



# Security Guide

---



## Security Guide

Publication Date: 07/06/2018

SUSE LLC

10 Canal Park Drive

Suite 200

Cambridge MA 02141

USA

<https://www.suse.com/documentation> 

# Contents

## **1 SUSE® OpenStack Cloud: Security Features Overview 1**

- 1.1 Security features in SUSE OpenStack Cloud 8 1
- 1.2 Role-Based Access Control (RBAC) Support for Neutron Networks 1
- 1.3 Separate Service Administrator Role 1
- 1.4 Inter-service Password Enhancements 2
- 1.5 SELinux for KVM 2
- 1.6 Data In Transit Protection 2
- 1.7 Data-at-Rest Protection Using Project-Based Encryption 3
- 1.8 CADF-Compliant Security Audit Logs 3
- 1.9 PCI Readiness 3

## **2 Key Management with the Barbican Service 4**

- 2.1 Barbican Service Overview 4
- 2.2 Key Features 4
- 2.3 Installation 5
- 2.4 Auditing Barbican Events 9
- 2.5 Barbican Key Management Service Bootstrap Data 10
- 2.6 Known issues and workarounds 14

## **3 Key Management Service Administration 15**

- 3.1 Post-installation verification and administration 15
- 3.2 Updating the Barbican Key Management Service 15
- 3.3 Barbican Settings 15

3.4	Enable or Disable Auditing of Barbican Events	16
3.5	Updating the Barbican API Service Configuration File	17
3.6	Starting and Stopping the Barbican Service	18
3.7	Changing or Resetting a Password	18
3.8	Checking Barbican Status	19
3.9	Updating Logging Configuration	19
<b>4</b>	<b>SUSE® OpenStack Cloud: Service Admin Role Segregation in the Identity Service</b>	<b>21</b>
4.1	Overview	21
4.2	Pre-Installed Service Admin Role Components	21
4.3	Features and Benefits	22
4.4	Roles	22
<b>5</b>	<b>Role-Based Access Control in Neutron</b>	<b>24</b>
5.1	Creating a Network	24
5.2	Creating an RBAC Policy	25
5.3	Listing RBACs	25
5.4	Listing the Attributes of an RBAC	25
5.5	Deleting an RBAC Policy	26
5.6	Sharing a Network with All Tenants	26
5.7	Target Project (demo2) View of Networks and Subnets	30
5.8	Target Project: Creating a Port Using demo-net	31
5.9	Target Project Booting a VM Using Demo-Net	32
5.10	Limitations	34

## 6 Configuring Keystone and Horizon to use X.509 Client Certificates 36

- 6.1 Keystone configuration 36
- 6.2 HAProxy Configuration 40
- 6.3 Create CA and client certificates 41
- 6.4 Horizon configuration 41
- 6.5 Browser configuration 43
- 6.6 User accounts 43
- 6.7 How it works 45

## 7 Transport Layer Security (TLS) Overview 47

- 7.1 SUSE OpenStack Cloud 8 clean install vs. upgrade 47
- 7.2 TLS Configuration 48
  - Using the Default My Public Cert 48 • Certificate Terms 48 • Configuring TLS in the input model 50 • Generating and Signing Certificates 51 • User-provided certificates and trust chains 52 • Edit the Input Model to Include Your Certificate Files 53 • Generating a Self-signed CA 54 • Generate a Certificate Signing Request 56 • Generate a Server Certificate 57 • Upload to the Cloud Lifecycle Manager 59 • Configuring the Cipher Suite 61 • Testing 61 • Verifying That the Trust Chain is Correctly Deployed 61 • Turning TLS on or off 62
- 7.3 Enabling TLS for MySQL Traffic 63
  - Enabling TLS on the database server for client access 63 • MySQL replication over TLS 64 • Enabling TLS for MySQL replication on a new deployment 64 • Enabling TLS for MySQL replication on an existing system 65 • Testing whether a service is using TLS 65
- 7.4 Enabling TLS for RabbitMQ Traffic 66
  - Testing 67

7.5	Troubleshooting TLS	68
	Troubleshooting TLS certificate errors when running playbooks with a limit 68 • Certificate Update Failure 68 • Troubleshooting trust chain installation 68 • Troubleshooting certificates 70	
<b>8</b>	<b>SUSE® OpenStack Cloud: Preventing Host Header Poisoning</b>	<b>71</b>
<b>9</b>	<b>Encryption of Passwords and Sensitive Data</b>	<b>73</b>
9.1		73
9.2	Protecting sensitive data on the Cloud Lifecycle Manager	74
9.3	Interacting with Encrypted Files	75
<b>10</b>	<b>Encryption of Ephemeral Volumes</b>	<b>77</b>
10.1	Enabling ephemeral volume encryption	77
<b>11</b>	<b>Refining Access Control with AppArmor</b>	<b>79</b>
11.1	AppArmor in SUSE OpenStack Cloud 8	79
<b>12</b>	<b>Data at Rest Encryption</b>	<b>80</b>
12.1	Configuring KMIP and ESKM	80
12.2	Configuring Cinder volumes for encryption	83
12.3	For More Information	84
<b>13</b>	<b>Security Audit Logs</b>	<b>85</b>
13.1	The need for auditing	85
13.2	Audit middleware	85
13.3	Centralized auditing configuration	86

# 1 SUSE® OpenStack Cloud: Security Features Overview

## 1.1 Security features in SUSE OpenStack Cloud 8

Enterprises need protection against security breaches, insider threats, and operational issues that increase the risk to sensitive data. By combining technologies from both OpenStack services and Micro Focus Security–Data Security products, SUSE OpenStack Cloud 8 provides capabilities that help you protect your data at rest and in transit, enable centralized key management, and comply with Payment Card Industry Data Security Standard (PCI-DSS).

In SUSE OpenStack Cloud 8, a number of security enhancements are available to strengthen and harden your cloud deployment. Below is an overview of some of the features and brief descriptions. Follow the links to the relevant topics for instructions on setup, configuration, and use of these security features.

## 1.2 Role-Based Access Control (RBAC) Support for Neutron Networks

The RBAC feature in this release enables better security as administrators can now control who has access to specific networks. This is a significant improvement over the previous all-or-nothing approach to shared networks. This is beneficial from a security standpoint as some projects (or tenants) have stricter security policies. For example, a finance department must run PCI-compliant workloads in isolation from other departments, and thus cannot share their Neutron network resources. RBAC enables cloud admins to create granular security policies for sharing Neutron resources with one or more tenants or projects using the standard CRUD (Create, Read, Update, Delete) model. More information can be found in [Chapter 5, Role-Based Access Control in Neutron](#).

## 1.3 Separate Service Administrator Role

Each OpenStack service account now has an optional role available to restrict the OpenStack functions each account can access. This feature enables cloud administrators to apply service-specific role-based, admin-level access to a specific UserID, with the ability to audit ad-

min-level actions. This functionality provides better security by not only providing full visibility into admin-level activities via audit logs, but also by fulfilling compliance requirements such as PCI DSS v3.1 standards. More information in [Section 4.1, “Overview”](#).

## 1.4 Inter-service Password Enhancements

You can conveniently change the inter-service passwords used for authenticating communications between services in your SUSE OpenStack Cloud deployment, promoting better compliance with your organization’s security policies. The inter-service passwords that can be changed include (but are not limited to) Keystone, MariaDB, RabbitMQ, Cloud Lifecycle Manager, Monasca and Barbican. Admins can implement this feature by running the configuration processor to generate new passwords followed by Ansible playbook commands to change the credentials.

## 1.5 SELinux for KVM

SELinux (also known as Security-Enhanced Linux) provides enhanced security at the hypervisor layer on the Red Hat compute nodes by mitigating the risk of hypervisor attacks and strongly isolating the guest VMs. It enforces mandatory access control security policies for the compute nodes (svirt process) running KVM, thus reducing the risk of a hypervisor breakout. By providing a locked down profile for the KVM/QEMU processes that the guest VMs run in, it strongly isolates the guest VMs. With such strong security measures as SELinux, malicious attacks on VMs and the underlying host OS are much less possible. SELinux provides enhanced security for instances managed by libvirt. It does not, however, provide enhanced security for OpenStack processes.

## 1.6 Data In Transit Protection

With SUSE OpenStack Cloud 8, data transmission between internal API endpoints is encrypted using TLS v 1.2 to protect sensitive data against unauthorized disclosure and modification (spoofing and tampering attacks). Additionally, you can configure TLS using your own certificates, from a Certificate Authority of your choice, providing deployment flexibility. More at [Section 7.2, “TLS Configuration”](#).



## 1.7 Data-at-Rest Protection Using Project-Based Encryption

You can encrypt sensitive data-at-rest on per tenant or project basis, while storing and managing keys externally and centrally using Enterprise Secure Key Manager (ESKM). This capability requires the Barbican API and OASIS KMIP (Key Management Interoperability Protocol) plugins for integration, and supports encryption of Cinder block storage with SUSE OpenStack Cloud 8. More information at [Chapter 12, Data at Rest Encryption](#).

## 1.8 CADF-Compliant Security Audit Logs

Security audit logs for critical services such as Keystone, Nova, Cinder, Glance, Heat, Neutron, Barbican are available in a standard CADF (Cloud Audit Data Federation) format. These logs contain information on events such as unauthorized logins, admin level access, unsuccessful login attempts, and anomalous deletion of VMs that are critical from a security threat monitoring standpoint. Audit logs are useful as a tool for risk mitigation, identifying suspicious or anomalous activity, and for fulfilling compliance. For more information see [Chapter 13, Security Audit Logs](#).

## 1.9 PCI Readiness

SUSE OpenStack Cloud 8 is PCI (Payment Card Industry) ready, enabling retail and finance industries that are subject to PCI compliance, to become certified. The readiness is based on lab assessment and verification conducted by an external audit firm, against the more than 250 security requirements specified in the PCI DSS (Data Security Standard) v3.1 standards document. Since SUSE OpenStack Cloud satisfies the requirements that fall under vendor responsibility, customers can proceed with their certification efforts with full confidence and peace of mind that SUSE OpenStack Cloud will not be a blocker.

## 2 Key Management with the Barbican Service

### 2.1 Barbican Service Overview

Barbican is an OpenStack key management service offering secure storage, provisioning, and management of key data. The Barbican service provides management of secrets, keys and certificates via multiple storage back-ends. The support for various back ends is provided via a plug-in mechanism, a Key Management Interoperability Protocol (KMIP) plug-in for a KMIP-compliant HSM Hardware Secure Module (HSM) device. Barbican supports symmetric and asymmetric key generation using various algorithms. Cinder, neutron-lbaas v2 and Nova will integrate with Barbican for their encryption key generation and storage.

Barbican has two types of core feature sets:

- The *Barbican component*, a Web Server Gateway Interface (WSGI) application that exposes a REST API for secrets/containers/orders.
- *Barbican workers* for asynchronous processing, which is used for various messaging-event-driven tasks related to certificate generation.

### 2.2 Key Features

The major features of the Barbican key management service are:

- The ability to encrypt volumes/disks. In an OpenStack context, this means support for encrypting Cinder volumes (volume encryption). Cinder has its own key manager interface (KeyMgr) and can use BarbicanClient as one of its implementations. By default in SUSE OpenStack Cloud 8, Cinder uses Barbican as its key manager when Barbican is enabled. KeyMgr encrypts data in the virtualization host before writing data to the remote disk. There are three options available in SUSE OpenStack Cloud:
  - Tenant-based encryption for block volume storage using Barbican for KMS,
  - Barbican with KMIP and PKCS11 and external KMS (certified with Micro Focus ESKM),
  - 3PAR StoreServ Data-At-Rest Encryption,
- Storage and retrieval of secrets (passwords)

- Certificate management for Load Balancer as a Service V2 (previously known as Neutron-LBaaS)
- The ability to define and manage access policies for key material
- Administrative functionality, and the ability to control the lifecycle of key material
- A well-defined auditing ability in OpenStack services for key access and lifecycle events
- Key management as a service for PaaS application(s) deployed on an OpenStack cloud
- The ability to scale key management effectively and make it highly available (able to handle failover)



### Warning

Do not delete the certificate container associated with your load balancer listeners before deleting the load balancers themselves. If you delete the certificate first, future operations on your load balancers and failover will stop working.

## 2.3 Installation

New installations of SUSE OpenStack Cloud 8:

- For new installations, no changes are needed for Barbican to be enabled. When installing your cloud, you should use the input models which already define the necessary Barbican components. When using the pre-defined input model files that come with SUSE OpenStack Cloud 8, nothing else needs to be done in those files.
- Generate a master key.



### Warning

Do not change your master key after deploying it to Barbican.

- If you decide to make configuration changes to your clean install of SUSE OpenStack Cloud 8, you will need to redeploy the Barbican service. For details on available customization options, please see [Chapter 3, Key Management Service Administration](#).

## Master Key Configuration

Barbican currently supports databases and KMIP as its secret store back-ends. In OpenStack upstream additional back-ends are available, such as the PKCS11 and dogtag plug-ins, but they are not tested or supported by SUSE OpenStack Cloud.

In SUSE OpenStack Cloud, by default Barbican is configured to use a database as a secret (keys) storage back-end. This back-end encrypts Barbican-managed keys with a project level key (*KEK*/Key Encryption Key) before storing it in the database. Project-level keys are encrypted using a master key. As part of the initial Barbican configuration, you must generate and configure this master key.

When Barbican is used with `simple_crypto_plugin` as its secret store back-end, its master key needs to be defined **before initial deployment**. If no key is specified before deployment, the default master key is used—**this practice is discouraged**.

1. Generate the master key using the provided python `*generate_kek*` script on the Cloud Lifecycle Manager node:

```
python ~/openstack/ardana/ansible/roles/KEYMGR-API/templates/generate_kek
```

The master key is generated at stdout from this command.

2. Set the master key in `~/openstack/my_cloud/config/barbican/barbican_deploy_config.yml`.
3. If there is an existing `barbican_customer_master_key` value, replace it with the generated master key you just generated.
4. Commit the change to the Git repository:

```
cd ~/openstack
git add -A
git commit -m "My config"
```

5. Run ready-deployment:

```
cd ~/openstack/ardana/ansible/
ansible-playbook -i hosts/localhost ready-deployment.yml
```

6. When the master key is set, continue with your cloud deployment.

## Upgrade Master Key Configuration

### 1. Check the master key.

If a master key is already defined, check `~/openstack/ardana/ansible/roles/barbican-common/vars/barbican_deploy_config.yml` for `barbican_customer_master_key` value. If the value does not have a prefix `@ardana@`, it is not encrypted. It is highly recommended to encrypt this value.

### 2. Encrypt the existing key during upgrade:

#### a. Set up the environment variable.

```
ARDANA_USER_PASSWORD_ENCRYPT_KEY
```

which contains the key used to encrypt Barbican master key.

#### b. Before you run any playbooks, you need to export the encryption key in the following environment variable:

i. 

```
export ARDANA_USER_PASSWORD_ENCRYPT_KEY=<USER_ENCRYPTION_KEY>
```

ii. 

```
python  
*roles/KEYMGR-API/templates/generate_kek <barbican_customer_master_key>
```

#### iii. Master key is generated at stdout.

#### iv. Set this master key in file

```
~/openstack/ardana/ansible/roles/barbican-common/vars/  
barbican_deploy_config.yml
```

#### v. Replace existing `barbican_customer_master_key` value with the master key you just generated.

#### vi. Commit the change in git repository.

vii. 

```
cd ~/openstack/ardana/ansible/  
ansible-playbook -i hosts/localhost ready-deployment.yml
```

#### viii. When the master key is set, continue with cloud deployment.

3. Changing the master key during the upgrade process is discouraged. Changing the master key will result in a read error for existing secrets as they were encrypted using the previous master key.



### Note

For a Barbican deployment with a database back-end, the master key needs to be generated and configured before Barbican is deployed for the first time. Once the master key is set, it must not be modified.



