# Red Hat OpenStack Platform 17.0

## Security and Hardening Guide

Good Practices, Compliance, and Security Hardening

Last Updated: 2024-03-22

# Red Hat OpenStack Platform 17.0 Security and Hardening Guide

Good Practices, Compliance, and Security Hardening

OpenStack Team
rhos-docs@redhat.com

## Legal Notice

## Abstract

This guide provides good practice advice and conceptual information about hardening the security of a Red Hat OpenStack Platform environment.

# Table of Contents

# MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message .

# CHAPTER 1. INTRODUCTION TO SECURITY

Use the tools provided with Red Hat Openstack Platform (RHOSP) to prioritize security in planning, and in operations, to meet users' expectations of privacy and the security of their data. Failure to implement security standards can lead to downtime or data breaches. Your use case might be subject to laws that require passing audits and compliance processes.

- For information about hardening Ceph, see Data security and hardening guide.

NOTE

Follow the instructions in this guide to harden the security of your environment. However, these recommendations do not guarantee security or compliance. You must assess security from the unique requirements of your environment.

## 1.1. RED HAT OPENSTACK PLATFORM SECURITY

By default, Red Hat OpenStack Platform (RHOSP) director creates the overcloud with the following tools and access controls for security:

**SElinux**

SELinux provides security enhancement for RHOSP by providing access controls that require each process to have explicit permissions for every action.

**Podman**

Podman as a container tool is a secure option for RHOSP as it does not use a client/server model that requires processes with root access to function.

**System access restriction**

You can only log into overcloud nodes using either the SSH key that director creates for tripleo-admin during the overcloud deployment, or a SSH key that you have created on the overcloud. You cannot use SSH with a password to log into overcloud nodes, or log into overcloud nodes using root.

You can configure director with the following additional security features based on the needs and trust level of your organization:

- Public TLS and TLS-everywhere
- Hardware security module integration with OpenStack Key Manager (barbican)
- Signed images and encrypted volumes
- Password and fernet key rotation using workflow executions

## 1.2. UNDERSTANDING THE RED HAT OPENSTACK PLATFORM ADMIN ROLE

When you assign a user the role of **admin**, this user has permissions to view, change, create, or delete any resource on any project. This user can create shared resources that are accessible across projects, such as publicly available glance images, or provider networks. Additionally, a user with the **admin** role can create or delete users and manage roles.

The project to which you assign a user the **admin** role is the default project in which **openstack** commands are executed. For example, if an **admin** user in a project named **development** runs the following command, a network called **internal-network** is created in the **development** project:

```
openstack network create internal-network
```

The **admin** user can create an **internal-network** in any project by using the **--project** parameter:

```
openstack network create internal-network --project testing
```

## 1.3. IDENTIFYING SECURITY ZONES IN RED HAT OPENSTACK PLATFORM

Security zones are resources, applications, networks and servers that share common security concerns. Design security zones so to have common authentication and authorization requirements, and users. You can define your own security zones to be as granular as needed based on the architecture of your cloud, the level of acceptable trust in your environment, and your organization's standardized requirements. The zones and their trust requirements can vary depending upon whether the cloud instance is public, private, or hybrid.

For example, a you can segment a default installation of Red Hat OpenStack Platform into the following zones:

**Table 1.1. Security zones**

| Zone | Networks | Details |
|---|---|---|
| Public | external | The public zone hosts the external networks, public APIs, and floating IP addresses for the external connectivity of instances. This zone allows access from networks outside of your administrative control and is an untrusted area of the cloud infrastructure. |
| Guest | tenant | The guest zone hosts project networks. It is untrusted for public and private cloud providers that allow unrestricted access to instances. |
| Storage access | storage, storage_mgmt | The storage access zone is for storage management, monitoring and clustering, and storage traffic. |
| Control | ctlplane, internal_api, ipmi | The control zone also includes the undercloud, host operating system, server hardware, physical networking, and the Red Hat OpenStack Platform director control plane. |

## 1.4. LOCATING SECURITY ZONES IN RED HAT OPENSTACK PLATFORM

Run the following commands to collect information on the physical configuration of your Red Hat OpenStack Platform deployment:

### Prerequisites

- You have an installed Red Hat OpenStack Platform environment.
- You are logged into the director as stack.

### Procedure

1. Source **stackrc**:

```
$ source /home/stack/stackrc
```

2. Run **openstack subnet list** to match the assigned ip networks to their associated zones:

```
openstack subnet list -c Name -c Subnet
+---------------------+------------------+
| Name                | Subnet           |
+---------------------+------------------+
| ctlplane-subnet     | 192.168.101.0/24 |
| storage_mgmt_subnet | 172.16.105.0/24  |
| tenant_subnet       | 172.16.102.0/24  |
| external_subnet     | 10.94.81.0/24    |
| internal_api_subnet | 172.16.103.0/24  |
| storage_subnet      | 172.16.104.0/24  |
+---------------------+------------------+
```

3. Run **openstack server list** to list the physical servers in your infrastructure:

```
openstack server list -c Name -c Networks
+------------------------+-------------------------+
| Name                   | Networks                |
+------------------------+-------------------------+
| overcloud-controller-0  | ctlplane=192.168.101.15 |
| overcloud-controller-1  | ctlplane=192.168.101.19 |
| overcloud-controller-2  | ctlplane=192.168.101.14 |
| overcloud-novacompute-0 | ctlplane=192.168.101.18 |
| overcloud-novacompute-2 | ctlplane=192.168.101.17 |
| overcloud-novacompute-1 | ctlplane=192.168.101.11 |
+------------------------+-------------------------+
```

4. Use the **ctlplane** address from the **openstack server list** command to query the configuration of a physical node:

```
ssh tripleo-admin@192.168.101.15 ip addr
```

## 1.5. CONNECTING SECURITY ZONES

You must carefully configure any component that spans multiple security zones with varying trust levels or authentication requirements. These connections are often the weak points in network architecture. Ensure that you configure these connections to meet the security requirements of the highest trust level of any of the zones being connected. In many cases, the security controls of the connected zones are a primary concern due to the likelihood of attack. The points where zones meet present an additional potential point of attack and adds opportunities for attackers to migrate their attack to more sensitive parts of the deployment.

In some cases, OpenStack operators might want to consider securing the integration point at a higher standard than any of the zones in which it resides. Given the above example of an API endpoint, an adversary could potentially target the Public API endpoint from the public zone, leveraging this foothold in the hopes of compromising or gaining access to the internal or admin API within the management zone if these zones were not completely isolated.

The design of OpenStack is such that separation of security zones is difficult. Because core services will usually span at least two zones, special consideration must be given when applying security controls to them.

## 1.6. THREAT MITIGATION

Most types of cloud deployment, public, private, or hybrid, are exposed to some form of security threat. The following practices help mitigate security threats:

- Apply the principle of least privilege.
- Use encryption on internal and external interfaces.
- Use centralized identity management.
- Keep Red Hat OpenStack Platform updated.

Compute services can provide malicious actors with a tool for DDoS and brute force attacks. Methods of prevention include egress security groups, traffic inspection, intrusion detection systems, and customer education and awareness. For deployments accessible by public networks or with access to public networks, such as the Internet, ensure that processes and infrastructure are in place to detect and address outbound abuse.

**Additional resources**

- Implementing TLS-e with Ansible
- Integrating OpenStack Identity (keystone) with Red Hat Identity Manager (IdM)
- Keeping Red Hat OpenStack Platform Updated

# CHAPTER 2. SECURITY ENHANCEMENTS

The following sections provide some suggestions to harden the security of your overcloud.

