. If you are making this change after your initial install, run the following reconfigure playbook to make this change in your environment:

```
ardana > cd ~/scratch/ansible/next/ardana/ansible/
ardana > ansible-playbook -i hosts/verb_hosts FND-CLU-reconfigure.yml
```