### Note

Changing the master key can result in read errors for existing secrets as those secrets are stored in the database and are encrypted using the previous master key. Once a new master key is used, Barbican will not be able to read those secrets. Also it will not be able to create new secrets within that project as the project key is encrypted using previous master key.


## KMIP Plug-in Support

Barbican has a KMIP plug-in to store encryption keys (called secrets in Barbican service terminology) in an HSM device using the KMIP protocol. This plug-in has been tested against Micro Focus ESKM with KMIP server. To enable support for it, Barbican needs to be configured with the corresponding plug-in connection details, and client certificate information needs to be defined in its configuration. The ESKM KMIP server uses a client certificate to validate a KMIP client connection established by the Barbican server. As part of that KMIP configuration, playbooks provide a mechanism to upload your client certs to nodes hosting the Barbican API server.

KMIP deployment instructions can be found in [Section 12.1, “Configuring KMIP and ESKM”](#).



### Note

Installation and deployment of the Micro Focus ESKM or any other HSM devices and dependent components is beyond the scope of this document. Please refer the relevant documentation for your choice of product. For example, you can get more information on Micro Focus ESKM and related Data Security and Encryption Products at <https://software.microfocus.com/en-us/products/eskm-enterprise-secure-key-management/overview> .

## 2.4 Auditing Barbican Events

The Barbican service supports auditing and uses *Chapter 13, Security Audit Logs* to generate auditing data in Cloud Auditing Data Federation (CADF) format. The SUSE OpenStack Cloud input model has a mechanism to enable and disable auditing on a per-service basis. When Barbican auditing is enabled, it writes audit messages to an audit log file that is separate from the Barbican internal logging. The base location of audit log file is driven by common auditing configuration.

### Enabling and Disabling Auditing

The auditing of Barbican events can be enabled and disabled through the Barbican reconfigure playbook. As part of the configuration of Barbican, its audit messages can be directed to a log or to a messaging queue. By default, messages are written to the Barbican log file. Once an architecture-level decision is made with regards to the default consumer of audit events (either logging or messaging), the Barbican service can be configured to use it as the default option when auditing is enabled.

Auditing can be disabled or enabled by following these steps on the Cloud Lifecycle Manager node.

#### PROCEDURE 2.1: ENABLING OR DISABLING AUDITING

1. Edit the file `~/openstack/my_cloud/definition/cloudConfig.yml`. All audit-related configuration is defined in the `audit-settings` section. You must use valid YAML syntax when specifying values.
2. Any service (including Barbican) that is listed under `enabled-services` or `disabled-services` will override the default setting. To enable auditing, make sure that the Barbican service name is in the `enabled-services` list of the `audit-settings` section or is not present in `disabled-services` list when `default:` is set to `enabled`.

The relevant section of `cloudConfig.yml` is shown below. Enabled-services are commented out.

The `default: enabled` setting applies to all services. If you want to disable (or enable) a few, whichever is the opposite of the default global setting you used, you can do so in a `disabled-services` (or `enabled-services`) section below it. Here the `enabled-services` entry is commented out. You should only have either a default of `enabled` (or `disabled`) or a section of `disabled` (or `enabled`). There is no need to duplicate the setting.

```
audit-settings:
```

```
default: enabled
#enabled-services:
# - keystone
# - barbican
disabled-services:
  - nova
  - barbican
  - keystone
  - cinder
  - ceilometer
  - neutron
```

3. When you are satisfied with your settings, copy the files to `~/openstack/my_cloud/definition/`, and commit the changes in the git repository. For example, if you are using the Entry Scale KVM model, you would copy from `~/openstack/examples/entry-scale-kvm` and commit.

```
cp -r ~/openstack/examples/entry-scale-kvm/* ~/openstack/my_cloud/definition/
cd ~/openstack
git add -A
git commit -m "My config"
```

4. Run the configuration processor and ready-deployment:

```
cd ~/openstack/ardana/ansible/
ansible-playbook -i hosts/localhost config-processor-run.yml
ansible-playbook -i hosts/localhost ready-deployment.yml
```

5. Run barbican-reconfigure:

```
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts barbican-reconfigure.yml
```

## 2.5 Barbican Key Management Service Bootstrap Data

When the key management service is installed, some of the Keystone-related initial data is bootstrapped as part of its initial deployment. The data added is primarily related to Barbican user, roles, service and endpoint definitions, and Barbican role assignments.



## User, Roles, Service and Endpoint Definitions

Type	Name or key-value pair	Purpose	Comments
Key-stone User Account	barbican	Barbican user account associated with administrative privileges.	Password is randomly generated and made available in the Barbican client environment setup script, <u>barbican.osrc</u> , , on the Cloud Lifecycle Manager node.
Key-stone User Account	barbican_service	Service account used for Keystone token validation by <u>barbican</u> service.	Password is randomly generated and stored in barbican paste configuration, <u>barbican-api-paste.ini</u> .
Key-stone Role	key-manager:creator	Barbican specific role with privilege to create, modify, list, and delete keys and certificates.	This role has the same privileges defined for <u>creator</u> role in upstream Barbican. Referenced in the service policy config file, <u>policy.json</u> .
Key-stone Role	key-manager:admin	Barbican-specific role that has administrative privileges. Privileges include modifications (update and delete) in container's consumer, transport keys, certificate authorities (CA), assignment, and management of per-project CA.	This role has the same privileges defined for <u>admin</u> role in upstream Barbican. Referenced in the service policy config file, <u>policy.json</u> .
Key-stone Role	key-manager:observer	Barbican specific role which has privileges limited to read/list of keys, certificates.	This role has the same privileges defined for <u>observer</u> role in upstream Barbican. Referenced in the service policy config file, <u>policy.json</u> .

Type	Name or key-value pair	Purpose	Comments
Key-stone Role	key-manager:auditor	Barbican specific role which has privileges limited to reading meta-data of keys, certificates. This role does not allow reading and listing of actual keys and certificates.	This role has the same privileges defined for <u>auditor</u> role in upstream Barbican. Referenced in the service policy config file, <u>policy.json</u> .
Key-stone Role	key-manager:service-admin	Barbican specific role which has privilege to modify global preferred CA and modify default project quotas.	This role has the same privileges defined for <u>key-manager:service-admin</u> role in upstream Barbican. Referenced in the service policy config file, <u>policy.json</u> .
Key-stone Service	name: barbican type: key-manager	Barbican service definition. Service type is <i>key-manager</i> .	
Key-stone End-point	interface: internal region: region1	Barbican internal endpoint. This is the load-balanced endpoint exposed for internal service usage.	
Key-stone End-point	interface: public region: region1	Barbican public endpoint. This is the load-balanced endpoint exposed for external/public service usage.	

## Role Assignments

User name	Project name	Role name	Purpose
bar-bican	admin	key-manager:admin	User is assigned Barbican administration privileges on Keystone-defined <u>admin</u> project. This allows the user to manage Barbican resources associated with that project using the Barbican CLI setup.
bar-bican	admin	key-manager:service-admin	User is assigned Barbican service administration privileges on Keystone-defined <u>admin</u> project. This role and the one above allows full Barbican-related administration capabilities.
bar-bican	admin	admin	User assigned Keystone defined administrative role on its <u>admin</u> project. This way customer can continue to use Barbican CLI and OpenStack CLI without need to switch when testing or verifying data.
admin	admin	key-manager:admin	Keystone-defined <u>admin</u> user is given Barbican related administrative privileges on Keystone-defined <u>admin</u> project.
admin	admin	key-manager:service-admin	In lines of above role assignment, Barbican specific service administrator role is assigned to allow global preferred CA and quotas modifications.
barbi-can_service	services	service	Barbican service account is given <u>service</u> role on <u>services</u> project for token validation. API server uses this for creating scoped service token and then includes it as <u>X-Service-Token</u> when requesting customer/client token validation from Keystone.

## 2.6 Known issues and workarounds

1. Make sure that in your Certificate Signing Request (CSR) Common Name matches the barbican\_kmip\_username value defined in roles/barbican-common/vars/barbican\_deploy\_config.yml. Otherwise you may see an internal server error message in Barbican for secret create request.
2. Barbican does not return a clear error message with regards to client certificate setup and its connectivity with KMIP server. During secret create request, a general "Internal Server Error" is returned when the certificate is invalid or missing any of necessary client certificate data (client certificate, key and CA root certificate).

## 3 Key Management Service Administration

### 3.1 Post-installation verification and administration

In a production environment, you can verify your installation of the Barbican key management service by running the `barbican-status.yml` Ansible playbook on the Cloud Lifecycle Manager node.

```
ansible-playbook -i hosts/verb_hosts barbican-status.yml
```

In any non-production environment, along with the playbook, you can also verify the service by storing and retrieving the secret from Barbican.

### 3.2 Updating the Barbican Key Management Service

Some Barbican features and service configurations can be changed. This is done using the Cloud Lifecycle Manager Reconfigure Ansible playbook. For example, the log level can be changed from INFO to DEBUG and vice-versa. If needed, this change can be restricted to a set of nodes via the playbook's host limit option. Barbican administration tasks should be performed by an admin user with a token scoped to the default domain via the Keystone identity API. These settings are preconfigured in the `barbican.osrc` file. By default, `barbican.osrc` is configured with the admin endpoint. If the admin endpoint is not accessible from your network, change `OS_AUTH_URL` to point to the public endpoint.

### 3.3 Barbican Settings

The following Barbican configuration settings can be changed:

- Anything in the main Barbican configuration file: `/etc/barbican/barbican.conf`
- Anything in the main Barbican worker configuration file: `/etc/barbican/barbican-worker.conf`

You can also update the following configuration options and enable the following features. For example, you can:

- Change the verbosity of logs written to Barbican log files (`var/log/barbican/`).
- Enable and disable auditing of the Barbican key management service
- Edit `barbican_secret_store` plug-ins. The two options are:
  - `store_crypto` used to store the secrets in the database
  - `kmip_plugin` used to store the secrets into KMIP-enabled external devices

## 3.4 Enable or Disable Auditing of Barbican Events

Auditing of Barbican key manager events can be disabled or enabled by following steps on the Cloud Lifecycle Manager node.

1. Edit the file `~/openstack/my_cloud/definition/cloudConfig.yml`.  
All audit-related configuration is defined under `audit-settings` section. Valid YAML syntax is required when specifying values.  
Service name defined under `enabled-services` or `disabled-services` override the default setting (i.e. `default: enabled` or `default: disabled`)
2. To enable auditing, make sure that the barbican service name is listed in the `enabled-services` list of `audit-settings` section or is not listed in the `disabled-services` list when default: is set to `enabled`.
3. To disable auditing for the Barbican service specifically, make sure that `barbican service name` is in `disabled-services` list of the `audit-settings` section or is not present in the `enabled-services` list when default: is set to `disabled`. You should not specify the service name in both lists. If it is specified in both, the enabled-services list takes precedence.
4. Commit the change in git repository.

```
cd ~/openstack/ardana/ansible
git add -A
git commit -m "My config"
```

5. Run the `configuration-processor-run` and `ready-deployment` playbooks, followed by the `barbican-reconfigure` playbook:

```
cd ~/openstack/ardana/ansible/
ansible-playbook -i hosts/localhost config-processor-run.yml
ansible-playbook -i hosts/localhost ready-deployment.yml
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts barbican-reconfigure.yml
```

## 3.5 Updating the Barbican API Service Configuration File

1. The Barbican API service configuration file (`/etc/barbican/barbican.conf`), located on each control plane server (controller node) is generated from the following template file located on the Cloud Lifecycle Manager node: `/var/lib/ardana/openstack/my_cloud/config/barbican/barbican.conf.j2`. Modify this template file as appropriate. This is a Jinja2 template, which expects certain template variables to be set. Do not change values inside double curly braces: `{{ }}`.
2. Once the template is modified, copy the files to `~/openstack/my_cloud/definition/`, and commit the change to the local git repository:

```
cp -r ~/hp-ci/padawan/* ~/openstack/my_cloud/definition/
cd ~/openstack/ardana/ansible
git add -A
git commit -m "My config"
```

3. Then rerun the configuration processor and ready-deployment playbooks:

```
cd ~/openstack/ardana/ansible
ansible-playbook -i hosts/localhost config-processor-run.yml
ansible-playbook -i hosts/localhost ready-deployment.yml
```

4. Finally, run the `barbican-reconfigure` playbook in the deployment area:

```
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts barbican-reconfigure.yml
```

## 3.6 Starting and Stopping the Barbican Service

You can start or stop the Barbican service from the Cloud Lifecycle Manager nodes by running the appropriate Ansible playbooks:

To stop the Barbican service:

```
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts barbican-stop.yml
```

To start the Barbican service:

```
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts barbican-start.yml
```

## 3.7 Changing or Resetting a Password

To change the password for the Barbican administrator:

1. Copy the file as shown below:

```
cp ~/openstack/my_cloud/info/private_data_metadata_ccp.yml \
~/openstack/change_credentials/
```

2. Then edit `private_data_metadata_ccp.yml` found here:

```
~/openstack/change_credentials/private_data_metadata_ccp.yml
```

3. Change credentials for the Barbican admin user and/or Barbican service user. Remove everything else. The file will look similar to this:

```
barbican_admin_password:
  value: 'testing_123'
  metadata:
    - clusters:
        - cluster1
          component: barbican-api
          cp: ccp
          version: '2.0'
barbican_service_password:
  value: 'testing_123'
  metadata:
    - clusters:
        - cluster1
```



```
component: barbican-api
cp: ccp
version: '2.0'
```

The value (shown in bold) is optional; it is used to set a user-chosen password. If left blank, the playbook will generate a random password.

4. Execute the following playbooks from ~/openstack/ardana/ansible/:

```
cd ~/openstack/ardana/ansible/
ansible-playbook -i hosts/localhost config-processor-run.yml -e encrypt="" -e
rekey=""
ansible-playbook -i hosts/localhost ready-deployment.yml
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts barbican-reconfigure-credentials-change.yml
```

5. SSH to the controller and make sure the password has been properly updated.

```
/etc/barbican# vi barbican-api-paste.ini
```

## 3.8 Checking Barbican Status

You can check the status of Barbican by running the barbican-status.yml Ansible playbook on the Cloud Lifecycle Manager node.

```
ansible-playbook -i hosts/verb_hosts barbican-status.yml
```



### Note

Make sure you remove/delete ~/openstack/change\_credentials/private\_data\_metadata.yml after successfully changing the password.

## 3.9 Updating Logging Configuration

All Barbican logging is set to INFO by default. To change the level from the Cloud Lifecycle Manager, there are two options available

1. Edit the Barbican configuration file, /barbican\_deploy\_config.yml, in the following directory.

```
~/openstack/my_cloud/config/barbican/
```

To change log level entry (barbican\_loglevel) to DEBUG, edit the entry:

```
barbican_loglevel = {{ openstack_loglevel | default('DEBUG') }}
```

To change the log level to INFO, edit the entry:

```
barbican_loglevel = {{ openstack_loglevel | default('INFO') }}
```

2. Edit file ~/openstack/ardana/ansible/roles/KEYMGR-API/templates/api-logging.conf.j2 and update the log level accordingly.

Commit the change to the local git repository:

```
cd ~/openstack/ardana/ansible
git add -A
git commit -m "My config"
```

Run the configuration-processor-run and ready-deployment playbooks, followed by the barbican-reconfigure playbook:

```
ansible-playbook -i hosts/localhost config-processor-run.yml
ansible-playbook -i hosts/localhost ready-deployment.yml
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts barbican-reconfigure.yml
```

## 4 SUSE® OpenStack Cloud: Service Admin Role Segregation in the Identity Service

### 4.1 Overview

Under the default OpenStack user policies, a user can have either member privilege or admin privilege. Admin privilege is assigned by creating a user account with the role of admin. However, the default admin role is too broad and often grants users more privilege than they need, giving them access to additional tasks and resources that they shouldn't have.

Ideally, each user account should only be assigned privileges necessary to perform tasks they are required to perform. According to the widely accepted principle of least privilege, a user who needs to perform administrative tasks should have a user account with the privileges required to perform only those administrative tasks and no others. This prevents the granting of too much privilege while retaining the individual accountability of the user.