## 2.1. USING SECURE ROOT USER ACCESS

The overcloud image automatically contains hardened security for the **root** user. For example, each deployed overcloud node automatically disables direct SSH access to the **root** user. You can still access the **root** user on overcloud nodes. Each overcloud node has a **tripleo-admin** user account. This user account contains the undercloud public SSH key, which provides SSH access without a password from the undercloud to the overcloud node.

### Prerequisites

- You have an installed Red Hat OpenStack Platform director environment.
- You are logged into the director as stack.

### Procedure

1. On the undercloud node, log in to the an overcloud node through SSH as the **tripleo-admin** user.
2. Switch to the **root** user with **sudo -i**.

## 2.2. ADDING SERVICES TO THE OVERCLOUD FIREWALL

When you deploy Red Hat OpenStack Platform, each core service is deployed with a default set of firewall rules on each overcloud node. You can use the **ExtraFirewallRules** parameter to create rules to open ports for additional services, or create rules to restrict services.

Each rule name becomes the comment for the respective **iptables** rule. Each rule name starts with a three-digit prefix to help Puppet order the rules in the final **iptables** file. The default Red Hat OpenStack Platform rules use prefixes in the 000 to 200 range. When you create rules for new services, prefix the name with a three-digit number higher than 200.

### Procedure

1. Use a string to define each rule name under the **ExtraFireWallRules** parameter. You can use the following parameters under the rule name to define the rule:
   - dport:: The destination port associated to the rule.
   - proto:: The protocol associated to the rule. Defaults to **tcp**.
   - action:: The action policy associated to the rule. Defaults to **accept**.
   - source:: The source IP address associated to the rule.
     The following example shows how to use rules to open additional ports for custom applications:

     ```
     cat > ~/templates/firewall.yaml <<EOF
     parameter_defaults:
       ExtraFirewallRules:
     ```

```
    '300 allow custom application 1':
      dport: 999
      proto: udp
    '301 allow custom application 2':
      dport: 8081
      proto: tcp
EOF
```

> **NOTE**
>
> When you do not set the **action** parameter, the result is **accept**. You can only set the **action** parameter to **drop**, **insert**, or **append**.

2. Include the **~/templates/firewall.yaml** file in the **openstack overcloud deloy** command. Include all templates that are necessary for your deployment:

```
openstack overcloud deploy --templates /
...
-e /home/stack/templates/firewall.yaml /
....
```

## 2.3. REMOVING SERVICES FROM THE OVERCLOUD FIREWALL

You can use rules to restrict services. The number that you use in the rule name determines where in **iptables** the rule will be inserted. The following procedure shows how to restrict the **rabbitmq** service to the InternalAPI network.

### Procedure

1. On a Controller node, find the number of the default **iptables** rule for **rabbitmq**:

```
[tripleo-admin@overcloud-controller-2 ~]$ sudo iptables -L | grep rabbitmq
ACCEPT     tcp  --  anywhere             anywhere             multiport dports vtr-
emulator,epmd,amqp,25672,25673:25683 state NEW /* 109 rabbitmq-bundle ipv4 */
```

2. In an environment file uder **parameter_defaults**, use the **ExtraFirewallRules** parameter to restrict **rabbitmq** to the InternalApi network. The rule is given a lower number thant the default **rabbitmq** rule number or 109:

```
cat > ~/templates/firewall.yaml <<EOF
parameter_defaults:
  ExtraFirewallRules:
    '098 allow rabbit from internalapi network':
      dport:
      - 4369
      - 5672
      - 25672
      proto: tcp
      source: 10.0.0.0/24
    '099 drop other rabbit access':
      dport:
      - 4369
      - 5672
      - 25672
      proto: tcp
      action: drop
EOF
```

NOTE

When you do not set the **action** parameter, the result is **accept**. You can only set the **action** parameter to **drop**, **insert**, or **append**.

3. Include the **~/templates/firewall.yaml** file in the **openstack overcloud deloy** command. Include all templates that are necessary for your deployment:

```
openstack overcloud deploy --templates /
...
-e /home/stack/templates/firewall.yaml /
....
```

## 2.4. CHANGING THE SIMPLE NETWORK MANAGEMENT PROTOCOL (SNMP) STRINGS

Director provides a default read-only SNMP configuration for your overcloud. It is advisable to change the SNMP strings to mitigate the risk of unauthorized users learning about your network devices.

NOTE

When you configure the **ExtraConfig** interface with a string parameter, you must use the following syntax to ensure that heat and Hiera do not interpret the string as a Boolean value: **'"<VALUE>"'**.

Set the following hieradata using the **ExtraConfig** hook in an environment file for your overcloud:

**SNMP traditional access control settings**

**snmp::ro_community**

IPv4 read-only SNMP community string. The default value is **public**.

**snmp::ro_community6**

IPv6 read-only SNMP community string. The default value is **public**.

**snmp::ro_network**

Network that is allowed to **RO query** the daemon. This value can be a string or an array. Default value is **127.0.0.1**.

**snmp::ro_network6**

Network that is allowed to **RO query** the daemon with IPv6. This value can be a string or an array. The default value is **::1/128**.

**tripleo::profile::base::snmp::snmpd_config**

Array of lines to add to the *snmpd.conf* file as a safety valve. The default value is **[]**. See the SNMP Configuration File web page for all available options.

For example:

–

```
parameter_defaults:
  ExtraConfig:
    snmp::ro_community: mysecurestring
    snmp::ro_community6: myv6securestring
```

This changes the read-only SNMP community string on all nodes.

#### SNMP view-based access control settings (VACM)

**snmp::com2sec**

An array of VACM com2sec mappings. Must provide SECNAME, SOURCE and COMMUNITY.

**snmp::com2sec6**

An array of VACM com2sec6 mappings. Must provide SECNAME, SOURCE and COMMUNITY.

For example:

```
parameter_defaults:
  ExtraConfig:
    snmp::com2sec: ["notConfigUser default mysecurestring"]
    snmp::com2sec6: ["notConfigUser default myv6securestring"]
```

This changes the read-only SNMP community string on all nodes.

For more information, see the **snmpd.conf** man page.

## 2.5. USING THE OPEN VSWITCH FIREWALL

You can configure security groups to use the Open vSwitch (OVS) firewall driver in Red Hat OpenStack Platform director. Use the **NeutronOVSFirewallDriver** parameter to specify firewall driver that you want to use:

- **iptables_hybrid** - Configures the Networking service (neutron) to use the iptables/hybrid based implementation.
- **openvswitch** - Configures the Networking service to use the OVS firewall flow-based driver.

The **openvswitch** firewall driver includes higher performance and reduces the number of interfaces and bridges used to connect guests to the project network.

#### IMPORTANT

Multicast traffic is handled differently by the Open vSwitch (OVS) firewall driver than by the iptables firewall driver. With iptables, by default, VRRP traffic is denied, and you must enable VRRP in the security group rules for any VRRP traffic to reach an endpoint. With OVS, all ports share the same OpenFlow context, and multicast traffic cannot be processed individually per port. Because security groups do not apply to all ports (for example, the ports on a router), OVS uses the **NORMAL** action and forwards multicast traffic to all ports as specified by RFC 4541.

#### NOTE

The **iptables_hybrid** option is not compatible with OVS-DPDK. The **openvswitch** option is not compatible with OVS Hardware Offload.

Configure the **NeutronOVSFirewallDriver** parameter in the **network-environment.yaml** file:

```
NeutronOVSFirewallDriver: openvswitch
```

- **NeutronOVSFirewallDriver** : Configures the name of the firewall driver that you want to use when you implement security groups. Possible values depend on your system configuration. Some examples are **noop**, **openvswitch**, and **iptables_hybrid**. The default value of an empty string results in a supported configuration.