Service Administrator Roles is an alternative to the current one-size-fits-all admin role model and can help you institute different privileges for different administrators.


### 4.2 Pre-Installed Service Admin Role Components

The main components of Service Administrator Roles are:

- nova\_admin role in the identity service (Keystone) and support in nova\_policy.json
- neutron\_admin role in the identity service and support in neutron\_policy.json
- cinder\_admin role in the identity service and support in cinder\_policy.json
- glance\_admin role in the identity service and support in glance\_policy.json



#### Warning: Changing `glance_policy.json` may Introduce a Security Issue

The OpenStack Security Note OSSN-0075 <https://wiki.openstack.org/wiki/OSSN/OSSN-0075>  describes a scenario where a malicious tenant is able to reuse deleted Glance image IDs to share malicious images with other tenants in a manner that is undetectable to the victim tenant.

The default policy `glance_policy.json` that is shipped with SUSE OpenStack Cloud prevents this by ensuring only admins can deactivate/reactivate images:

```
"deactivate": "role:admin"
"reactivate": "role:admin"
```

It is suggested to *not* change these settings. If you do change them please refer to the OSSN-0075 <https://wiki.openstack.org/wiki/OSSN/OSSN-0075> for details on the exact scope of the security issue.

## 4.3 Features and Benefits

Service Administrator Roles offer the following features and benefits:

- Support separation of duties through more granular roles
- Are enabled by default
- Are backwards compatible
- Have predefined service administrator roles in the identity service
- Have predefined `policy.json` files with corresponding service admin roles to facilitate quick and easy deployment

## 4.4 Roles

The following are the roles defined in SUSE OpenStack Cloud 8. These roles serve as a way to group common administrative needs at the OpenStack service level. Each role represents administrative privilege into each service. Multiple roles can be assigned to a user. You can assign a Service Admin Role to a user once you have determined that the user is authorized to perform administrative actions and access resources in that service.

### Pre-Installed Service Admin Roles

The following service admin roles exist by default:

#### nova\_admin role

Assign this role to users whose job function it is to perform Nova compute-related administrative tasks.

#### **neutron\_admin role**

Assign this role to users whose job function it is to perform neutron networking-related administrative tasks.

#### **cinder\_admin role**

Assign this role to users whose job function it is to perform Cinder storage-related administrative tasks.

#### **glance\_admin role**

Assign this role to users whose job function it is to perform Glance image service-related administrative tasks.

For configuration steps, see *Book "User Guide Overview", Chapter 4 "Cloud Admin Actions with the Command Line"*.

## 5 Role-Based Access Control in Neutron

This topic explains how to achieve more granular access control for your Neutron networks.

Previously in SUSE OpenStack Cloud, a network object was either private to a project or could be used by all projects. If the network's shared attribute was True, then the network could be used by every project in the cloud. If false, only the members of the owning project could use it. There was no way for the network to be shared by only a subset of the projects.

Neutron Role Based Access Control (RBAC) solves this problem for networks. Now the network owner can create RBAC policies that give network access to target projects. Members of a targeted project can use the network named in the RBAC policy the same way as if the network was owned by the project. Constraints are described in the section [Section 5.10, "Limitations"](#).

With RBAC you are able to let another tenant use a network that you created, but as the owner of the network, you need to create the subnet and the router for the network.

To use RBAC, Neutron configuration files do not need to be changed.

### 5.1 Creating a Network

```
$ neutron net-create demo-net
$ neutron net-show demo-net
```

```
+-----+-----+
| Field          | Value                                |
+-----+-----+
| admin_state_up | True                                |
| id             | c3d55c21-d8c9-4ee5-944b-560b7e0ea33b |
| mtu            | 0                                  |
| name           | demo-net                           |
| router:external| False                              |
| shared         | False                              |
| status         | ACTIVE                             |
| subnets       | d9b765da-45eb-4543-be96-1b69a00a2556 |
| tenant_id      | 75eb5efae5764682bca2fede6f4d8c6f   |
+-----+-----+
```

## 5.2 Creating an RBAC Policy

Here we will create an RBAC policy where a member of the project called 'demo' will share the network with members of project 'demo2'

To create the RBAC policy, run:

```
neutron rbac-create --target-tenant <demo2-project-uuid> --type network --action
access_as_shared demo-net
```

Here is an example where the demo2 project id is 5a582af8b44b422fafcd4545bd2b7eb5

```
$ neutron rbac-create --target-tenant 5a582af8b44b422fafcd4545bd2b7eb5 \
--type network --action access_as_shared demo-net
```

## 5.3 Listing RBACs

To list all the RBAC rules/policies, execute:

```
$ neutron rbac-list
+-----+-----+
| id                  | object_id                  |
+-----+-----+
| 0fd89dcb-9809-4a5e-adc1-39dd676cb386 | c3d55c21-d8c9-4ee5-944b-560b7e0ea33b |
+-----+-----+
```

## 5.4 Listing the Attributes of an RBAC

To see the attributes of a specific RBAC policy, run

```
$ neutron rbac-show <policy ID>
```

For example:

```
$ neutron rbac-show 0fd89dcb-9809-4a5e-adc1-39dd676cb386
```

Here is the output:

```
+-----+-----+
| Field          | Value                      |
+-----+-----+
```

```
+-----+-----+
| action      | access_as_shared |
| id          | 0fd89dcb-9809-4a5e-adc1-39dd676cb386 |
| object_id   | c3d55c21-d8c9-4ee5-944b-560b7e0ea33b |
| object_type | network           |
| target_tenant | 5a582af8b44b422fafcd4545bd2b7eb5 |
| tenant_id   | 75eb5efae5764682bca2fed6f4d8c6f |
+-----+-----+
```

## 5.5 Deleting an RBAC Policy

To delete an RBAC policy, run `rbac-delete` passing the policy id:

```
$ neutron rbac-delete <policy ID>
```

For example:

```
$ neutron rbac-delete 0fd89dcb-9809-4a5e-adc1-39dd676cb386
```

Here is the output:

```
Deleted rbac_policy: 0fd89dcb-9809-4a5e-adc1-39dd676cb386
```

## 5.6 Sharing a Network with All Tenants

Either the administrator or the network owner can make a network shareable by all tenants.

The administrator can make a tenant's network shareable by all tenants. To make the network `demo-shareall-net` accessible by all tenants in the cloud:

To share a network with all tenants:

1. Get a list of all projects

```
stack@padawan-ccp-c0-m1-mgmt:~$ . ./service.osrc
stack@padawan-ccp-c0-m1-mgmt:~$ openstack project list
```

which produces the list:

```
+-----+-----+
| ID          | Name          |
+-----+-----+
```



```
+-----+-----+
| 1be57778b61645a7alc07ca0ac488f9e | demo |
| 5346676226274cd2b3e3862c2d5ceadd | admin |
| 749a557b2b9c482ca047e8f4abf348cd | swift-monitor |
| 8284a83df4df429fb04996c59f9a314b | swift-dispersion |
| c7a74026ed8d4345a48a3860048dcb39 | demo-sharee |
| e771266d937440828372090c4f99a995 | glance-swift |
| f43fb69f107b4b109d22431766b85f20 | services |
+-----+-----+
```

## 2. Get a list of networks:

```
stack@padawan-ccp-c0-m1-mgmt:~$ neutron net-list
```

This produces the following list:

```
+-----+-----+
+-----+
| id | name | subnets |
+-----+
| f50f9a63-c048-444d-939d-370cb0af1387 | ext-net | ef3873db-fc7a-4085-8454-5566fb5578ea  
172.31.0.0/16 |
| 9fb676f5-137e-4646-ac6e-db675a885fd3 | demo-net | 18fb0b77-fc8b-4f8d-9172-ee47869f92cc  
10.0.1.0/24 |
| 8eada4f7-83cf-40ba-aa8c-5bf7d87cca8e | demo-shareall-net | 2bbc85a9-3ffe-464c-944b-2476c7804877  
10.0.250.0/24 |
| 73f946ee-bd2b-42e9-87e4-87f19edd0682 | demo-share-subset | c088b0ef-f541-42a7-b4b9-6ef3c9921e44  
10.0.2.0/24 |
+-----+
+-----+
```

## 3. Set the network you want to share to a shared value of True:

```
stack@padawan-ccp-c0-m1-mgmt:~$ neutron net-update 8eada4f7-83cf-40ba-aa8c-5bf7d87cca8e --shared True
```

You should see the following output:

```
Updated network: 8eada4f7-83cf-40ba-aa8c-5bf7d87cca8e
```

## 4. Check the attributes of that network by running the following command using the ID of the network in question:

```
stack@padawan-ccp-c0-m1-mgmt:~$ neutron net-show 8eada4f7-83cf-40ba-aa8c-5bf7d87cca8e
```

The output will look like this:

```
+-----+-----+
+-----+
```

Field	Value
admin_state_up	True
id	8eada4f7-83cf-40ba-aa8c-5bf7d87cca8e
mtu	0
name	demo-shareall-net
provider:network_type	vxlan
provider:physical_network	
provider:segmentation_id	1055
router:external	False
shared	True
status	ACTIVE
subnets	2bbc85a9-3ffe-464c-944b-2476c7804877
tenant_id	1be57778b61645a7a1c07ca0ac488f9e

- As the owner of the `demo-shareall-net` network, view the RBAC attributes for `demo-shareall-net` (`id=8eada4f7-83cf-40ba-aa8c-5bf7d87cca8e`) by first getting an RBAC list:

```
stack@padawan-ccp-c0-m1-mgmt:~$ echo $OS_USERNAME ; echo $OS_PROJECT_NAME
demo
demo
stack@padawan-ccp-c0-m1-mgmt:~$ neutron rbac-list
```

This produces the list:

id	object_id
...	
3e078293-f55d-461c-9a0b-67b5dae321e8	8eada4f7-83cf-40ba-aa8c-5bf7d87cca8e

- View the RBAC information:

```
stack@padawan-ccp-c0-m1-mgmt:~$ neutron rbac-show 3e078293-f55d-461c-9a0b-67b5dae321e8
```

Field	Value
action	access_as_shared
id	3e078293-f55d-461c-9a0b-67b5dae321e8
object_id	8eada4f7-83cf-40ba-aa8c-5bf7d87cca8e
object_type	network
target_tenant	*

tenant_id	1be57778b61645a7a1c07ca0ac488f9e
-----------	----------------------------------

- With network RBAC, the owner of the network can also make the network shareable by all tenants. First create the network:

```
stack@padawan-ccp-c0-m1-mgmt:~$ echo $OS_PROJECT_NAME ; echo $OS_USERNAME
demo
demo
stack@padawan-ccp-c0-m1-mgmt:~$ neutron net-create test-net
```

The network is created:

Field	Value
admin_state_up	True
id	a4bd7c3a-818f-4431-8cdb-fedf7ff40f73
mtu	0
name	test-net
router:external	False
shared	False
status	ACTIVE
subnets	
tenant_id	1be57778b61645a7a1c07ca0ac488f9e

- Create the RBAC. It is important that the asterisk is surrounded by single-quotes to prevent the shell from expanding it to all files in the current directory.

```
stack@padawan-ccp-c0-m1-mgmt:~$ neutron rbac-create --type network \
--action access_as_shared --target-tenant '*' test-net
```

Here are the resulting RBAC attributes:

Field	Value
action	access_as_shared
id	0b797cc6-debc-48a1-bf9d-d294b077d0d9
object_id	a4bd7c3a-818f-4431-8cdb-fedf7ff40f73
object_type	network
target_tenant	*
tenant_id	1be57778b61645a7a1c07ca0ac488f9e

## 5.7 Target Project (demo2) View of Networks and Subnets

Note that the owner of the network and subnet is not the tenant named `demo2`. Both the network and subnet are owned by tenant `demo`. `Demo2` members cannot create subnets of the network. They also cannot modify or delete subnets owned by `demo`.

As the tenant `demo2`, you can get a list of neutron networks:

```
$ neutron net-list
```

```
+-----+-----+
+-----+
| id                  | name      | subnets
|
+-----+-----+
+-----+-----+
| f60f3896-2854-4f20-b03f-584a0dcce7a6 | ext-net   | 50e39973-b2e3-466b-81c9-31f4d83d990b
|
| c3d55c21-d8c9-4ee5-944b-560b7e0ea33b | demo-net  | d9b765da-45eb-4543-be96-1b69a00a2556
10.0.1.0/24 |
...
+-----+-----+
+-----+-----+
```

And get a list of subnets:

```
$ neutron subnet-list --network-id c3d55c21-d8c9-4ee5-944b-560b7e0ea33b
```

```
+-----+-----+-----+-----+
+-----+
| id                  | name      | cidr      | allocation_pools
|
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| d9b765da-45eb-4543-be96-1b69a00a2556 | sb-demo-net | 10.0.1.0/24 | {"start": "10.0.1.2", "end":
"10.0.1.254"} |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

To show details of the subnet:

```
$ neutron subnet-show d9b765da-45eb-4543-be96-1b69a00a2556
```

```
+-----+-----+
| Field          | Value
+-----+-----+
```

```

| allocation_pools | {"start": "10.0.1.2", "end": "10.0.1.254"} |
| cidr             | 10.0.1.0/24                               |
| dns_nameservers  |                                             |
| enable_dhcp      | True                                       |
| gateway_ip       | 10.0.1.1                                   |
| host_routes      |                                             |
| id               | d9b765da-45eb-4543-be96-1b69a00a2556    |
| ip_version       | 4                                         |
| ipv6_address_mode |                                             |
| ipv6_ra_mode     |                                             |
| name             | sb-demo-net                               |
| network_id       | c3d55c21-d8c9-4ee5-944b-560b7e0ea33b    |
| subnetpool_id    |                                             |
| tenant_id        | 75eb5efae5764682bca2fede6f4d8c6f       |
+-----+-----+

```

## 5.8 Target Project: Creating a Port Using demo-net

The owner of the port is demo2. Members of the network owner project (demo) will not see this port.

Running the following command:

```
$ neutron port-create c3d55c21-d8c9-4ee5-944b-560b7e0ea33b
```

Creates a new port:

```

+-----+
+-----+
| Field          | Value
+-----+
+-----+
| admin_state_up | True
|
| allowed_address_pairs |
|
| binding:vnic_type | normal
|
| device_id        |
|
| device_owner      |
|
| dns_assignment    | {"hostname": "host-10-0-1-10", "ip_address": "10.0.1.10", "fqdn":
"host-10-0-1-10.openstacklocal."} |
| dns_name          |
|
| fixed_ips         | {"subnet_id": "d9b765da-45eb-4543-be96-1b69a00a2556", "ip_address": "10.0.1.10"}
|

```

id	03ef2dce-20dc-47e5-9160-942320b4e503
mac_address	fa:16:3e:27:8d:ca
name	
network_id	c3d55c21-d8c9-4ee5-944b-560b7e0ea33b
security_groups	275802d0-33cb-4796-9e57-03d8ddd29b94
status	DOWN
tenant_id	5a582af8b44b422fafcd4545bd2b7eb5
+-----+	
+-----+	

## 5.9 Target Project Booting a VM Using Demo-Net

Here the tenant demo2 boots a VM that uses the demo-net shared network:

```
$ nova boot --flavor 1 --image $OS_IMAGE --nic net-id=c3d55c21-
d8c9-4ee5-944b-560b7e0ea33b demo2-vm-using-demo-net-nic
```

+-----+	
Property	Value
+-----+	
OS-EXT-AZ:availability_zone	
OS-EXT-STS:power_state	0
OS-EXT-STS:task_state	scheduling
OS-EXT-STS:vm_state	building
OS-SRV-USG:launched_at	-
OS-SRV-USG:terminated_at	-
accessIPv4	
accessIPv6	
adminPass	sS9uSv9PT79F
config_drive	
created	2016-01-04T19:23:24Z
flavor	m1.tiny (1)
hostId	
id	3a4dc44a-027b-45e9-acf8-054a7c2dca2a
image	cirros-0.3.3-x86_64 (6ae23432-8636-4e...1efc5)
key_name	-
metadata	{}
name	demo2-vm-using-demo-net-nic
os-extended-volumes:volumes_attached	[]
progress	0
security_groups	default
status	BUILD
tenant_id	5a582af8b44b422fafcd4545bd2b7eb5
updated	2016-01-04T19:23:24Z
user_id	a0e6427b036344fdb47162987cb0cee5

```
+-----+-----+-----+-----+
```

Run nova-list:

```
$ nova list
```

See the VM running:

```
+-----+-----+-----+-----+-----+
+-----+
| ID              | Name                               | Status | Task State | Power State | Networks
+-----+
| 3a4dc...a7c2dca2a | demo2-vm-using-demo-net-nic | ACTIVE | -          | Running      | demo-
net=10.0.1.11 |
+-----+
+-----+
```

Run neutron port-list:

```
stack@padawan-ccp-c0-m1-mgmt:~$ neutron port-list --device-id 3a4dc44a-027b-45e9-
acf8-054a7c2dca2a
```

View the subnet:

```
+-----+-----+-----+-----+
+-----+
| id              | name | mac_address      | fixed_ips
+-----+
| 7d14ef8b-9...80348f |      | fa:16:3e:75:32:8e | {"subnet_id": "d9b765da-45...00a2556",
"ip_address": "10.0.1.11"} |
+-----+
+-----+
```

Run neutron port-show:

```
stack@padawan-ccp-c0-m1-mgmt:~$ neutron port-show 7d14ef8b-9d48-4310-8c02-00c74d80348f
```

```
+-----+
+-----+
| Field          | Value
+-----+
| admin_state_up | True
+-----+
| allowed_address_pairs |
+-----+
```

binding:vnic_type	normal
device_id	3a4dc44a-027b-45e9-acf8-054a7c2dca2a
device_owner	compute:None
dns_assignment	{"hostname": "host-10-0-1-11", "ip_address": "10.0.1.11", "fqdn": "host-10-0-1-11.openstacklocal."}
dns_name	
extra_dhcp_opts	
fixed_ips	{"subnet_id": "d9b765da-45eb-4543-be96-1b69a00a2556", "ip_address": "10.0.1.11"}
id	7d14ef8b-9d48-4310-8c02-00c74d80348f
mac_address	fa:16:3e:75:32:8e
name	
network_id	c3d55c21-d8c9-4ee5-944b-560b7e0ea33b
security_groups	275802d0-33cb-4796-9e57-03d8ddd29b94
status	ACTIVE
tenant_id	5a582af8b44b422fafcd4545bd2b7eb5
+-----+	

## 5.10 Limitations

Note the following limitations of RBAC in Neutron.

- Neutron network is the only supported RBAC Neutron object type.
- The "access\_as\_external" action is not supported – even though it is listed as a valid action by `python-neutronclient`.
- The neutron-api server will not accept action value of 'access\_as\_external'. The `access_as_external` definition is not found in the specs.
- The target project users cannot create, modify, or delete subnets on networks that have RBAC policies.
- The subnet of a network that has an RBAC policy cannot be added as an interface of a target tenant's router. For example, the command `neutron router-interface-add tgt-tenant-router <sb-demo-net uuid>` will error out.



- The security group rules on the network owner do not apply to other projects that can use the network.
- A user in target project can boot up VMs using a VNIC using the shared network. The user of the target project can assign a floating IP (FIP) to the VM. The target project must have SG rules that allows SSH and/or ICMP for VM connectivity.
- Neutron RBAC creation and management are currently not supported in Horizon. For now, the Neutron CLI has to be used to manage RBAC rules.
- A RBAC rule tells Neutron whether a tenant can access a network (Allow). Currently there is no DENY action.
- Port creation on a shared network fails if --fixed-ip is specified in the neutron port-create command.

## 6 Configuring Keystone and Horizon to use X.509 Client Certificates

The Keystone service supports X.509 SSL certificate authentication and authorization for accessing the Horizon dashboard in SUSE OpenStack Cloud. This feature is disabled by default, and must be manually configured and enabled by running a number of Ansible playbooks.



### Note

Enabling client SSL certificate authentication and authorization for the Horizon dashboard is a non-core feature in SUSE OpenStack Cloud.

### 6.1 Keystone configuration

To configure and enable X.509 SSL authentication and authorization support for the Keystone service, perform the following steps.

1. Create a new configuration file named `x509auth.yml` and place it in any directory in your deployer node. For example, perform the following command to create the file in the `/tmp` directory:

```
touch /tmp/x509auth.yml
```

2. Edit the new file to include the following text. Note that YAML files are whitespace-sensitive. Preserve the indentation format of the following text.

```
keystone_x509auth_conf:
  identity_provider:
    id: intermediateca
    description: This is the trusted issuer HEX Id.
  mapping:
    id: x509_mapping1
    rules_file: /tmp/x509auth_mapping.json
  protocol:
    id: x509
  remote_id: intermediateca
  ca_file: /tmp/cacert.pem
```

The preceding example sets a number of configuration parameters for the X.509/Keystone configuration. The following are detailed descriptions of each parameter.

- **identity\_provider** This section identifies and describes an outside identity provider.
  - **id**: Any unique, readable string that identifies the identity provider.
  - **description**: A description of the identity provider.
- **mapping**: This section describes a JSON-format file that maps X.509 client certificate attributes to a local Keystone user.
  - **id**: Any unique, readable string that identifies the user-certificate mapping.
  - **rules\_file**: The filepath to a JSON file that contains the client certificate attributes mapping.
- **protocol**: This section sets the cryptographic protocol to be used.
  - **id**: The cryptographic protocol used for the certificate-based authentication/authorization.
- **remote\_id**: By default, this field expects the client certificate's issuer's common name (CN) as a value. The expected value is set in the `keystone.conf` file, where the default setting is:

```
remote_id_attribute = SSL_CLIENT_I_DN_CN
```

- **ca\_file**: The file that contains the client certificate's related intermediary and root CA certificates.

Note: In the `/tmp/x509auth.yml` file, the `ca_file` value should be a file that contains both the root and signing CA certificates (often found in `/home/pki/cacert.pem`).

3. Create a JSON-formatted mapping file. To do so, edit the `x509auth.yml` file you created in [Step 2](#) to reference this file in the **mapping**→**rules\_file** parameter. You can create the file with the following example command:

```
touch /tmp/x509auth_mapping.json
```

4. Edit the JSON file you created in [Step 3](#) to include the following content:

```
[
  {
    "local": [
      {
        "user": {
          "name": "{0}",
          "domain": {
            "name": "{1}"
          },
          "type": "local"
        }
      }
    ],
    "remote": [
      {
        "type": "SSL_CLIENT_S_DN_CN"
      },
      {
        "type": "SSL_CLIENT_S_DN_O"
      },
      {
        "type": "SSL_CLIENT_I_DN",
        "any_one_of": [
        ]
      }
    ]
  }
]
```

5. Enter the distinguished name(s) (DN) of the certificate issuer(s) that issued your client certificates into the **any\_one\_of** field in the **remote** block. The **any\_one\_of** field is a comma-separated list of all certificate issuers that you want the Keystone service to trust. All DNs in the **any\_one\_of** field must adhere to the following format: A descending list of DN elements, with each element separated by a forward slash (/). The following is an example of a properly formatted DN for a certificate issuer named intermedia.

```
/C=US/ST=California/O=EXAMPLE/OU=Engineering/CN=intermediateca/
emailAddress=user@example.com
```

The following example file illustrates an `x509auth_mapping.json` file with the inter-media certificate issuer added to the **any\_one\_of** field. Note that the DN string is in quotes.

```
[
    {
        "local": [
            {
                "user": {
                    "name": "{0}",
                    "domain": {
                        "name": "{1}"
                    },
                    "type": "local"
                }
            }
        ],
        "remote": [
            {
                "type": "SSL_CLIENT_S_DN_CN"
            },
            {
                "type": "SSL_CLIENT_S_DN_O"
            },
            {
                "type": "SSL_CLIENT_I_DN",
                "any_one_of": [
                    "/C=US/ST=California/O=EXAMPLE/OU=Engineering/
CN=intermediateca/emailAddress=user@example.com"
                ]
            }
        ]
    }
]
```

The Keystone service will trust all client certificates issued by any of the certificate issuers listed in the **any\_one\_of** field.

6. Run the following commands to enable the new X.509/Keystone settings.

```
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts keystone-reconfigure.yml -e@/tmp/x509auth.yml
```

## 6.2 HAProxy Configuration

Because of the experimental nature of the HAProxy feature, it is important to minimize the risk of impacting other services. If you have implemented, or wish to implement the HAProxy feature alongside client SSL certificate login to the Horizon dashboard in your cloud, please complete the following steps to make the necessary manual configuration changes.



### Note

You must perform the Keystone configuration steps in the previous section before performing the following HAProxy configuration changes.

1. Locate and open the `~/openstack/ardana/ansible/roles/haproxy/templates/haproxy.cfg` file.
2. Locate the following line in the `haproxy.cfg` file.

```
listen {{ network.vip }}-{{ port }}
```

Enter the following codeblock in the open space immediately preceding the `listen {{ network.vip }}-{{ port }}` line.

```
{%- if service == 'KEY_API' and port == '5000' %}
    {% set bind_defaults = 'ca-file /etc/ssl/private/cacert.pem verify optional' %}
{%- endif %}
```

After entering the preceding code, your `haproxy.cfg` file should look like the following example.

```
{%- if network.terminate_tls is defined and network.terminate_tls and port == '80'
%}
    {% set port = '443' %}
{%- endif %}

{%- if service == 'KEY_API' and port == '5000' %}
    {% set bind_defaults = 'ca-file /etc/ssl/private/cacert.pem verify optional' %}
{%- endif %}

listen {{ network.vip }}-{{ port }}
    {%- set options = network.vip_options | default(vip_options_defaults) %}
    {%- if options > 0 %}
        {%- for option in options %}
```

```
{{ option }}
    {%- endfor %}
    {%- endif %}
    bind {{ network.vip }}:{{ port }} {% if network.terminate_tls is defined
and network.terminate_tls %} ssl crt {{ frontend_server_cert_directory }}/
{{ network.cert_file }} {{ bind_defaults }} {% endif %}
```

3. Commit the changes to your local git repository.

```
git add -A
git commit -m "Added HAProxy configuration changes"
```

4. Run the configuration processor and ready-deployment Ansible playbooks.

```
cd ~/openstack/ardana/ansible
ansible-playbook -i hosts/localhost config-processor-run.yml
ansible-playbook -i hosts/localhost ready-deployment.yml
```

5. Implement the HAProxy configuration changes.

```
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts FND-CLU-reconfigure.yml
```

## 6.3 Create CA and client certificates

An X.509 client certificate can be issued from any certificate authority (CA). You can use the openssl command-line tool to generate certificate signing requests (CSRs). Once a CA has signed your CSR, the CA will return a signed certificate that you can use to authenticate to Horizon.

Read more about openssl here: <https://www.openssl.org/> 



### Note

Your cloud's load balancer will reject any self-signed client SSL certificates. Ensure that all client certificates are signed by a certificate authority that your cloud recognizes.

## 6.4 Horizon configuration

Complete the following steps to configure Horizon to support SSL certificate authorization and authentication.

1. Edit the `~/openstack/ardana/ansible/roles/HZN-WEB/defaults/main.yml` file and set the following parameter to `True`.

```
horizon_websso_enabled: True
```

2. Locate the last line in the `~/openstack/ardana/ansible/roles/HZN-WEB/defaults/main.yml` file. The default configuration for this line should look like the following.

```
horizon_websso_choices:
- {protocol: saml2, description: "ADFS Credentials"}
```

- If your cloud **does not** have AD FS enabled, then replace the preceding `horizon_websso_choices:` parameter with the following.

```
- {protocol: x509, description: "X.509 SSL Certificate"}
```

The resulting block should look like the following.

```
horizon_websso_choices:
- {protocol: x509, description: "X.509 SSL Certificate"}
```

- If your cloud **does** have AD FS enabled, then simply add the following parameter to the `horizon_websso_choices:` section. Do not replace the default parameter, add the following line to the existing block.

```
- {protocol: saml2, description: "ADFS Credentials"}
```

If your cloud has AD FS enabled, the final block of your `~/openstack/ardana/ansible/roles/HZN-WEB/defaults/main.yml` should have the following entries.

```
horizon_websso_choices:
- {protocol: x509, description: "X.509 SSL Certificate"}
- {protocol: saml2, description: "ADFS Credentials"}
```

3. Run the following commands to add your changes to the local git repository, and reconfigure the Horizon service, enabling the changes made in **Step 1**:

```
cd ~/openstack
git add -A
git commit -m "my commit message"
cd ~/openstack/ardana/ansible/
ansible-playbook -i hosts/localhost config-processor-run.yml
```



```
ansible-playbook -i hosts/localhost ready-deployment.yml
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts horizon-reconfigure.yml
```

## 6.5 Browser configuration

To enable your web browser to present a certificate to the Horizon dashboard upon login, you first need to import the certificate. The steps to complete this action will vary from browser to browser. Please refer to your browser's documentation for specific instructions.

1. Import the desired certificate into your web browser's certificate store.
2. After importing the certificate, verify that it appears in your browser's certificate manager.

## 6.6 User accounts

For the Keystone service to use X.509 certificates to grant users access to Horizon, there must be a Keystone user account associated with each certificate. Keystone associates user accounts with certificates by matching the common name (CN) and organization (O) of a presented certificate with the username and domain of an existing Keystone user.

When an X.509 certificate is presented to Horizon for authentication/authorization, Horizon passes the certificate information along to the Keystone service. Keystone attempts to match the CN and O of the certificate with the username and domain of an existing local user account. For this operation to be successful, there must be a Keystone user account and domain that match the CN and O of the certificate.

For example, if a user named Sam presents a certificate to Horizon with the following information,

- CN = sam
- O = EXAMPLE

Then there must be an existing Keystone user account with the following values,

- Username = sam
- Domain = EXAMPLE

Further, Sam's client certificate must have been issued by one of the certificate issuers listed in the **any\_one\_of** field in the `x509auth_mapping.json` file.

Also, when creating a local Keystone user, you must assign the user account a project scope. Without a project scope, the authorization portion of the sign-on process will fail.

The following steps illustrate how to use the CLI to create a domain, create and manage a user, and assign a permissions role to the new user.

1. Create a new domain, named `EXAMPLE`.

```
openstack domain create EXAMPLE
```

2. Create a new project named `xyz`, under the `EXAMPLE` domain.

```
openstack project create --domain EXAMPLE xyz
```

3. Create a new user named `Sam` in the `EXAMPLE` domain. Set the password and email for the new account.

```
openstack user create --domain EXAMPLE --password pass \  
--email sam@example.com --enable sam
```

4. Create a new role named `role1`.

```
openstack role create role1
```

5. Grant the new role, `role1` to the new user `Sam` from the `EXAMPLE` domain. Note that both the user account and domain must be referenced by their unique ID numbers rather than their friendly names.