# CHAPTER 3. DOCUMENTING YOUR RHOSP ENVIRONMENT

Documenting the system components, networks, services, and software is important in identifying security concerns, attack vectors, and possible security zone bridging points. The documentation for your Red Hat OpenStack Platform (RHOSP) deployment should include the following information:

- A description of the system components, networks, services, and software in your RHOSP production, development, and test environments.
- An inventory of any ephemeral resources, such as virtual machines or virtual disk volumes.

## 3.1. DOCUMENTING THE SYSTEM ROLES

Each node in your Red Hat OpenStack Platform (RHOSP) deployment serves a specific role, either contributing to the infrastructure of the cloud, or providing cloud resources.

Nodes that contribute to the infrastructure run the cloud-related services, such as the message queuing service, storage management, monitoring, networking, and other services required to support the operation and provisioning of the cloud. Examples of infrastructure roles include the following:

- Controller
- Networker
- Database
- Telemetry

Nodes that provide cloud resources offer compute or storage capacity for instances running on your cloud. Examples of resource roles include the following:

- CephStorage
- Compute
- ComputeOvsDpdk
- ObjectStorage

Document the system roles that are used in your environment. These roles can be identified within the templates used to deploy RHOSP. For example, there is a NIC configuration file for each role in use in your environment.

### Procedure

1. Check the existing templates for your deployment for files that specify the roles currently in use. There is a NIC configuration file for each role in use in your environment. In the following example, the RHOSP environment includes the **ComputeHCI** role, the **Compute** role, and the **Controller** role:

```
$ cd ~/templates
$ tree
.
├── environments
│   └── network-environment.yaml
├── hci.yaml
├── network
│   └── config
│       └── multiple-nics
│           ├── computehci.yaml
│           ├── compute.yaml
│           └── controller.yaml
├── network_data.yaml
├── plan-environment.yaml
└── roles_data_hci.yaml
```

2. Each role for your RHOSP environment performs many interrelated services. You can document the services used by each role by inspecting a **roles** file.

   a. If a **roles** file was generated for your templates, you can find it in the **~/templates** directory:

   ```
   $ cd ~/templates
   $ find . -name *role*
   > ./templates/roles_data_hci.yaml
   ```

   b. If a **roles** file was not generated for your templtes, you can generate one for the roles you currently use to inspect for documentation purposes:

   ```
   $ openstack overcloud roles generate \
   > --roles-path /usr/share/openstack-tripleo-heat-templates/roles \
   > -o roles_data.yaml Controller Compute
   ```

## 3.2. CREATING A HARDWARE INVENTORY

You can retrieve hardware information aobut your Red Hat OpenStack Platform deployment by viewing data that is collected during introspection. Introspection gathers hardware information from the nodes about the CPU, memory, disks, and so on.

### Prerequisites

- You have an installed Red Hat OpenStack Platform director environment.
- You have introspected nodes for your Red Hat OpenStack Platform deployment.
- You are logged into the director as stack.

### Procedure

1. From the undercloud, source the **stackrc** file:

   ```
   $ source ~/stackrc
   ```

2. List the nodes in your environment:

   ```
   $ openstack baremetal node list -c Name
   +--------------+
   | Name         |
   +--------------+
   | controller-0 |
   | controller-1 |
   ```

```
| controller-2 |
| compute-0    |
| compute-1    |
| compute-2    |
+--------------+
```

3. For each baremetal node from which to gather information, and run the following command to retrieve the introspection data:

   ```
   $ openstack baremetal introspection data save <node> | jq
   ```

   Replace **<node>** with the name of the node from the list you retrieved in step 1.

4. Optional: To limit the output to a specific type of hardware, you can retrieve a list of the inventory keys and view introspection data for a specific key:

   a. Run the following command to get a list of top level keys from introspection data:

   ```
   $ openstack baremetal introspection data save controller-0 | jq '.inventory | keys'

   [
     "bmc_address",
     "bmc_v6address",
     "boot",
     "cpu",
     "disks",
     "hostname",
     "interfaces",
     "memory",
     "system_vendor"
   ]
   ```

   b. Select a key, for example **disks**, and run the following to get more information:

   ```
   $ openstack baremetal introspection data save controller-1 | jq '.inventory.disks'
   [
     {
       "name": "/dev/sda",
       "model": "QEMU HARDDISK",
       "size": 85899345920,
       "rotational": true,
       "wwn": null,
       "serial": "QM00001",
       "vendor": "ATA",
       "wwn_with_extension": null,
       "wwn_vendor_extension": null,
       "hctl": "0:0:0:0",
       "by_path": "/dev/disk/by-path/pci-0000:00:01.1-ata-1"
     }
   ]
   ```

## 3.3. CREATING A SOFTWARE INVENTORY

Document the software components in use on nodes deployed in your Red Hat OpenStack Platform

(RHOSP) infrastructure. System databases, RHOSP software services and supporting components such as load balancers, DNS, or DHCP services, are critical when assessing the impact of a compromise or vulnerability in a library, application, or class of software.

- You have an installed Red Hat OpenStack Platform environment.
- You are logged into the director as stack.

**Procedure**

1. Ensure that you know the entry points for systems and services that can be subject to malicious activity. Run the following commands on the undercloud:

```
$ cat /etc/hosts
$ source stackrc ; openstack endpoint list
$ source overcloudrc ; openstack endpoint list
```

2. RHOSP is deployed in containerized services, therefore you can view the software components on an overcloud node by checking the running containers on that node. Use **ssh** to connect to an overcloud node and list the running containers. For example, to view the overcloud services on **compute-0**, run a command similar to the following:

```
$ ssh tripleo-admin@compute-0 podman ps
```

# CHAPTER 4. IDENTITY AND ACCESS MANAGEMENT

The Identity service (keystone) provides authentication and authorization for cloud users in a Red Hat OpenStack Platform environment. You can use the Identity service for direct end-user authentication, or configure it to use external authentication methods to meet your security requirements or to match your current authentication infrastructure.

## 4.1. RED HAT OPENSTACK PLATFORM FERNET TOKENS

Fernet is the default token provider that replaces the **UUID** token provider. Each fernet token remains valid for up to an hour, by default. This allows a user to perform a series of tasks without needing to reauthenticate.

After you authenticate, the Identity service (keystone):

- Issues an encrypted bearer token known as a fernet token. This token represents your identity.
- Authorizes you you to perform operations based on your role.

**Additional resources**

- Using Fernet keys for encryption in the overcloud

## 4.2. OPENSTACK IDENTITY SERVICE ENTITIES

The Red Hat OpenStack Identity service (keystone) recognizes the following entities:

**Users**

OpenStack Identity service (keystone) users are the atomic unit of authentication. A user must be assigned a role on a project in order to authenticate.

**Groups**

OpenStack Identity service groups are a logical grouping of users. A group can be provided access to projects under specific roles. Managing groups instead of users can simplify the management of roles.

**Roles**

OpenStack Identity service roles define the OpenStack APIs that are accessible to users or groups who are assigned those roles.

**Projects**

OpenStack Identity service projects are isolated groups of users who have common access to a shared quota of physical resources and the virtual infrastructure built from those physical resources.

**Domains**

OpenStack Identity service domains are high-level security boundaries for projects, users, and groups. You can use OpenStack Identity domains to centrally manage all keystone-based identity components. Red Hat OpenStack Platform supports multiple domains. You can represent users of different domains by using separate authentication backends.

## 4.3. AUTHENTICATING WITH KEYSTONE

You can adjust the authentication security requirements required by OpenStack Identity service (keystone).

When you deploy Red Hat OpenStack Platform (RHOSP), it is possible to specify password requirements that are more complex than the default passwords that are generated for services. When this occurs, services cannot authenticate, and the deployment fails.

You must initially deploy RHOSP without password complexity requirements. After the deployment completes, add the **KeystonePasswordRegex** parameter to your templates, and re-run the deployment.

To harden your environment, implement password complexity requirements that meet the standards of your organization. For information about NIST recommended password complexity requirements, see publication 88-63B, Appendix A.

**Table 4.1. Identity service authentication parameters**

| Parameter | Description |
|---|---|
| **KeystoneChangePasswordUponFirstUse** | Enabling this option requires users to change their password when the user is created, or upon administrative reset. |
| **KeystoneDisableUserAccountDaysInactive** | The maximum number of days a user can go without authenticating before being considered "inactive" and automatically disabled (locked). |
| **KeystoneLockoutDuration** | The number of seconds a user account is locked when the maximum number of failed authentication attempts (as specified by **KeystoneLockoutFailureAttempts**) is exceeded. |
| **KeystoneLockoutFailureAttempts** | The maximum number of times that a user can fail to authenticate before the user account is locked for the number of seconds specified by **KeystoneLockoutDuration**. |
| **KeystoneMinimumPasswordAge** | The number of days that a password must be used before the user can change it. This prevents users from changing their passwords immediately in order to wipe out their password history and reuse an old password. |
| **KeystonePasswordExpiresDays** | The number of days for which a password is considered valid before requiring users to change it. |
| **KeystonePasswordRegex** | The regular expression that is used to validate password strength requirements. |
| **KeystonePasswordRegexDescription** | Describe your password regular expression here in language for humans. |
| **KeystoneUniqueLastPasswordCount** | This controls the number of previous user password iterations to keep in history, in order to enforce that newly created passwords are unique. |

**Additional resources**

- Identity (keystone) parameters.

### 4.3.1. Using Identity service heat parameters to stop invalid login attempts

Repetitive failed login attempts can be a sign of an attempted brute-force attack. You can use the Identity Service to limit access to accounts after repeated unsuccessful login attempts.

**Prerequisites**

- You have an installed Red Hat OpenStack Platform director environment.
- You are logged into the director as stack.

**Procedure**

1. To configure the maximum number of times that a user can fail to authenticate before the user account is locked, set the value of the **KeystoneLockoutFailureAttempts** and **KeystoneLockoutDuration** heat parameters in an environment file. In the following example, the **KeystoneLockoutDuration** is set to one hour:

```
parameter_defaults
   KeystoneLockoutDuration: 3600
   KeystoneLockoutFailureAttempts: 3
```

2. Include the environment file in your deploy script. When you run your deploy script on a previously deployed environment, it is updated with the additional parameters:

```
openstack overcloud deploy --templates \
...
-e keystone_config.yaml
...
```

## 4.4. AUTHENTICATING WITH EXTERNAL IDENTITY PROVIDERS

You can use an external identity provider (IdP) to authenticate to OpenStack service providers (SP). SPs are the services provided by an OpenStack cloud.

When you use a separate IdP, external authentication credentials are separate from the databases used by other OpenStack services. This separation reduces the risk of a compromise of stored credentials.

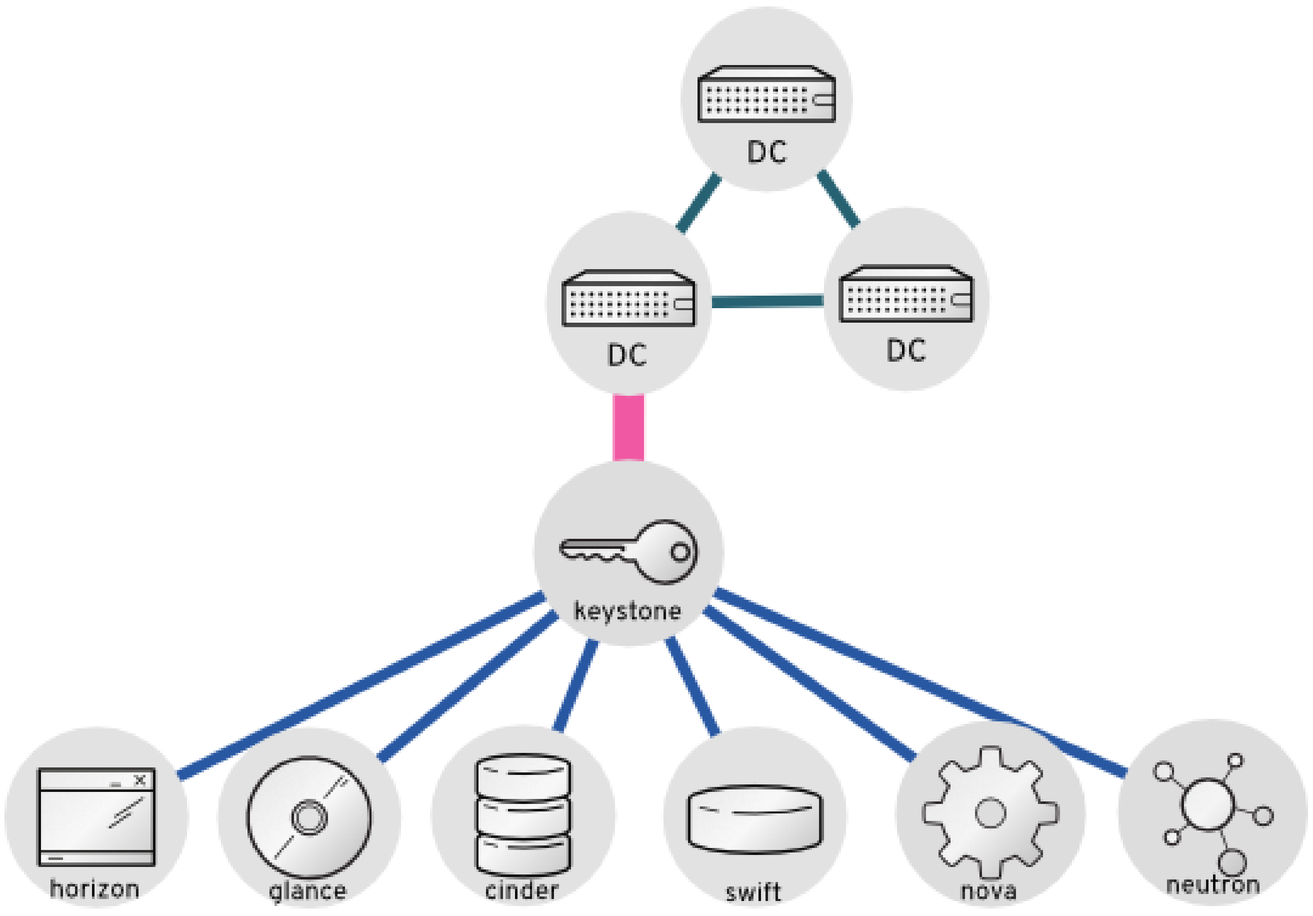
Each external IdP has a one-to-one mapping to an OpenStack Identity service (keystone) domain. You can have multiple coexisting domains with Red Hat OpenStack Platform.

External authentication provides a way to use existing credentials to access resources in Red Hat OpenStack Platform without creating additional identities. The credential is maintained by the user's IdP.