```
openstack role add --user 04f3db9e7f3f45dc82e1d5f20b4acfcc \  
--domain 6b64021839774991b5e0df16077f11eb role1
```

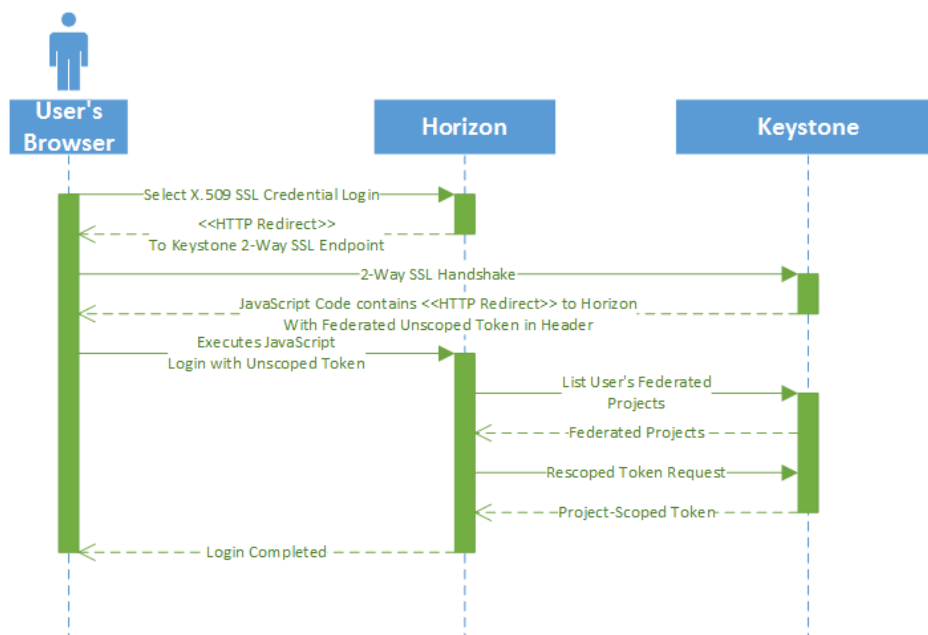
6. Add the user `Sam` to the newly-created project from [Step 2](#). Note that the project and user account must be referenced by their respective unique ID numbers rather than their friendly names.

```
openstack role add --project 4e2ad14406b247c7a9fc0a48c0b1713e \  
--user 04f3db9e7f3f45dc82e1d5f20b4acfcc role1
```

## 6.7 How it works

The SSL authentication and authorization process is detailed in the following steps.

1. User directs a web browser to the SUSE OpenStack Cloud Horizon login landing page.
2. The user selects the "X.509 Certificate" login option from the dropdown menu.
3. Horizon responds with an HTTP 302 redirect, redirecting the browser to the SSL-protected Keystone (federated) authentication endpoint.
4. The browser then prompts user to select the certificate to use for the login (if there is more than one certificate for the given trusted Certificate Authority (CA)).
5. The web browser establishes a 2-way SSL handshake with the Keystone service.
6. Keystone, utilizing federation mapping, maps the user to a federated persona and issues an (federated) unscoped token.
7. The token is then passed to the browser, along with JavaScript code that redirects the browser back to the Horizon dashboard.
8. The browser then logs into the Horizon dashboard using the newly issued unscoped token to authenticate the user.
9. Horizon queries the Keystone service for the list of federated projects the authenticated user has access to.
10. Horizon then rescopes the token to the first project, granting the user authorization.
11. The login process is completed.



## 7 Transport Layer Security (TLS) Overview

The Transport Layer Security (TLS) protocol, successor to SSL, provides the mechanisms to ensure authentication, non-repudiation, confidentiality, and integrity of user communications to and between the SUSE OpenStack Cloud services from internal and public endpoints.

OpenStack endpoints are HTTP (REST) services providing APIs to other OpenStack services on the management network. All traffic to OpenStack services coming in on the public endpoints and some traffic between services can be secured using TLS connections.

In SUSE OpenStack Cloud 8, the following are enabled for TLS

- API endpoints in the internal and admin VIPs can now be secured by TLS.
- API endpoints can be provided with their own certificates (this is shown in the model examples) or they can simply use the default certificate.
- The Barbican key management service API can be secured by TLS from the load balancer to the service endpoint.
- You can add multiple trust chains (certificate authority (CA) certificates).
- Fully qualified domain names (FQDNs) can be used for public endpoints and now they can be changed. The external name in the input model files (in `~/openstack/my_cloud/definition/data/network_groups.yml`) is where the domain name is indicated and changed.
- There are two monitoring alarms specific to certificates, 14-days to certificate expiration and 1-day to expiration.
- TLS can be turned off/on for individual endpoints.

### 7.1 SUSE OpenStack Cloud 8 clean install vs. upgrade

Clean install: all TLS-encrypted services are already listed under `tls-components` in `network_groups.yml`

You just have to:

- Add your self-signed CA cert and server cert (for testing)
- Or add your public (or company) CA-signed server cert and the public (or company) CA cert (for production)

Upgrade: you don't have TLS enabled already on the internal endpoints so you need to

- Add your self-signed CA cert and server cert (for testing)
- Or add your public (or company) CA-signed server cert and the public (or company) CA cert (for production)
- Add all the services to `tls-components` in `network_groups.yml`

For information on enabling and disabling TLS, see [Section 7.2, "TLS Configuration"](#).

For instructions on installing certificates, see [Section 7.2, "TLS Configuration"](#).

## 7.2 TLS Configuration

In SUSE OpenStack Cloud 8, you can provide your own certificate authority and certificates for internal and public virtual IP addresses (VIPs), and you should do so for any production cloud. The certificates automatically generated by SUSE OpenStack Cloud are useful for testing and setup, but you should always install your own for production use. Certificate installation is discussed below.

### 7.2.1 Using the Default My Public Cert

Read the following if you are using the default `cert-name: my-public-cert` in your model.

The bundled test cert for public endpoints, located at `~/openstack/my_cloud/config/tls/certs/my-public-cert`, is now expired but was left in the product in case you changed the content with your valid cert. Please verify if the certificate is expired and generate your own by following the guidelines further down on this page or by using a generic instruction from the web.

You can verify the expiry by running this command:

```
openssl x509 -in ~/openstack/my_cloud/config/tls/certs/my-public-cert -noout -enddate  
notAfter=Feb 12 01:18:46 2019 GMT
```

### 7.2.2 Certificate Terms

Before you begin, the following list of terms will be helpful when generating and installing certificates.

## Openstack-generated public CA

An OpenStack-generated public CA ([`openstack\_frontend\_cacert.crt`](#)) is available for you in [`/usr/local/share/ca-certificates`](#).

## Fully qualified domain name (FQDN) of the public VIP

The registered domain name. A FQDN is not mandatory. It is valid to have no FQDN and use IP addresses instead. You can use FQDNs on public endpoints, and you may change them whenever the need arises.

## Certificate authority (CA) certificate

Your certificates must be signed by a CA, such as your internal IT department or a public certificate authority. For this example we will use a self-signed certificate.

## Server certificate

It is easy to confuse server certificates and CA certificates. Server certificates reside on the server and CA certificates reside on the client. A server certificate affirms that the server that sent it serves a set of IP addresses, domain names, and set of services. A CA certificate is used by the client to authenticate this claim.

## SAN (subject-alt-name)

The set of IP addresses and domain names in a server certificate request: A template for a server certificate.

## Certificate signing request (CSR)

A blob of data generated from a certificate request and sent to a CA, which would then sign it, produce a server certificate, and send it back.

## External VIP

External virtual IP address

## Internal VIP

Internal virtual IP address

The major difference between an external VIP certificate and an internal VIP certificate is that the internal VIP has approximately 40 domain names in the SAN. This is because each service has a different domain name in SUSE OpenStack Cloud 8. So it is unlikely that you can create an internal server certificate before running the configuration processor. But after a configuration processor run, a certificate request would be created for each of your cert-names.

### 7.2.3 Configuring TLS in the input model

For this example certificate configuration, let's assume there's no FQDN for the external VIP and that you're going to use the default IP address provided by SUSE OpenStack Cloud 8. Let's also assume that for the internal VIP you will use the defaults as well. If you were to call your certificate authority "example-CA," the CA certificate would then be called "example-CA.crt" and the key would be called "example-CA.key." In the following examples, the external VIP certificate will be named "example-public-cert" and the internal VIP certificate will be named "example-internal-cert."

Any time you make a cert change when using your own CA:

- You should use a distinct name from those already existing in `config/tls/cacerts`. This also means you should not *reuse* your CA names. You should use unique and distinguishable names such as `MyCompanyXYZ_PrivateRootCA.crt`. A new name is what indicates that a file is new or changed, so reusing a name means that the file is not considered changed even its contents have changed.
- You should not remove any existing CA files from `config/tls/cacerts`.
- If you want to remove an existing CA use the following steps:

1. Remove the file.
2. Then run:

```
ansible -i hosts/verb_hosts FND-STN -a 'sudo keytool -delete -alias debian:<filename  
to remove> \  
-keystore /usr/lib/jvm/java-7-openjdk-amd64/jre/lib/security/cacerts -storepass  
changeit'
```

#### Important

Be sure to install your own certificate for all production clouds after installing and testing your cloud. If you ever want to test or troubleshoot later, you will be able to revert to the sample certificate to get back to a stable state for testing.

#### Note

Unless this is a new deployment, do not update both the certificate and the CA together. Add the CA first and then run a site deploy. Then update the certificate and run `tls-reconfigure`, `FND-CLU-stop`, `FND-CLU-start` and then `hlm-reconfigure`. If a playbook has



failed, rerun it with `-vv` to get detailed error information. The configure, HAProxy restart, and reconfigure steps are included below. If this is a new deployment and you are adding your own certs/CA before running `site.yml` this caveat does not apply.

You can add your own certificate by following the instructions below. All changes must go into the following file:

```
~/openstack/my_cloud/definition/data/network_groups.yml
```

The entries for TLS for the internal and admin load balancers are:

```
- provider: ip-cluster
  name: lb
  tls-components:
  - default
  components:
  # These services do not currently support TLS so they are not listed
  # under tls-components
  - nova-metadata
  roles:
  - internal
  - admin
  cert-file: openstack-internal-cert
  # The openstack-internal-cert is a reserved name and
  # this certificate will be autogenerated. You
  # can bring in your own certificate with a different name

  # cert-file: customer-provided-internal-cert
  # replace this with name of file in "config/tls/certs/"
```

The configuration processor will also create a request template for each named certificate under `info/cert_reqs/`, which looks like:

```
info/cert_reqs/customer-provided-internal-cert
```

## 7.2.4 Generating and Signing Certificates

These request templates contain the subject `Alt-names` that the certificates need. You can add to this template before generating your certificate signing request.

You would then send the CSR to your CA to be signed, and once you receive the certificate, place it in `config/tls/certs`

When you bring in your own certificate, you may want to bring in the trust chains (or CA certificate) for this certificate. This is usually not required if the CA is a public signer that is typically bundled with the operating system. However, we suggest you include it anyway by copying the file into the directory `config/cacerts/`.

## 7.2.5 User-provided certificates and trust chains

SUSE OpenStack Cloud generates its own internal certificates but is designed to allow you to bring in your own certificates for the VIPs. Here is the general process.

1. You must have a server certificate and a CA certificate to go with it (unless the signer is a public CA and it's already bundled with most distributions).
2. You must decide the names of the server certificates and configure the `network_groups.yml` file in the input model such that each load balancer provider has at least one cert-name associated with it.
3. Run the configuration processor. You may or may not have the certificate file at this point. The configuration processor would create certificate request file artefacts under `info/cert_reqs/` for each of the cert-name(s) in the `network_groups.yml` file. While there's no special reason to use the request file created for an external endpoint VIP certificate, it is important to use the request files created for internal certificates since the canonical names for the internal VIP as there can be many of them and they will be service specific. Each of these needs to be in the Subject Alt Names attribute of the certificate.
4. Create a certificate signing request for this request file and send it to your internal CA or a public CA to get it certified and issued with a certificate. You will now have a server certificate and possibly a trust chain or CA certificate.
5. Upload to the Cloud Lifecycle Manager. Server certificates should be added to `config/tls/certs` and CA certificates should be added to `config/tls/cacerts`. The file extension should be `.crt` for the CA certificate to be processed by SUSE OpenStack Cloud. Detailed steps are next.

## 7.2.6 Edit the Input Model to Include Your Certificate Files

1. Edit the load balancer configuration in openstack/my\_cloud/definition/data/network\_groups.yml:

```
load-balancers:
  - provider: ip-cluster
    name: lb
    tls-components:
      - default
    components:
      - cassandra
      - nova-metadata
    roles:
      - internal
      - admin
    cert-file: example-internal-cert #<<<----- Certificate name for the internal VIP

  - provider: ip-cluster
    name: extlb
    external-name: myardana.test #<<<----- Use just IP for the external VIP in this example
    tls-components:
      - default
    roles:
      - public
    cert-file: example-public-cert #<<<----- Certificate name for the external VIP
```

2. Commit your changes to the local git repository and run the configuration processor:

```
cd ~/openstack/ardana/ansible
git add -A
git commit -m "changed VIP certificates"
ansible-playbook -i hosts/localhost config-processor-run.yml
```

3. Verify that certificate requests have been generated by the configuration processor for every certificate file configured in the networks\_groups.yml file. In this example, there are two files, as shown from the list command:

```
ls ~/openstack/my_cloud/info/cert_reqs
example-internal-cert
example-public-cert
```

## 7.2.7 Generating a Self-signed CA



### Note

In a production setting you will not perform this step. You will use your company's CA or a valid public CA.

This section demonstrates to how you can create your own self-signed CA and then use this CA to sign server certificates. This CA can be your organization's IT internal CA that is self-signed and whose CA certificates are deployed on your organization's machines. This way the server certificate becomes legitimate.

1. Copy the commands below to the command line and execute. This will cause the two files to be created: example-CA.key and example-CA.crt.

```
export EXAMPLE_CA_KEY_FILE='example-CA.key'
export EXAMPLE_CA_CERT_FILE='example-CA.crt'
openssl req -x509 -batch -newkey rsa:2048 -nodes -out "${EXAMPLE_CA_CERT_FILE}" \
-keyout "${EXAMPLE_CA_KEY_FILE}" \
-subj "/C=UK/O=hp/CN=YourOwnUniqueCertAuthorityName" \
-days 365
```

You can tweak the subj and days settings above to meet your needs, or to test. For instance, if you want to test what happens when a CA expires, you can set 'days' to a very low value.

2. Select the configuration processor-generated request file from info/cert\_reqs/:

```
cat ~/openstack/my_cloud/info/cert_reqs/example-internal-cert
```

3. Copy this file to your working directory and append a .req extension to it.

```
cp ~/openstack/my_cloud/info/cert_reqs/example-internal-cert \
example-internal-cert.req
```

### EXAMPLE 7.1: CERTIFICATE REQUEST FILE

The certificate request file should look similar to the following example:

```
[req]
distinguished_name = req_distinguished_name
req_extensions = v3_req
prompt = no
```

```
[ req_distinguished_name ]
CN = "openstack-vip"

[ v3_req ]
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[ alt_names ]
DNS.1 = "deployerinccloud-ccp-c0-m1-mgmt"
DNS.2 = "deployerinccloud-ccp-vip-CEI-API-mgmt"
DNS.3 = "deployerinccloud-ccp-vip-CND-API-mgmt"
[...]
DNS.47 = "192.168.245.5"
IP.1 = "192.168.245.5"

=====end of certificate request file.
```



## Note

In the case of a public VIP certificate, please add all the FQDNs you want it to support. Currently, SUSE OpenStack Cloud does not add the hostname for the external-name specified in `network_groups.yml` to the certificate request file. However, you can add it to the certificate request file manually. Here we assume that `myopenstack.test` is your external-name. In that case you would add this line (to the full sample certificate request file shown in [Example 7.1, "Certificate request file"](#)):

```
DNS.48 = "myopenstack.test"
```



## Note

Any attempt to use IP addresses rather than FQDNs in certificates must use subject alternate name entries that list both the IP address (needed for Google) and DNS with an IP (needed for a Python bug workaround). Failure to create the certificates in this manner will cause future installations of Go-based tools (such as Cloud Foundry, Stackato and other PaaS components) to fail.

In the case of a public VIP certificate, add all the FQDNs you want it to support. Openstack does not add the hostname for the external-name specified in `network_groups.yml` to the certificate request file. However, you can add it to the certificate request file manually. Assume that `myopenstack.test` is your external-name. In that case you would add this line to the full sample certificate request file shown above.