You can use IdPs such as Red Hat Identity Management (IdM), and Microsoft Active Directory Domain Services (AD DS) for identity management. In this configuration, the OpenStack Identity service has read-only access to the LDAP user database. The management of API access based on user or group role is performed by keystone. Roles are assigned to the LDAP accounts by using the OpenStack Identity service.

### 4.4.1. How LDAP integration works

In the diagram below, keystone uses an encrypted LDAPS connection to connect to an Active Directory Domain Controller. When a user logs in to horizon, keystone receives the supplied user credentials and passes them to Active Directory.



#### Additional resources

- Integrating OpenStack Identity (keystone) with Active Directory
- Integrating OpenStack Identity (keystone) with Red Hat Identity Manager (IdM)
- Configuring director to use domain specific LDAP backends

# CHAPTER 5. SECURING RED HAT OPENSTACK DEPLOYMENTS WITH TLS AND PKI

Red Hat OpenStack Platform consists of many networks and endpoints that handle sensitive or confidential data that you can secure. When you use Transport Layer Security (TLS), you secure traffic with symmetric key encryption. The key and cipher are negotiated in the TLS handshake, which requires validation of the server's identity through a shared trust in an intermediary called a Certificate Authority (CA).

Public Key Infrastructure (PKI) is a framework for validating an entity through a certificate authority.

## 5.1. COMPONENTS OF PUBLIC KEY INFRASTRUCTURE (PKI)

The core components of PKI are shown in the folling table:

**Table 5.1. Key Terms**

| Term | Definition |
|---|---|
| End entity | The user, process, or system that validates itself through the use of a digital certificate. |
| Certificate Authority (CA) | The CA is an entity that is trusted by both the end entity, and the relying party that validates the end entity. |
| Relying party | The relying party receives the digital certificate as validation of the end entity, and has the capability of verifying the digital certificate. |
| Digital certificates | Signed public key certificates have a verifiable entity and a public key, and are issued by a CA. When a CA signs a certificate, it creates a message digest from the certificate encrypted with its private key. You can verify the signature using the public key associated with CA. The X.509 standard is used to define the certificates. |
| Registration Authority (RA) | An RA is an optional dedicated authority that can perform management functions such as authenticating end entities before they are issued a certificate by a CA. The CA authenticates end entities if there is no RA. |
| Certificate Revocation List (CRL) | A CRL is a list of certificate serial numbers that have been revoked. End entities presenting certificates with revoked serial numbers are not trusted in a PKI model. |
| CRL issuer | An optional system to which a CA delegates the publication of certificate revocation lists. |

| Term | Definition |
|---|---|
| Certificate Repository | The location where the end entity certificates and certificate revocation lists are stored and queried. |

## 5.2. CERTIFICATE AUTHORITY REQUIREMENTS AND RECOMMENDATIONS

You must get certificates signed by a widely recognized certificate authority (CA) for publicly available Red Hat OpenStack Platform Dashboards or publicly accessible APIs.

You must give a DNS domain or subdomain to each endpoint that you secure with TLS. The domains you provide are used to create the certificates issued by a CA. Customers access the dashboard or the API using the DNS name so that the CA can validate the endpoint.

Red Hat recommends using a separate and internally managed CA to secure internal traffic. This allows the cloud deployer to maintain control of their Private Key Infrastructure (PKI) implementation and makes requesting, signing and deploying certificates for internal systems easier.

You can enable SSL/TLS on your overcloud endpoints. Due to the number of certificates required to configure TLS everywhere (TLS-e), director integrates with a Red Hat Identity Management (IdM) server to act as a certificate authority and manage the overcloud certificates. For more information on configuring TLS-e, see Implementing TLS-e with Ansible.

To check the status of TLS support across the OpenStack components, refer to the TLS Enablement status matrix.

If want to you use a SSL certificate with your own certificate authority, see Enabling SSL/TLS on overcloud public endpoints.

**NOTE**

This will configure Red Hat OpenStack Platform with SSL/TLS on publicly accessible endpoints only.

## 5.3. IDENTIFYING TLS VERSIONS IN YOUR ENVIRONMENT

**IMPORTANT**

TLS version 1.0 is is deprecated for Red Hat OpenStack platform. Additionally, you must at minimum use TLS 1.2 for NIST-approval. For more information, see Guidelines for the Selection, Configuration, and Use of Transport Layer Security (TLS) Implementations.

You can use **cipherscan** to determine the versions of TLS being presented by your deployment. Cipherscan can be cloned from https://github.com/mozilla/cipherscan. This example output demonstrates results received from **horizon**:

**NOTE**

Run **cipherscan** from a non-production system, as it might install additional dependencies when you first run it.

**Procedure**

- Run cipher scan against the accessible **URL** of the Dashboard service:

```
$ ./cipherscan https://openstack.lab.local
..............................
Target: openstack.lab.local:443

prio  ciphersuite                  protocols  pfs                curves
1     ECDHE-RSA-AES128-GCM-SHA256  TLSv1.2    ECDH,P-256,256bits  prime256v1
2     ECDHE-RSA-AES256-GCM-SHA384  TLSv1.2    ECDH,P-256,256bits  prime256v1
3     DHE-RSA-AES128-GCM-SHA256    TLSv1.2    DH,1024bits         None
4     DHE-RSA-AES256-GCM-SHA384    TLSv1.2    DH,1024bits         None
5     ECDHE-RSA-AES128-SHA256      TLSv1.2    ECDH,P-256,256bits  prime256v1
6     ECDHE-RSA-AES256-SHA384      TLSv1.2    ECDH,P-256,256bits  prime256v1
7     ECDHE-RSA-AES128-SHA         TLSv1.2    ECDH,P-256,256bits  prime256v1
8     ECDHE-RSA-AES256-SHA         TLSv1.2    ECDH,P-256,256bits  prime256v1
9     DHE-RSA-AES128-SHA256        TLSv1.2    DH,1024bits         None
10    DHE-RSA-AES128-SHA           TLSv1.2    DH,1024bits         None
11    DHE-RSA-AES256-SHA256        TLSv1.2    DH,1024bits         None
12    DHE-RSA-AES256-SHA           TLSv1.2    DH,1024bits         None
13    ECDHE-RSA-DES-CBC3-SHA       TLSv1.2    ECDH,P-256,256bits  prime256v1
14    EDH-RSA-DES-CBC3-SHA         TLSv1.2    DH,1024bits         None
15    AES128-GCM-SHA256            TLSv1.2    None                None
16    AES256-GCM-SHA384            TLSv1.2    None                None
17    AES128-SHA256                TLSv1.2    None                None
18    AES256-SHA256                TLSv1.2    None                None
19    AES128-SHA                   TLSv1.2    None                None
20    AES256-SHA                   TLSv1.2    None                None
21    DES-CBC3-SHA                 TLSv1.2    None                None

Certificate: trusted, 2048 bits, sha256WithRSAEncryption signature
TLS ticket lifetime hint: None
NPN protocols: None
OCSP stapling: not supported
Cipher ordering: server
Curves ordering: server - fallback: no
Server supports secure renegotiation
Server supported compression methods: NONE
TLS Tolerance: yes

Intolerance to:
 SSL 3.254           : absent
 TLS 1.0             : PRESENT
 TLS 1.1             : PRESENT
 TLS 1.2             : absent
 TLS 1.3             : absent
 TLS 1.4             : absent
```

When scanning a server, Cipherscan advertises support for a specific TLS version, which is the highest TLS version it is willing to negotiate. If the target server correctly follows TLS protocol, it will respond with the highest version that is mutually supported, which may be lower than what Cipherscan initially advertised. If the server does proceed to establish a connection with the client using that specific version, it is not considered to be intolerant to that protocol version. If it does not establish the connection (with the specified version, or any lower version), then intolerance for that version of protocol is considered to be present. For example:

```
Intolerance to:
 SSL 3.254         : absent
 TLS 1.0          : PRESENT
 TLS 1.1          : PRESENT
 TLS 1.2          : absent
 TLS 1.3          : absent
 TLS 1.4          : absent
```

In this output, intolerance of **TLS 1.0** and **TLS 1.1** is reported as **PRESENT**, meaning that the connection could not be established, and that Cipherscan was unable to connect while advertising support for those TLS versions. As a result, it is reasonable to conclude that those (and any lower) versions of the protocol are not enabled on the scanned server.