```
DNS.48 = "myopenstack.test"
```



## Note

Any attempt to use IP addresses rather than FQDNs in certificates must use subject alternate name entries that list both the IP address (needed for Google) and DNS with an IP (needed for a Python bug workaround). Failure to create the certificates in this manner will cause future installations of Go-based tools (such as Cloud Foundry, Stackato and other PaaS components) to fail.

## 7.2.8 Generate a Certificate Signing Request



## Note

In a production setting you will not perform this step. You will use your company's CA or a valid public CA.

Now that you have a CA and a certificate request file, it's time to generate a CSR.



## Note

Please use a unique CN for your example Certificate Authority and do not install multiple CA certificates with the same CN into your cloud.

```
export EXAMPLE_SERVER_KEY_FILE='example-internal-cert.key'
export EXAMPLE_SERVER_CSR_FILE='example-internal-cert.csr'
export EXAMPLE_SERVER_REQ_FILE=example-internal-cert.req
openssl req -newkey rsa:2048 -nodes -keyout "$EXAMPLE_SERVER_KEY_FILE" \
-out "$EXAMPLE_SERVER_CSR_FILE" -extensions v3_req -config "$EXAMPLE_SERVER_REQ_FILE"
```

In production you would usually send the generated `example-internal-cert.csr` file to your IT department. But in this example you are your own CA, so sign and generate a server certificate.

## 7.2.9 Generate a Server Certificate



### Note

In a production setting you will not perform this step. You will send the CSR created in the previous section to your company CA or a to a valid public CA and have them sign and send you back the certificate.

This section demonstrates how you would use the self-signed CA that you created earlier to sign and generate a server certificate. A server certificate is essentially a signed public key, the signer being a CA and trusted by a client. When you install this signed CA's certificate (called CA certificate or trust chain) on the client machine, you are telling the client to trust this CA, and to implicitly trust any server certificates that are signed by this CA. This creates a trust anchor.

### CA configuration file

When the CA signs the certificate, it uses a configuration file that tells it to verify the CSR. Note that in a production scenario the CA takes care of this for you.

1. Create a file called `openssl.cnf` and add the following contents to it.

```
# Copyright 2010 United States Government as represented by the
# Administrator of the National Aeronautics and Space Administration.
# All Rights Reserved.
#...

# OpenSSL configuration file.
#

# Establish working directory.

dir = .

[ ca ]
default_ca = CA_default

[ CA_default ]
serial = $dir/serial
database = $dir/index.txt
new_certs_dir = $dir/
certificate = $dir/cacert.pem
private_key = $dir/cakey.pem
unique_subject = no
default_crl_days = 365
```

```

default_days = 365
default_md = md5
preserve = no
email_in_dn = no
nameopt = default_ca
certopt = default_ca
policy = policy_match
copy_extensions = copy

[ policy_match ]
countryName = optional
stateOrProvinceName = optional
organizationName = optional
organizationalUnitName = optional
commonName = supplied
emailAddress = optional

[ req ]
default_bits = 1024 # Size of keys
default_keyfile = key.pem # name of generated keys
default_md = md5 # message digest algorithm
string_mask = nombstr # permitted characters
distinguished_name = req_distinguished_name
req_extensions = v3_req
x509_extensions = v3_ca

[ req_distinguished_name ]
# Variable name Prompt string
#-----
0.organizationName = Organization Name (company)
organizationalUnitName = Organizational Unit Name (department, division)
emailAddress = Email Address
emailAddress_max = 40
localityName = Locality Name (city, district)
stateOrProvinceName = State or Province Name (full name)
countryName = Country Name (2 letter code)
countryName_min = 2
countryName_max = 2
commonName = Common Name (hostname, IP, or your name)
commonName_max = 64

[ v3_ca ]
basicConstraints = CA:TRUE
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer:always
subjectAltName = @alt_names

```



```
[ v3_req ]
basicConstraints = CA:FALSE
subjectKeyIdentifier = hash

[ alt_names ]

##### end of openssl.cnf #####
```

2. Sign the server certificate with your CA. Copy the commands below to the command line and execute. This will cause the file, example-internal-cert.crt, to be created.

```
export EXAMPLE_SERVER_CERT_FILE='example-internal-cert.crt'
export EXAMPLE_SERVER_CSR_FILE='example-internal-cert.csr'
export EXAMPLE_CA_KEY_FILE='example-CA.key'
export EXAMPLE_CA_CERT_FILE='example-CA.crt'

touch index.txt
openssl rand -hex -out serial 6

openssl ca -batch -notext -md sha256 -in "$EXAMPLE_SERVER_CSR_FILE" \
-cert "$EXAMPLE_CA_CERT_FILE" \
-keyfile "$EXAMPLE_CA_KEY_FILE" \
-out "$EXAMPLE_SERVER_CERT_FILE" \
-config openssl.cnf -extensions v3_req
```

3. Concatenate both the server key and certificate in preparation for uploading to the Cloud Lifecycle Manager.

```
cat example-internal-cert.key example-internal-cert.crt > example-internal-cert
```

4. You have only created the internal-cert in this example. Repeat the above sequence for example-public-cert. Make sure you use the appropriate certificate request generated by the configuration processor.

## 7.2.10 Upload to the Cloud Lifecycle Manager

1. The two files created from the example above will need to be uploaded to the Cloud Lifecycle Manager and copied into config/tls:
  - example-internal-cert
  - example-CA.crt

2. On the Cloud Lifecycle Manager, execute the following copy commands. If you created an external cert, copy that in a similar manner.

```
cp example-internal-cert ~/openstack/my_cloud/config/tls/certs/  
cp example-CA.crt ~/openstack/my_cloud/config/tls/cacerts/
```

3. Log into the Cloud Lifecycle Manager node. Save and commit the changes to the local Git repository:

```
cd ~/openstack/ardana/ansible  
git add -A  
git commit -m "updated certificate and CA"
```

4. Rerun the `config-processor-run` playbook, and run `ready-deployment.yml`:

```
cd ~/openstack/ardana/ansible  
ansible-playbook -i hosts/localhost config-processor-run.yml  
ansible-playbook -i hosts/localhost ready-deployment.yml
```

5. If you receive any prompts, enter the required information.



## Note

For automated installation (for example, CI) you can specify the required passwords on the Ansible command line. For example, the command below will disable encryption by the configuration processor:

```
ansible-playbook -i hosts/localhost config-processor-run.yml -e encrypt="" -e  
rekey=""
```

6. Run this series of runbooks to complete the deployment:

```
cd ~/scratch/ansible/next/ardana/ansible  
ansible-playbook -i hosts/verb_hosts tls-reconfigure.yml  
ansible-playbook -i hosts/verb_hosts FND-CLU-stop.yml  
ansible-playbook -i hosts/verb_hosts FND-CLU-start.yml  
ansible-playbook -i hosts/verb_hosts monasca-stop.yml  
ansible-playbook -i hosts/verb_hosts monasca-start.yml  
ansible-playbook -i hosts/verb_hosts ardana-reconfigure.yml
```

## 7.2.11 Configuring the Cipher Suite

By default, the cipher suite is set to: `HIGH:!aNULL:!eNULL:!DES:!3DES`. This setting is recommended in the [OpenStack documentation \(http://docs.openstack.org/security-guide/secure-communication/introduction-to-ssl-and-tls.html\)](http://docs.openstack.org/security-guide/secure-communication/introduction-to-ssl-and-tls.html). You may override this by editing `config/haproxy/defaults.yml`. The parameters can be found under the `haproxy_globals` list.

```
- "ssl-default-bind-ciphers HIGH:!aNULL:!eNULL:!DES:!3DES"
- "ssl-default-server-ciphers HIGH:!aNULL:!eNULL:!DES:!3DES"
```

Make the changes as needed. Keep the two options identical.

## 7.2.12 Testing

You can determine if an endpoint is behind TLS by running the following command, which probes a Keystone identity service endpoint that is behind TLS:

```
tux > echo | openssl s_client -connect 192.168.245.5:5000 | \
  openssl x509 -fingerprint -noout
depth=0 CN = openstack-vip
verify error:num=20:unable to get local issuer certificate
verify return:1
depth=0 CN = openstack-vip
verify error:num=27:certificate not trusted
verify return:1
depth=0 CN = openstack-vip
verify error:num=21:unable to verify the first certificate
verify return:1
DONE
SHA1 Fingerprint=C6:46:1E:59:C6:11:BF:72:5E:DD:FC:FF:B0:66:A7:A2:CC:32:1C:B8
```

The next command probes a MariaDB endpoint that is not behind TLS:

```
echo | openssl s_client -connect 192.168.245.5:3306 | openssl x509 -fingerprint -noout
140448358213264:error:140770FC:SSL routines:SSL23_GET_SERVER_HELLO:unknown protocol:s23_clnt.c:795:
unable to load certificate
140454148159120:error:0906D06C:PEM routines:PEM_read_bio:no start line:pem_lib.c:703:Expecting: TRUSTED
CERTIFICATE
```

## 7.2.13 Verifying That the Trust Chain is Correctly Deployed

You can determine if the trust chain is correctly deployed by running the following commands:

```
tux > echo | openssl s_client -connect 192.168.245.9:5000 2>/dev/null \
```

```
| grep code
Verify return code: 21 (unable to verify the first certificate)
tux > echo | openssl s_client -connect 192.168.245.9:5000 -CAfile \
/usr/local/share/ca-certificates/openstack_frontend_cacert.crt 2>/dev/null \
| grep code
Verify return code: 0 (ok)
```

The first command produces error 21, which is then fixed by providing the the CA certificate file. This verifies that the CA certificate matches the server certificate.

## 7.2.14 Turning TLS on or off

You should leave TLS enabled in production. However, if you need to disable it for any reason, you must change `tls-components` to `components` in `network_groups.yml` (as shown earlier) and comment out the cert-file. Additionally, if you have a `network_groups.yml` file from a previous installation, TLS will not be enabled unless you change `components` to `tls-components` in that file. By default, Horizon is configured with TLS in the input model. You should not disable TLS in the input model for Horizon as that is a public endpoint and is required. Additionally, you should keep all services behind TLS, but using the input model file `network_groups.yml` you may turn TLS off for a service for troubleshooting or debugging. TLS should always be enabled for production environments.

If you are using an example input model on a clean install, all supported TLS services will be enabled before deployment of your cloud. If you want to change this setting later, such as when upgrading, you can change the input model and reconfigure the system. The process is as follows:

1. Edit the input model `network_groups.yml` file appropriately as described above, changing `tls-components` to `components`.
2. Commit the changes to the Git repository:

```
cd ~/openstack/ardana/ansible/
git add -A
git commit -m "TLS change"
```

3. Change directories again and run the configuration processor and ready deployment playbooks:

```
ansible-playbook -i hosts/localhost config-processor-run.yml
ansible-playbook -i hosts/localhost ready-deployment.yml
```

4. Change directories again and run the reconfigure playbook:

```
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts ardana-reconfigure.yml
```

## 7.3 Enabling TLS for MySQL Traffic

MySQL traffic can be encrypted using TLS. For completely new SUSE OpenStack Cloud deployments using the supplied input model example files, you will have to uncomment the commented entries for `tls-component-endpoints:`. For upgrades from a previous version, you will have to add the entries to your input model files if you have not already done so. This topic explains how to do both.

### 7.3.1 Enabling TLS on the database server for client access

1. Edit `network_groups.yml` to either add `mysql` under `tls-component-endpoints` in your existing file from a previous version, or uncomment it if installing from scratch.

```
tls-component-endpoints:
  - mysql
```

2. After making the necessary changes, commit the changed file to git and run the `config-processor-run` and `reconfigure Ansible` playbooks:

```
cd ~/openstack
git add -A
git commit -m "My changed config"
cd ~/openstack/ardana/ansible/
ansible-playbook -i hosts/localhost config-processor-run.yml -e encrypt="<encryption key>" -e rekey=""
ansible-playbook -i hosts/localhost ready-deployment.yml
cd ~/scratch/ansible/next/ardana/ansible
```

3. Next, either run `site.yml` if you're installing a new system:

```
ansible-playbook -i hosts/verb_hosts site.yml
```

4. or `ardana-reconfigure` if you are reconfiguring an existing one:

```
ansible-playbook -i hosts/verb_hosts ardana-reconfigure.yml
```

## 7.3.2 MySQL replication over TLS

MySQL replication over TLS is disabled. This is true even if you followed the instruction to turn on MySQL TLS in the previous section. Those steps turn on the service interactions to the database.

### Turning on MySQL replication over TLS



#### Note

Using TLS connections for MySQL replication will incur a performance cost.

You should have already enabled TLS for MySQL client interactions in the previous section. If not, read [Section 7.3.1, “Enabling TLS on the database server for client access”](#).

TLS for MySQL replication is not turned on by default. Therefore, you will need to follow a manual process. Again, the steps are different for new systems and upgrades.

## 7.3.3 Enabling TLS for MySQL replication on a new deployment

1. Log into the Cloud Lifecycle Manager node and before running the config processor edit the `~/openstack/my_cloud/config/galera/defaults.yml` file.
2. Search for `mysql_gcomms_bind_tls`. You should find this section:

```
# TLS disabled for cluster
#mysql_gcomms_bind_tls: "{{ host.bind['FND_MDB'].mysql_gcomms.tls }}"
mysql_gcomms_bind_tls: False
```

3. Uncomment the appropriate line so the file looks like this:

```
# TLS disabled for cluster
mysql_gcomms_bind_tls: "{{ host.bind['FND_MDB'].mysql_gcomms.tls }}"
mysql_gcomms_bind_tls: False
```

4. Follow the steps to deploy or reconfigure your cloud: [Step 2](#) in [Section 7.3.1, “Enabling TLS on the database server for client access”](#).

## 7.3.4 Enabling TLS for MySQL replication on an existing system

If your cluster is already up, perform these steps to enable MySQL replication over TLS:

1. Edit the following two files: `~/openstack/my_cloud/config/galera/default-faults.yml` and `~/scratch/ansible/next/ardana/ansible/roles/FND-MDB/default-faults/main.yml`. Note that these files are identical. The first is a master file and the second is a scratch version that is used for the current deployment. Make the same changes as explained in [Section 7.3.3, "Enabling TLS for MySQL replication on a new deployment"](#).
2. Then run the following command:

```
ansible-playbook -i hosts/verb_hosts tls-galera-reconfigure.yml
```

After this your MySQL should come up and replicate over TLS. You need to follow this section again if you ever want to switch TLS off for MySQL replication. You also must repeat these steps if any lifecycle operation changes the `mysql_gcomms_bind_tls` option.

## 7.3.5 Testing whether a service is using TLS

Almost all services that have a database are able to communicate over TLS. You can test whether a service, in this example the Identity service (Keystone), is communicating with MySQL over TLS by executing the following steps:

1. Log into the Cloud Lifecycle Manager as root and run the `mysql` command.

```
root@<server>:~# mysql
```

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

2. Run:

```
mysql> select * from information_schema.user_statistics where user='keystone'\G
```

3. Note the results. `TOTAL_SSL_CONNECTIONS` should not be zero:

```
***** 1. row *****
      USER: keystone
TOTAL_CONNECTIONS: 316
CONCURRENT_CONNECTIONS: 0
   CONNECTED_TIME: 905790
      BUSY_TIME: 205
```

```

          CPU_TIME: 141
        BYTES_RECEIVED: 197137617
          BYTES_SENT: 801964
    BINLOG_BYTES_WRITTEN: 0
          ROWS_FETCHED: 972421
          ROWS_UPDATED: 6893
        TABLE_ROWS_READ: 1025866
        SELECT_COMMANDS: 660209
        UPDATE_COMMANDS: 3039
          OTHER_COMMANDS: 299746
        COMMIT_TRANSACTIONS: 0
    ROLLBACK_TRANSACTIONS: 295200
        DENIED_CONNECTIONS: 0
          LOST_CONNECTIONS: 83
          ACCESS_DENIED: 0
          EMPTY_QUERIES: 71778
        TOTAL_SSL_CONNECTIONS: 298
1 row in set (0.00 sec)

mysql>

```

## 7.4 Enabling TLS for RabbitMQ Traffic

RabbitMQ traffic can be encrypted using TLS. To enable it, you will have to add entries for `tls-component-endpoints` in your input model files if you have not already done so. This topic explains how.

1. Edit `openstack/my_cloud/definition/data/network_groups.yml`, adding `rabbitmq` to the `tls-component-endpoints` section:

```

tls-component-endpoints:
  - barbican-api
  - mysql
  - rabbitmq

```

2. Commit the changes:

```

cd ~/openstack
git add -A
git commit -m "My changed config"

```

3. Then run the typical deployment steps:

```

cd ~/openstack/ardana/ansible/

```



```
ansible-playbook -i hosts/localhost config-processor-run.yml -e encrypt="<encryption key>" -e rekey=""
ansible-playbook -i hosts/localhost ready-deployment.yml
```

#### 4. Change directories:

```
cd ~/scratch/ansible/next/ardana/ansible
```

#### 5. Then for a fresh TLS install run:

```
ansible-playbook -i hosts/verb_hosts site.yml
```

#### 6. Or, to reconfigure an existing system run:

```
ansible-playbook -i hosts/verb_hosts ardana-reconfigure.yml
```

### 7.4.1 Testing

On the one of the rabbitmq nodes you can list the clients and their TLS status by running:

```
$ sudo rabbitmqctl -q list_connections ssl state ssl_protocol user name
```

You will see output like this where true indicates the client is using TLS for the connection, and false, as shown here, indicates the connection is over TCP:

```
Listing connections ...

rmq_barbican_user      false
```

Other indicators will be `rabbit_use_ssl = True` in the Oslo messaging section of client configurations. The list of clients that support TLS are as follows:

- Barbican
- Ceilometer
- Cinder
- Designate
- Eon
- Glance

- Heat
- IroniC
- Keystone
- Monasca
- Neutron
- Nova
- Octavia

## 7.5 Troubleshooting TLS

### 7.5.1 Troubleshooting TLS certificate errors when running playbooks with a limit

Has the deployer been restarted after the original site installation or is this a new deployer? If so, TLS certificates need to be bootstrapped before a playbook is run with limits. You can do this by running the following command.

```
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts tls-reconfigure.yml --limit TLS-CA
```

### 7.5.2 Certificate Update Failure

In general, if a certificate update fails, it is because of the following: Haproxy hasn't restarted or the Trust chain is not installed. This is the certificate of the CA that signed the server certificate.

### 7.5.3 Troubleshooting trust chain installation

It is important to note that while SUSE OpenStack Cloud 8 allows you to add new trust chains, it would be better if you add all the required trust chains during the initial deploy. Trust chain changes can impact services.

However, this does not apply to certificates. There is a certificate-related issue whereby haproxy is not restarted if certificate content has been changed but the certificate file name remained the same. If you are having issues and you have replaced the content of existing CA file with new content, create another CA file with a new name. Also make sure the CA file has a .crt extension. Do not update both certificate and the CA together. Add the CA first and then run a site deploy. Then update the certificate and run `tls-reconfigure`, `FND-CLU-stop`, `FND-CLU-start` and then `ardana-reconfigure`. If you know which playbook failed, rerun it with `-vv` to get detailed error information. The configure, HAproxy restart, and reconfigure steps are included in [Section 7.2, “TLS Configuration”](#).

You can run the following commands to see if client libraries see the CA you've added:

```
~/scratch/ansible/next/ardana/ansible$ ansible -i hosts/verb_hosts FND-STN -a 'sudo
keytool -list -alias \
    debian:username-internal-cacert-001.pem -keystore /usr/lib/jvm/java-7-openjdk-amd64/
jre/lib/security/cacerts -storepass changeit'
padawan-ccp-c0-m1-mgmt | FAILED | rc=1 >>
sudo: keytool: command not found

padawan-ccp-comp0001-mgmt | FAILED | rc=1 >>
sudo: keytool: command not found

padawan-ccp-comp0003-mgmt | FAILED | rc=1 >>
sudo: keytool: command not found

padawan-ccp-comp0002-mgmt | FAILED | rc=1 >>
sudo: keytool: command not found

padawan-ccp-c1-m1-mgmt | success | rc=0 >>
debian:username-internal-cacert-001.pem, May 9, 2016, trustedCertEntry,
Certificate fingerprint (SHA1):
E7:B2:6E:9E:00:FB:86:0F:E5:46:CD:B8:C5:67:13:53:4E:3D:8F:43

padawan-ccp-c1-m2-mgmt | success | rc=0 >>
debian:username-internal-cacert-001.pem, May 9, 2016, trustedCertEntry,
Certificate fingerprint (SHA1):
E7:B2:6E:9E:00:FB:86:0F:E5:46:CD:B8:C5:67:13:53:4E:3D:8F:43

padawan-ccp-c1-m3-mgmt | success | rc=0 >>
debian:username-internal-cacert-001.pem, May 9, 2016, trustedCertEntry,
Certificate fingerprint (SHA1):
E7:B2:6E:9E:00:FB:86:0F:E5:46:CD:B8:C5:67:13:53:4E:3D:8F:43
```

Java client libraries are used by Monasca, so compute nodes will not have them. So the first three errors are expected. Check that the fingerprint is correct by checking the CA:

```
~/scratch/d002-certs/t002$ openssl x509 -in example-CA.crt -noout -fingerprint
SHA1 Fingerprint=E7:B2:6E:9E:00:FB:86:0F:E5:46:CD:B8:C5:67:13:53:4E:3D:8F:43
```

If they don't match there likely was a name collision. Add the CA cert again with a new file name. If you get Monasca errors but find that the fingerprints match, then try stopping and starting Monasca.

```
ansible-playbook -i hosts/verb_hosts monasca-stop.yml
ansible-playbook -i hosts/verb_hosts monasca-start.yml
```

## 7.5.4 Troubleshooting certificates

Certificates can fail in SUSE OpenStack Cloud 8 due to the following.

- Trust chain issue. This is dealt with in the previous section
- Wrong certificate: Compare the fingerprints. If they differ, then you have a wrong certificate somewhere.
- Date range of the certificate is either in the future or expired: Check the dates and change certificates as necessary, observing the naming cautions above.
- TLS handshake fails because the client doesn't support the ciphers the server offers. It's possible that you reused a certificate created for a different network model. Make sure the request file found under `info/cert_req/` are used to create the certificate. If not, the service VIP names may not match.

## 8 SUSE® OpenStack Cloud: Preventing Host Header Poisoning

Depending on the environment and context of your SUSE OpenStack Cloud deployment, it may be advisable to configure Horizon to protect against Host header poisoning (see ref. #1 below) by using Django's `ALLOWED_HOSTS` setting (see ref. #2 below). To configure Horizon to use the `ALLOWED_HOSTS` setting, take the following steps:

1. Edit the haproxy settings to reconfigure the health check for Horizon to specify the allowed hostname(s). This needs to be done first, before configuring Horizon itself. Otherwise, if Horizon is first configured to restrict the values of the "Host" header on incoming HTTP requests, the haproxy health checks will start to fail. So, the haproxy configuration needs to be updated first, if this is being done on an existing installation.
  - a. On your Cloud Lifecycle Manager node, make a backup copy of this file and then open `/usr/share/ardana/input-model/2.0/services/horizon.yml`
  - b. Find the line that contains "option httpchk" and modify it so it reads the following way:

```
- "option httpchk GET / HTTP/1.1\r\nHOST:\ my.example.com"
# Note the escaped escape characters.
```

In this example, `my.example.com` is the hostname associated with the Horizon VIP on the external API network. However, you aren't restricted to just one allowed host. In addition, allowed hosts can contain wildcards (though not in the `horizon.yml` file; there you must have an actual resolvable hostname or a routeable IP address). However, for this change to the haproxy healthcheck, it is suggested that the hostname associated with the Horizon VIP on the external API network be used.

2. Edit the template file that will be use for Horizon's `local_settings.py` configuration file
  - a. While still on your Cloud Lifecycle Manager node, open `~/openstack/my_cloud/config/horizon/local_settings.py`.
  - b. Change the line that sets the "ALLOWED\_HOSTS" setting. This can be a list of hostnames and (V)IPs that eventually get routed to Horizon. Wildcards are supported.

```
ALLOWED_HOSTS = ['my.example.com', '*.example.net', '192.168.245.6']
```

In the above example, any HTTP request received with a hostname not matching any in this list will receive an HTTP 400 reply.

- c. Commit the change with a "git commit -a" command.

### 3. Run the configuration processor

```
cd ~/openstack/ardana/ansible
ansible-playbook -i hosts/localhost config-processor-run.yml
```

- 4. Enable the configuration: This can be done in one of a few ways: As part of a site deploy play as part of an upgrade play or by re-running the FND-CLU and Horizon deploys on an existing deployment: If modifying an existing deploy, the FND-CLU deploy will need to be run first, since changing the ALLOWED\_HOSTS setting in Horizon first will cause the default health check to fail, if it doesn't specify a "Host" header in the HTTP request sent to check the health of Horizon's Apache virtual host.

```
cd ~/openstack/ardana/ansible
ansible-playbook -i hosts/localhost ready-deployment.yml
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts horizon-deploy.yml
ansible-playbook -i hosts/verb_hosts FND-CLU-deploy.yml
```

### References:

- <https://www.djangoproject.com/weblog/2013/feb/19/security/#s-issue-host-header-poisoning> ↗
- <https://docs.djangoproject.com/en/dev/ref/settings/#allowed-hosts> ↗

## 9 Encryption of Passwords and Sensitive Data

In SUSE OpenStack Cloud, sensitive connection data is encrypted. The passwords that are encrypted include:

- Inter-service passwords generated by the configuration processor (Keystone, MariaDB, RabbitMQ and Cassandra passwords)
- Secret keys generated by the configuration processor (MariaDB cluster-id, erlang cookie for RabbitMQ, Horizon secret key, Keystone admin token)
- User-supplied passwords (IPMI passwords, Block Storage back-end passwords)

### 9.1

What is encrypted	Encryption mechanism	Is password changeable	Is encryption key changeable
Inter-service passwords and secret keys generated by the configuration processor (Keystone, MariaDB, RabbitMQ and Cassandra passwords)	Uses PyCrypto libraries & Ansible vault for encryption	No	Yes Passphrase for the encryption key will be prompted when running Ansible playbook. Can also use command <u><b>ask-ansible-pass</b></u>
User supplied passwords (IPMI passwords, Block Storage back-end passwords)	OpenSSL	Yes	Yes The environment variable <code>AR-DANA_USER_PASSWORD_ENCRYPT_KEY</code> must contain the key used to encrypt those passwords.

Other protected data:

- The SSH private key used by Ansible to connect to client nodes from the Cloud Lifecycle Manager is protected with a passphrase.
- The Swift swift-hash prefix and suffix values are encrypted.
- All of the Ansible variables generated by the configuration processor are encrypted and held in Ansible Vault.

However, if a user wants to change the encryption keys then that can be done for all categories of password and secret-keys listed below, and the processes are documented.

The ssh private key passphrase needs to be entered once before any Ansible plays are run against the cloud.

The configuration processor encryption key will be prompted for when the relevant Ansible play is run. Once the configuration processor output has been encrypted, all subsequent Ansible plays need to have `--ask-ansible-pass` added to the command line to ensure that the encryption key which is needed by Ansible is prompted for.

Finally, if user-supplied passwords have been encrypted (this process uses the OpenSSL library) then the environment variable `ARDANA_USER_PASSWORD_ENCRYPT_KEY` must contain the key used to encrypt those passwords.

In the case where the `ARDANA_USER_PASSWORD_ENCRYPT_KEY` environment variable is either null, the empty string, or not defined, then no encryption will be performed on your passwords when using the `ardanaencrypt.py` script.

The generated passwords are stored in Ansible inputs generated by the configuration processor and also in the persistent state information maintained by the configuration processor.

## 9.2 Protecting sensitive data on the Cloud Lifecycle Manager

There are a number of mechanisms that can be used to protect sensitive data such as passwords, some Ansible inputs, and the SSH key used by Ansible on the Cloud Lifecycle Manager. See the installation documents for details. Please remember the need to guard against exposure of your environment variables, which may happen through observation over the shoulder.



There are instructions included in the installation documents that show how to encrypt your data using the `ardanaencrypt.py` script. You may want to change the encryption keys used to protect your sensitive data in the future and this shows you how:

- **SSH keys** - Run the command below to change the passphrase used to protect the key:

```
ssh-keygen -f id_rsa -p
```

- **configuration processor Key** - If you wish to change an encryption password that you have already used when running the configuration processor then enter the existing password at the first prompt and the new password at the second prompt when running the configuration processor playbook. See *Book "Installing with Cloud Lifecycle Manager", Chapter 13 "Installing Mid-scale and Entry-scale KVM"* for more details.
- **IPMI passwords if encrypted with `ardanaencrypt.py`** - Rerun the utility specifying a new encryption key when prompted. You will need to enter the plain text passwords at the password prompt.

## 9.3 Interacting with Encrypted Files

Once you have enabled encryption in your environment you may have a need to interact with these encrypted files at a later time. This section will show you how.

### **ardanaencrypt.py script password encryption**

If you used the `ardanaencrypt.py` script to encrypt your IPMI or other passwords and have a need to view them later, you can do so with these steps.

You will want to ensure that the `ARDANA_USER_PASSWORD_ENCRYPT_KEY` environment variable is set prior to running these commands:

```
export ARDANA_USER_PASSWORD_ENCRYPT_KEY="<encryption_key>"
```

To view an encrypted password, you can use this command below which will prompt you for the encrypted password value. It will then output the decrypted value:

```
./ardanaencrypt.py -d
```

### **Configuration processor encryption key**

If you have used the encryption options available with the configuration processor, which uses Ansible vault, you can do so with these commands. Each of these commands will prompt you for the password you used when setting the encryption initially.

To view an encrypted file in read-only mode, use this command:

```
ansible-vault view <filename>
```

To edit an encrypted file, use this command. This allows you to edit a decrypted version of the file without the need to decrypt and re-encrypt it:

```
ansible-vault edit <filename>
```

For other available commands, use the help file:

```
ansible-vault -h
```

## 10 Encryption of Ephemeral Volumes

By default, ephemeral volumes are not encrypted. If you wish to enable this feature, you should use the following steps.



### Note

For more details about this feature, see [Ephemeral storage encryption for LVM backend \(http://specs.openstack.org/openstack/nova-specs/specs/juno/approved/lvm-ephemeral-storage-encryption.html\)](http://specs.openstack.org/openstack/nova-specs/specs/juno/approved/lvm-ephemeral-storage-encryption.html).

### 10.1 Enabling ephemeral volume encryption

Before deploying the Compute nodes you will need to change the disk configuration to create a new volume-group which will be used for your ephemeral disks. To do this, following these steps:

1. Log in to the Cloud Lifecycle Manager.
2. Add details about the volume-group you will be using for your encrypted volumes. You have two options for this, you can either create a new volume-group or add the details for an already existing volume-group.
  - a. To create a new volume-group, add the following lines to your Compute disk configuration file.

The location of the Compute disk configuration file is:

```
~/openstack/my_cloud/definition/data/disks_compute.yml
```

```
name: vg-comp
  physical-volumes:
    - /dev/sdb
```

- b. To utilize an existing volume-group you can add the following lines to your no-va.conf file, using the name of your volume-group:

```
[libvirt]
images_type = lvm
images_volume_group = <volume_group_name>
```



## Note

The requirement here is to have free space available on a volume-group. The correct disk to use and the name for the volume group will depend on your environment's needs.

3. Modify the nova.conf file for the Compute and API nodes. Verify that the following entries exist, if they do not then add them and then restart the nova-compute and nova-api services:

```
[libvirt]
images_type = lvm
images_volume_group = vg-comp

[ephemeral_storage_encryption]
key_size = 256
cipher = aes-xts-plain64
enabled = True

[keymgr]
api_class = nova.keymgr.barbican.BarbicanKeyManager

[barbican]
endpoint_template = https://192.168.245.9:9311/v1
```

To restart the services, use the following commands:

```
sudo systemctl restart nova-compute
sudo systemctl restart nova-api
```

4. Assign the role in Keystone using the CLI tool. Using the openstack client you can assign the user key-manager:creator role for the project.
5. Boot an instance with an ephemeral disk and verify that the disk is encrypted. Once the instance is active it is possible to check on the Compute node if the ephemeral disk is encrypted.

SSH into the Compute node then run the following commands:

```
sudo dmsetup status
cryptsetup -v status <name_of_ephemeral_disk>
```

## 11 Refining Access Control with AppArmor

AppArmor is a Mandatory Access Control (MAC) system as opposed to a discretionary access control system. It is a kernel-level security module for Linux that controls access to low-level resources based on rights granted via policies to a program rather than to a user role. It enforces rules at the lowest software layer (the kernel level) preventing software from circumventing resource restrictions that reside at levels above the kernel. With AppArmor, the final gatekeeper is closest to the hardware.


Controlling resource access per application versus per user role allows you to enforce rules based on specifically what a program can do versus trying to create user roles that are broad enough yet specific enough to apply to a group of users. In addition, it prevents the trap of having to predict all possible vulnerabilities in order to be secure.

AppArmor uses a hybrid of whitelisting and blacklisting rules, and its security policies are/can be cascading, permitting inheritance from different or more general policies. Policies are enforced on a per-process basis.

AppArmor also lets you tie a process to a CPU core if you want, and set process priority.

AppArmor profiles are loaded into the kernel, typically on boot. They can run in either enforcement or complain modes. In enforcement mode, the policy is enforced and policy violation attempts are reported. In complain mode, policy violation attempts are reported but not prevented.

### 11.1 AppArmor in SUSE OpenStack Cloud 8

At this time, AppArmor is not enabled by default in SUSE OpenStack Cloud 8. However, we recommend enabling it for key virtualization processes on compute nodes. For more information, see the [SUSE Security Guide on AppArmor \(https://www.suse.com/documentation/sles-12/book\\_security/data/part\\_apparmor.html\)](https://www.suse.com/documentation/sles-12/book_security/data/part_apparmor.html) .

## 12 Data at Rest Encryption

The data at rest encryption features in SUSE OpenStack Cloud 8 include the Barbican key management service for safely storing encryption keys, and Cinder volume encryption. This topic explains how to configure a back end for Barbican key storage, and how to configure Cinder volumes to be encrypted.

The Barbican service in SUSE OpenStack Cloud 8 supports two types of back ends for safely storing encryption keys:

- A native database back end
- An HSM device (KMIP + Micro Focus ESKM)

### Configuring the key management back-end key store

Using the Cloud Lifecycle Manager reconfigure playbook, you can configure one of two back ends for the Barbican key management service:

- Native database: This is the default configuration in SUSE OpenStack Cloud 8.
- KMIP + Atalla ESKM: For a KMIP device, an SSL client certificate is needed as HSM devices generally require two-way SSL for security reasons. You will need a client certificate, a client private key and client root certificate authority recognized by your HSM device.

### 12.1 Configuring KMIP and ESKM

1. To configure KMIP + Atalla ESKM in place of the default database, begin by providing certificate information by modifying the sample configuration file, barbican\_kmip\_plugin\_config\_sample.yml, on the Cloud Lifecycle Manager node:

```
~/openstack/ardana/ansible/roles/KEYMGR-API/files/samples/  
barbican_kmip_plugin_config_sample.yml
```

2. Copy this file to a temporary directory such as /tmp.
3. Edit the file to provide either client certificates as absolute file paths as shown below in bold, or by pasting certificate and key content directly into the file.



## Note

File paths take precedence over content variables if both are provided.

4. To set file path variables, open `kmip_plugin_certs.yml` for editing and setting the paths to the cert files:

```
vi /tmp/kmip_plugin_certs.yml
# File paths takes precedence over cert content if both are provided.
# Here file path refers to local filesystem path where ansible is
# executed.
client_cert_file_path: /path/to/cert/file
client_key_file_path: /path/to/key/file
client_cacert_file_path: /path/to/cacert/file
```

5. Alternatively, set the content variables by opening `/tmp/kmip_plugin_certs.yml` and copy the certificates and keys directly into the file.

```
vi /tmp/kmip_plugin_certs.yml
# Following are samples you need to replace with your
# own content here or via file path approach mentioned above.
client_cert_content: |
    -----BEGIN CERTIFICATE-----
    MIID0jCCArqgAwIBAgICAKQwDQYJKoZIhvcNAQELBQAwgZQxCzAJBgNVBAYTA1VT
    MQswCQYDVQQIEwJDTzEUMBIGA1UEBxMLRnQuIENvbGxpbmMxGDAwBgNVBAoTD0hl
    ...
    d2xldHhqUGFja2FyZDEMMAoGA1UECxMDQ1RMMRYwFAYDVQQDFA1LTULQX0xvY2Fs
    L7x0qB6Zaf3IBk0Zqf5bmMfAQoKfxww==
    -----END CERTIFICATE-----
client_key_content: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIeowIBAAKCAQEARjYVZzdsSMsk520UD1E94jl0/AZGLlsAB152dEP5E9C3mXzQ
    ZYvfApMh8PFc53gZwLBCb4joylr8mZj/e7CwCUuo1cJHR9xnhdwK3RLeRbU3dfw8
    ...
    98DmYxBio8+wQWQdiAPRRthtnvhSWL67oYACPwvWUJJ+D18HfpWCEgCmBU3a8ZHc
    AaW8rRXtMZzuujGgAbA1hpf5zllHuiG/X7/XMDVGiRALMyBbHV57
    -----END RSA PRIVATE KEY-----
client_cacert_content: |
    -----BEGIN CERTIFICATE-----
    MIIEmjCCA4KgAwIBAgIBADANBgkqhkiG9w0BAQsFADCBlDELMAkGA1UEBhMCVVMx
    CzAJBgNVBAGTAKNPMRQwEgYDVQQHEwtGdC4gQ29sbGluczEYMBYGA1UEChMPSGV3
    ...
    FAimEB/a2E+A0oxwuHmhMg0k0pDuXIWn4BW+Z6z5h1j3PFyg/CZ548Fz0X0gvXC7
    Ejpkd+5R+24HloruUV1R2EYvmlr8UMFX80og1lu+
```

```
-----END CERTIFICATE-----
```

6. Provide certificate information to the Barbican service using the `barbican-reconfigure.yml` playbook:

```
cd ~/openstack/ardana/ansible
ansible-playbook -i hosts/verb_hosts barbican-reconfigure.yml -e@/tmp/
kmp_plugin_certs.yml
```

7. Provide HSM connection credentials for the Barbican service. In this step, provide the KMIP plug-in connection details to the Barbican service: Open the file `barbican_deploy_config.yml` found here:

```
~/openstack/ardana/ansible/roles/barbican-common/vars/barbican_deploy_config.yml
```

8. Set the value of `use_kmp_secretstore_plugin` to `True` to use the KMIP plug-in or `False` to use the default secret store plugin (`store_crypto`).
9. Next, add KMIP client connection credentials and KMIP server hostname and port to `barbican_deploy_config.yml`:

```
#####
##### KMIP Plugin Configuration Section #####
#####
# Flag to reflect whether KMIP plugin is to be used as back end for
#storing secrets
use_kmp_secretstore_plugin: True
# Note: Connection username needs to match with 'Common Name' provided
# in client cert request (CSR).
barbican_kmp_username: userName barbican_kmp_password: password
barbican_kmp_port: 1234 barbican_kmp_host: 111.222.333.444
```

10. Commit the changes to git:

```
cd ~/openstack/ardana/ansible
git add -A
git commit -m "My config"
```

and run the `barbican-reconfigure.yml` playbook in the deployment area:

```
ansible-playbook -i hosts/localhost ready-deployment.yml
cd ~/scratch/ansible/next/ardana/ansible
ansible-playbook -i hosts/verb_hosts barbican-reconfigure.yml
```



## 12.2 Configuring Cinder volumes for encryption

The data-at-rest encryption model in SUSE OpenStack Cloud provides support for encrypting Cinder volumes (Volume Encryption). These encrypted volumes are protected with an encryption key that can be stored in an HSM appliance.

Assuming Barbican and Cinder services have been installed, you can configure a Cinder volume type for encryption. Doing so will create a new Cinder volume type, "LUKS," that can be selected when creating a new volume. Such volumes will be encrypted using a 256bit AES key:

```
source ~/service.osrc
openstack role add --user admin --project admin cinder_admin
cinder type-create LUKS
cinder encryption-type-create \
  --cipher aes-xts-plain64 --key_size 256 --control_location \
  front-end LUKS nova.volume.encryptors.luks.LuksEncryptor
```

+-----+-----+-----+-----+			
+-----+-----+-----+-----+			
Volume Type ID		Provider	
Cipher	Key Size	Control Location	
+-----+-----+-----+-----+			
+-----+-----+-----+-----+			
99ed804b-7ed9-41a5-9c5e-e2002e9f9bb4	nova.volume.encryptors.luks.LuksEncryptor		aes-
xts-plain64	256	front-end	
+-----+-----+-----+-----+			
+-----+-----+-----+-----+			

You should now be able to create a new volume with the type LUKS, which will request a new key from Barbican. Once created, you can attach the new volume to an instance:

```
cinder create --display-name testVolumeEncrypted --volume-type LUKS --availability-zone nova 1
```

The volume list (`cinder show` with the volume ID) should now show that you have a new volume and that it is encrypted.

```
cinder show 2ebf610b-98bf-4914-aeel-9b866d7b1897
```

+-----+-----+-----+-----+	
Property	Value
+-----+-----+-----+-----+	
attachments	[]
availability_zone	nova
bootable	false
consistencygroup_id	None

created_at	2016-03-04T00:17:45.000000
description	None
encrypted	True
id	2ebf610b-98bf-4914-aeel-9b866d7b1897
metadata	{}
migration_status	None
multiattach	False
name	testVolumeEncrypted
os-vol-host-attr:host	ha-volume-manager@lvm-1#LVM_iSCSI
os-vol-mig-status-attr:migstat	None
os-vol-mig-status-attr:name_id	None
os-vol-tenant-attr:tenant_id	5f3b093c603f4dc8bc377d04e5385d42
os-volume-replication:driver_data	None
os-volume-replication:extended_status	None
replication_status	disabled
size	1
snapshot_id	None
source_volid	None
status	available
user_id	3bdde5491e174a8aafc5a88e01cac7
volume_type	LUKS
+-----+-----+	

When using an ESKM appliance as the back end, you can also confirm that key operations are going to your HSM via its admin portal.

UUID	Owner	Object Type	State
Creation Date			
8d54f41d-91dd-4f5e-bcfe-964af8213a8c	barbican	SymmetricKey	PreActive
2016-03-02 13:58:58			

## 12.3 For More Information

For more information on data at rest security with ESKM, see [Data Security Protection for SUSE OpenStack Cloud \(http://files.asset.microfocus.com/4aa6-5241/en/4aa6-5241.pdf\)](http://files.asset.microfocus.com/4aa6-5241/en/4aa6-5241.pdf).

## 13 Security Audit Logs

### 13.1 The need for auditing

Enterprises need the ability to audit and monitor workflows and data in accordance with their strict corporate, industry or governmental policies and compliance requirements such as FIPS-140-2, PCI-DSS, HIPAA, SOX, or ISO. To meet this need, SUSE OpenStack Cloud supports CADF (Cloud Auditing Data Federation)-compliant security audit logs that can easily be integrated with your organization's Security Information and Event Management (SIEM) tools. Such auditing is valuable not only to meet regulatory compliance requirements, but also for correlating threat forensics.

Note that logs from existing OpenStack services can also be used for auditing purposes, even though they are not in a consistent audit friendly CADF format today. All logs can easily be integrated with a SIEM tool such as HPE ArcSight, Splunk etc.

### 13.2 Audit middleware

Audit middleware is python middleware logic that addresses the aforementioned logging shortcomings. Audit middleware constructs audit event data in easily consumed CADF format. This data can be mined to answer critical questions about activities over REST resources such as who made the request, when, why, and so forth.

Audit middleware supports delivery of audit data via the Oslo messaging notifier feature. Each service is configured to route data to an audit-specific log file.

The following are key aspects of auditing support in SUSE OpenStack Cloud 8:

- Auditing is disabled by default and can be enabled only after SUSE OpenStack Cloud installation.
- Auditing support has been added to eight SUSE OpenStack Cloud services (Nova, Cinder, Glance, Keystone, Neutron, Heat, Barbican, and Ceilometer).
- Auditing has been added for interactions where REST API calls are invoked.
- All audit events are recorded in a service-specific audit log file.

- Auditing configuration is centrally managed and indicates for which services auditing is currently disabled or enabled.
- Auditing can be enabled or disabled on a per-service basis.

## 13.3 Centralized auditing configuration

In SUSE OpenStack Cloud, all auditing configuration is centrally managed and controlled via input model YAML files on the Cloud Lifecycle Manager node. The settings are configured in the file `~/openstack/my_cloud/definition/cloudConfig.yml` in a newly added audit-settings section shown below the following table.

Key	Value (default)	Type	Description	Expected value(s)	Comments
default	disabled	String	Flag to globally enable or disable auditing for all services.	<i>disabled, enabled</i>	A service's auditing behavior is determined via this default key value unless it is listed explicitly in the enabled-services or disabled-services list.
enabled-services	[] (empty list)	yaml list	Setting to explicitly enable auditing for listed services regardless of default flag setting.	<i>nova, cinder, glance, keystone, neutron, heat, barbican, ceilometer</i>	To enable a specific service, either add the service name in the enabled-services list when default is set to <u>disabled</u> or remove from disabled-services list when default is set to <u>enabled</u> .  If a service name is present in both enabled-services and disabled-services, then auditing will be enabled for that service.

Key	Value (default)	Type	Description	Expected value(s)	Comments
disabled-services	Nova, Barbican, Keystone, Cinder, Ceilometer, Neutron	yaml list	Setting to explicitly disable auditing for listed services regardless of default flag setting.	<i>nova, cinder, glance, keystone, neutron, heat, barbican, ceilometer</i>	To disable a specific service, either add the service name in disabled-services when default is set to <u>enabled</u> . or remove from enabled-services list when default is set to <u>disabled</u> .

Audit settings in `cloudConfig.yml` with default set to disabled and services selectively enabled:

```
product:
  version: 2
  cloud:
    ....
    ....
    # Disc space needs to be allocated to the audit directory before enabling
    # auditing.
    # keystone and nova has auditing enabled
    # cinder, ceilometer, glance, neutron, heat, barbican have auditing disabled
    audit-settings:
      audit-dir: /var/audit
      default: disabled
      enabled-services:
        - keystone
        - nova
      disabled-services:
        - cinder
        - ceilometer
```

Audit setting in `cloudConfig.yml` with default set to enabled and services selectively disabled:

```
product:
```

```
version: 2
cloud:
....
....
# Disc space needs to be allocated to the audit directory before enabling
# auditing.
# keystone, nova, glance, neutron, heat, barbican has auditing enabled
# cinder, ceilometer has auditing disabled
audit-settings:
audit-dir: /var/audit
default: enabled
enabled-services:
- keystone
- nova
disabled-services:
- cinder
- ceilometer
```

Because auditing is disabled by default, you will need to follow the steps below to enable it:

1. Book *“Operations Guide”*, Chapter 12 *“Managing Monitoring, Logging, and Usage Reporting”*, Section 12.2 *“Centralized Logging Service”*, Section 12.2.7 *“Audit Logging Overview”*, Section 12.2.7.1 *“Audit Logging Checklist”*
2. Book *“Operations Guide”*, Chapter 12 *“Managing Monitoring, Logging, and Usage Reporting”*, Section 12.2 *“Centralized Logging Service”*, Section 12.2.7 *“Audit Logging Overview”*, Section 12.2.7.2 *“Enable Audit Logging”*

For instructions on backing up and restoring audit logs, see: Book *“Operations Guide”*, Chapter 14 *“Backup and Restore”*, Section 14.13 *“Backing up and Restoring Audit Logs”* .