## 5.4. IDENTITY MANAGEMENT (IDM) SERVER RECOMMENDATIONS FOR OPENSTACK

Red Hat provides the following information to help you integrate your IdM server and OpenStack environment.

For information on preparing Red Hat Enterprise Linux for an IdM installation, see Installing Identity Management.

Run the **ipa-server-install** command to install and configure IdM. You can use command parameters to skip interactive prompts. Use the following recommendations so that your IdM server can integrate with your Red Hat OpenStack Platform environment:

**Table 5.2. Parameter recommendations**

| Option | Recommendation |
|---|---|
| **--admin-password** | Note the value you provide. You will need this password when configuring Red Hat OpenStack Platform to work with IdM. |
| **--ip-address** | Note the value you provide. The undercloud and overcloud nodes require network access to this ip address. |
| **--setup-dns** | Use this option to install an integrated DNS service on the IdM server. The undercloud and overcloud nodes use the IdM server for domain name resolution. |
| **--auto-forwarders** | Use this option to use the addresses in **/etc/resolv.conf** as DNS forwarders. |
| **--auto-reverse** | Use this option to resolve reverse records and zones for the IdM server IP addresses. If neither reverse records or zones are resolvable, IdM creates the reverse zones. This simplifies the IdM deployment. |

| Option | Recommendation |
|---|---|
| **--ntp-server**, **--ntp-pool** | You can use both or either of these options to configure your NTP source. Both the IdM server and your OpenStack environment must have correct and synchronized time. |

You must open the firewall ports required by IdM to enable communication with Red Hat OpenStack Platform nodes. For more information, see Opening the ports required by IdM .

### Additional resources

- Configuring and Managing Identity Management
- Red Hat Identity Management Documentation

## 5.5. IMPLEMENTING TLS-E WITH ANSIBLE

You can use the new **tripleo-ipa** method to enable SSL/TLS on overcloud endpoints, called TLS everywhere (TLS-e). Due to the number of certificates required, Red Hat OpenStack Platform integrates with Red Hat Identity management (IdM). When you use **tripleo-ipa** to configure TLS-e, IdM is the certificate authority.

### Prerequisites

- Ensure that all configuration steps for the undercloud, such as the creation of the stack user, are complete. For more details, see Director Installation and Usage for more details
- The IP address of your DNS server is configured on the undercloud to the IP address of the IdM server. One of the following parameters must be configured in your **undercloud.conf** file:
  - **DEFAULT/undercloud_nameservers**
  - **%SUBNET_SECTION%/dns_nameservers**

### Procedure

Use the following procedure to implement TLS-e on a new installation of Red Hat OpenStack Platform, or an existing deployment that you want to configure with TLS-e. You must use this method if you deploy Red Hat OpenStack Platform with TLS-e on pre-provisioned nodes.

**NOTE**

If you are implementing TLS-e for an existing environment, you are required to run commands such as **openstack undercloud install**, and **openstack overcloud deploy**. These procedures are idempotent and only adjust your existing deployment configuration to match updated templates and configuration files.

1. Configure the **/etc/resolv.conf** file:
   Set the appropriate search domains and the nameserver on the undercloud in **/etc/resolv.conf**. For example, if the deployment domain is **example.com**, and the domain of the FreeIPA server is **bigcorp.com**, then add the following lines to /etc/resolv.conf:

   -

```
search example.com bigcorp.com
nameserver $IDM_SERVER_IP_ADDR
```

2. Install required software:

```
sudo dnf install -y python3-ipalib python3-ipaclient krb5-devel
```

3. Export environmental variables with values specific to your environment.:

```
export IPA_DOMAIN=bigcorp.com
export IPA_REALM=BIGCORP.COM
export IPA_ADMIN_USER=$IPA_USER 1
export IPA_ADMIN_PASSWORD=$IPA_PASSWORD 2
export IPA_SERVER_HOSTNAME=ipa.bigcorp.com
export UNDERCLOUD_FQDN=undercloud.example.com 3
export USER=stack
export CLOUD_DOMAIN=example.com
```

1 2 The IdM user credentials are an administrative user that can add new hosts and services.

3 The value of the **UNDERCLOUD_FQDN** parameter matches the first hostname-to-IP address mapping in **/etc/hosts**.

4. Run the **undercloud-ipa-install.yaml** ansible playbook on the undercloud:

```
ansible-playbook \
--ssh-extra-args "-o StrictHostKeyChecking=no -o UserKnownHostsFile=/dev/null" \
/usr/share/ansible/tripleo-playbooks/undercloud-ipa-install.yaml
```

5. Add the following parameters to undercloud.conf

```
undercloud_nameservers = $IDM_SERVER_IP_ADDR
overcloud_domain_name = example.com
```

6. [Optional] If your IPA realm does not match your IPA domain, set the value of the **certmonger_krb_realm** parameter:

   a. Set the value of the **certmonger_krb_realm** in **/home/stack/hiera_override.yaml**:

   ```
   parameter_defaults:
     certmonger_krb_realm = EXAMPLE.COMPANY.COM
   ```

   b. Set the value of the **custom_env_files** parameter in **undercloud.conf** to **/home/stack/hiera_override.yaml**:

   ```
   custom_env_files = /home/stack/hiera_override.yaml
   ```

7. Deploy the undercloud:

```
openstack undercloud install
```

### Verification

Verify that the undercloud was enrolled correctly by completing the following steps:

1. List the hosts in IdM:

```
$ kinit admin
$ ipa host-find
```

2. Confirm that **/etc/novajoin/krb5.keytab** exists on the undercloud.

```
ls /etc/novajoin/krb5.keytab
```

NOTE

The **novajoin** directory name is for legacy naming purposes only.

## Configuring TLS-e on the overcloud

When you deploy the overcloud with TLS everywhere (TLS-e), IP addresses from the Undercloud and Overcloud will automatically be registered with IdM.

1. Before deploying the overcloud, create a YAML file **tls-parameters.yaml** with contents similar to the following. The values you select will be specific for your environment:

```
parameter_defaults:
    DnsSearchDomains: ["example.com"]
    CloudDomain: example.com
    CloudName: overcloud.example.com
    CloudNameInternal: overcloud.internalapi.example.com
    CloudNameStorage: overcloud.storage.example.com
    CloudNameStorageManagement: overcloud.storagemgmt.example.com
    CloudNameCtlplane: overcloud.ctlplane.example.com
    IdMServer: freeipa-0.redhat.local
    IdMDomain: redhat.local
    IdMInstallClientPackages: False

resource_registry:
      OS::TripleO::Services::IpaClient: /usr/share/openstack-tripleo-heat-
templates/deployment/ipa/ipaservices-baremetal-ansible.yaml
```

- The shown value of the **OS::TripleO::Services::IpaClient** parameter overrides the default setting in the **enable-internal-tls.yaml** file. You must ensure the **tls-parameters.yaml** file follows **enable-internal-tls.yaml** in the **openstack overcloud deploy** command.

- For more information about the parameters that you use to implement TLS-e, see Parameters for tripleo-ipa.

2. Deploy the overcloud. You will need to include the tls-parameters.yaml in the deployment command:

```
DEFAULT_TEMPLATES=/usr/share/openstack-tripleo-heat-templates/
CUSTOM_TEMPLATES=/home/stack/templates

openstack overcloud deploy \
-e ${DEFAULT_TEMPLATES}/environments/ssl/tls-everywhere-endpoints-dns.yaml \
-e ${DEFAULT_TEMPLATES}/environments/services/haproxy-public-tls-certmonger.yaml \
-e ${DEFAULT_TEMPLATES}/environments/ssl/enable-internal-tls.yaml \
-e ${CUSTOM_TEMPLATES}/tls-parameters.yaml \
...
```

3. Confirm each endpoint is using HTTPS by querying keystone for a list of endpoints:

```
openstack endpoint list
```

## 5.6. PARAMETERS FOR TRIPLEO-IPA

Use the fully qualified domain name (FQDN) of your cloud to define the cloud name and cloud domain parameters required for **tripleo-ipa**. For example, with an FQDN of **overcloud.example.com**, use the following values:

- CloudDomain: example.com
- CloudName: overcloud.example.com
- CloudNameCtlplane: overcloud.ctlplane.example.com
- CloudNameInternal: overcloud.internalapi.example.com
- CloudNameStorage: overcloud.storage.example.com
- CloudNameStorageManagement: overcloud.storagemgmt.example.com

Set the following additional parameters based on the requirements of your environment:

CertmongerKerberosRealm

Set **CertmongerKerberosRealm** parameter to the value of the IPA realm. This is required if the IPA realm does not match the IPA domain.

DnsSearchDomains

The **DnsSearchDomains** parameter is a comma-separated list. If the domain of the IdM server is different than the cloud domain, include the domain of the IdM server in the **DnsSearchDomains** parameter.

EnableEtcdInternalTLS

If you deploy TLSe on a distributed compute node (DCN) architecture, you must add the **EnableEtcdInternalTLS** parameter with the value of **True**.

IDMInstallClientPackages

If you have preprovisioned your compute nodes, set the **IDMInstallClientPackages** parameter to a value of **True**. Otherwise, set the value to **False**.

IDMModifyDNS

Set the **IDMModifyDNS** parameter to **false** to disable automatic IP registration of the overcloud nodes on Red Hat Identity Server.

IdmDomain

Set the **IdmDomain** parameter to the domain portion of the FQDN of your Red Hat Identity server. The value that you specify is also used as the value of the IdM realm. If the IdM domain and IdM realm differ, set the realm explicitly using the **CertmongerKerberosRealm** parameter.

IdmServer

Set the **IdmServer** parameter to the FQDN of your Red Hat Identity server. If you use a replicated IdM environment, then set multiple values using a comma delimited list. For more information on IdM replicas, see Installing an IdM replica .

## 5.7. ENCRYPTING MEMCACHED TRAFFIC UNDER TLS EVERYWHERE (TLS-E)

IMPORTANT

This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

You can now encrypt memcached traffic with TLS-e. This feature works with both novajoin and tripleo-ipa:

1. Create an environment file called **memcached.yaml** with the following contents to add TLS support for memcached:

```
parameter_defaults:
  MemcachedTLS: true
  MemcachedPort: 11212
```

2. Include the **memcached.yaml** environment file in the overcloud deployment process:

```
openstack overcloud deploy --templates \
-e /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-internal-tls.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-everywhere-endpoints-
dns.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/haproxy-public-tls-
certmonger.yaml \
-e /home/stack/memcached.yaml
...
```

### Additional Resources

- For more information about deploying TLSe with tripleo-ipa, see Implementing TLS-e with Ansible.

## 5.8. INCREASING THE SIZE OF PRIVATE KEYS

You can improve security by increasing the the size of private keys used to create certificates for encrypted service traffic. The default RHOSP private key size of 2048 bits matches the National Institute of Standards and Technology (NIST) recommended minimum.

- Use the **CertificateKeySize** parameter to change the size of private keys globally.
- Use a service specific parameter such as **RedisCertificateKeySize**, to modify a specific private key, or to override the global **CertificateKeySize** parameter.

Use these parameters in an environment heat template, and include the template in your overcloud deployment command. If you have already deployed the overcloud, you must rerun the same **openstack**

**overcloud deploy** command with the same templates you originally used, and include the new parameters for the changes to take effect.

In the following example, the global value for private keys is **4096**. The private key for **redis** is **2048** because the **RedisCertificateKeySize** overrides the global parameter:

### Example

```
parameter_defaults:
  CertificateKeySize: '4096'
  RedisCertificateKeySize: '2048'
```

## 5.9. REPLACING THE IDM SERVER FOR RED HAT OPENSTACK PLATFORM WITH ITS REPLICA

When you replace an existing IPA server with its replica, you must update the necessary parameters. Failure to do so results in failed overcloud deployments when you update your cluster's configuration.

### Procedure

1. On each overcloud node, edit the **/etc/ipa/default.conf** configuration file to ensure that the **server** and **xmlrpc_uri** parameters include the fully qualified domain name (FQDN) of the IdM server:

   ```
   #File modified by ipa-client-install

   [global]
   basedn = dc=redhat,dc=local
   realm = REDHAT.LOCAL
   domain = redhat.local
   server = freeipa-0.redhat.local
   host = undercloud-0.redhat.local
   xmlrpc_uri = https://freeipa-0.redhat.local/ipa/xml
   enable_ra = True
   ```

2. On the undercloud, edit the **/home/stack/templates/tls-parameters.yaml** environment file and ensure that the **IPA_SERVER_HOSTNAME** parameter matches the FQDN that is shown in the **xmlrpc_uri** and server parameters in **/etc/ipa/default.conf**. Ensure that all parameters match your environment:

   ```
   export IPA_DOMAIN=bigcorp.com
   export IPA_REALM=BIGCORP.COM
   export IPA_ADMIN_USER=$IPA_USER
   export IPA_ADMIN_PASSWORD=$IPA_PASSWORD
   export IPA_SERVER_HOSTNAME=ipa.bigcorp.com
   export UNDERCLOUD_FQDN=undercloud.example.com
   export USER=stack
   export CLOUD_DOMAIN=example.com
   ```

# CHAPTER 6. CONFIGURING CUSTOM SSL/TLS CERTIFICATES

You can manually configure the undercloud to use SSL/TLS for communication over public endpoints. When you manually configure undercloud endpoints with SSL/TLS, you are creating secure endpoints as a proof-of-concept. Red Hat recommends using a certificate authority solution.

When you use a certificate authority (CA) solution, you have production ready solutions such as a certificate renewals, certificate revocation lists (CRLs), and industry accepted cryptography. For information on using Red Hat Identity Manager (IdM) as a CA, see Implementing TLS-e with Ansible.

If you want to use a SSL certificate with your own certificate authority, you must complete the following configuration steps.

## 6.1. INITIALIZING THE SIGNING HOST

The signing host is the host that generates and signs new certificates with a certificate authority. If you have never created SSL certificates on the chosen signing host, you might need to initialize the host so that it can sign new certificates.

### Procedure

1. The **/etc/pki/CA/index.txt** file contains records of all signed certificates. Ensure that the filesystem path and **index.txt** file are present:

```
$ sudo mkdir -p /etc/pki/CA
$ sudo touch /etc/pki/CA/index.txt
```

2. The **/etc/pki/CA/serial** file identifies the next serial number to use for the next certificate to sign. Check if this file exists. If the file does not exist, create a new file with a new starting value:

```
$ echo '1000' | sudo tee /etc/pki/CA/serial
```

## 6.2. CREATING A CERTIFICATE AUTHORITY

Normally you sign your SSL/TLS certificates with an external certificate authority. In some situations, you might want to use your own certificate authority. For example, you might want to have an internal-only certificate authority.

### Procedure

1. Generate a key and certificate pair to act as the certificate authority:

```
$ openssl genrsa -out ca.key.pem 4096
$ openssl req  -key ca.key.pem -new -x509 -days 7300 -extensions v3_ca -out ca.crt.pem
```

2. The **openssl req** command requests certain details about your authority. Enter these details at the prompt. These commands create a certificate authority file called **ca.crt.pem**.

3. Set the certificate location as the value for the **PublicTLSCAFile** parameter in the **enable-tls.yaml** file. When you set the certificate location as the value for the **PublicTLSCAFile** parameter, you ensure that the CA certificate path is added to the **clouds.yaml** authentication file.

```
parameter_defaults:
  PublicTLSCAFile: /etc/pki/ca-trust/source/anchors/cacert.pem
```

## 6.3. ADDING THE CERTIFICATE AUTHORITY TO CLIENTS

For any external clients aiming to communicate using SSL/TLS, copy the certificate authority file to each client that requires access to your Red Hat OpenStack Platform environment.

**Procedure**

1. Copy the certificate authority to the client system:

   ```
   $ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
   ```

2. After you copy the certificate authority file to each client, run the following command on each client to add the certificate to the certificate authority trust bundle:

   ```
   $ sudo update-ca-trust extract
   ```

## 6.4. CREATING AN SSL/TLS KEY

Enabling SSL/TLS on an OpenStack environment requires an SSL/TLS key to generate your certificates.

**Procedure**

1. Run the following command to generate the SSL/TLS key (**server.key.pem**):

   ```
   $ openssl genrsa -out server.key.pem 2048
   ```

## 6.5. CREATING AN SSL/TLS CERTIFICATE SIGNING REQUEST

Complete the following steps to create a certificate signing request.

**Procedure**

1. Copy the default OpenSSL configuration file:

   ```
   $ cp /etc/pki/tls/openssl.cnf .
   ```

2. Edit the new **openssl.cnf** file and configure the SSL parameters that you want to use for director. An example of the types of parameters to modify include:

   ```
   [req]
   distinguished_name = req_distinguished_name
   req_extensions = v3_req

   [req_distinguished_name]
   countryName = Country Name (2 letter code)
   countryName_default = AU
   stateOrProvinceName = State or Province Name (full name)
   ```

```
stateOrProvinceName_default = Queensland
localityName = Locality Name (eg, city)
localityName_default = Brisbane
organizationalUnitName = Organizational Unit Name (eg, section)
organizationalUnitName_default = Red Hat
commonName = Common Name
commonName_default = 192.168.0.1
commonName_max = 64

[ v3_req ]
# Extensions to add to a certificate request
basicConstraints = CA:FALSE
keyUsage = nonRepudiation, digitalSignature, keyEncipherment
subjectAltName = @alt_names

[alt_names]
IP.1 = 192.168.0.1
DNS.1 = instack.localdomain
DNS.2 = vip.localdomain
DNS.3 = 192.168.0.1
```

Set the **commonName_default** to one of the following entries:

- If you are using an IP address to access director over SSL/TLS, use the **undercloud_public_host** parameter in the **undercloud.conf** file.
- If you are using a fully qualified domain name to access director over SSL/TLS, use the domain name.

Edit the **alt_names** section to include the following entries:

- **IP** - A list of IP addresses that clients use to access director over SSL.
- **DNS** - A list of domain names that clients use to access director over SSL. Also include the Public API IP address as a DNS entry at the end of the **alt_names** section.

NOTE

For more information about **openssl.cnf**, run the **man openssl.cnf** command.

3. Run the following command to generate a certificate signing request (**server.csr.pem**):

```
$ openssl req -config openssl.cnf -key server.key.pem -new -out server.csr.pem
```

Ensure that you include your OpenStack SSL/TLS key with the **-key** option.

This command generates a **server.csr.pem** file, which is the certificate signing request. Use this file to create your OpenStack SSL/TLS certificate.

## 6.6. CREATING THE SSL/TLS CERTIFICATE

To generate the SSL/TLS certificate for your OpenStack environment, the following files must be present:

**openssl.cnf**

The customized configuration file that specifies the v3 extensions.

**server.csr.pem**

The certificate signing request to generate and sign the certificate with a certificate authority.

**ca.crt.pem**

The certificate authority, which signs the certificate.

**ca.key.pem**

The certificate authority private key.

### Procedure

1. Create the **newcerts** directory if it does not already exist:

```
sudo mkdir -p /etc/pki/CA/newcerts
```

2. Run the following command to create a certificate for your undercloud or overcloud:

```
$ sudo openssl ca -config openssl.cnf -extensions v3_req -days 3650 -in server.csr.pem -out
server.crt.pem -cert ca.crt.pem -keyfile ca.key.pem
```

   This command uses the following options:

   **-config**

   Use a custom configuration file, which is the **openssl.cnf** file with v3 extensions.

   **-extensions v3_req**

   Enabled v3 extensions.

   **-days**

   Defines how long in days until the certificate expires.

   **-in**

   The certificate signing request.

   **-out**

   The resulting signed certificate.

   **-cert**

   The certificate authority file.

   **-keyfile**

   The certificate authority private key.

This command creates a new certificate named **server.crt.pem**. Use this certificate in conjunction with your OpenStack SSL/TLS key

## 6.7. ADDING THE CERTIFICATE TO THE UNDERCLOUD

Complete the following steps to add your OpenStack SSL/TLS certificate to the undercloud trust bundle.

### Procedure

1. Run the following command to combine the certificate and key:

```
$ cat server.crt.pem server.key.pem > undercloud.pem
```

This command creates a **undercloud.pem** file.

2. Copy the **undercloud.pem** file to a location within your **/etc/pki** directory and set the necessary SELinux context so that HAProxy can read it:

```
$ sudo mkdir /etc/pki/undercloud-certs
$ sudo cp ~/undercloud.pem /etc/pki/undercloud-certs/.
$ sudo semanage fcontext -a -t etc_t "/etc/pki/undercloud-certs(/.*)?"
$ sudo restorecon -R /etc/pki/undercloud-certs
```

3. Add the **undercloud.pem** file location to the **undercloud_service_certificate** option in the **undercloud.conf** file:

```
undercloud_service_certificate = /etc/pki/undercloud-certs/undercloud.pem
```

Do not set or enable the **generate_service_certificate** and **certificate_generation_ca** parameters. Director uses these parameters to automatically generate a certificate instead of using the **undercloud.pem** certificate you created manually.

4. Add the certificate authority that signed the certificate to the list of trusted Certificate Authorities on the undercloud so that different services within the undercloud have access to the certificate authority:

```
$ sudo cp ca.crt.pem /etc/pki/ca-trust/source/anchors/
$ sudo update-ca-trust extract
```

To verify the certificate authority was added to the undercloud, use **openssl** to check the trust bundle:

```
$ openssl crl2pkcs7 -nocrl -certfile /etc/pki/tls/certs/ca-bundle.crt | openssl pkcs7 -print_certs
-text | grep <CN of the CA issuer> -A 10 -B 10
```

Replace **<CN of the CA issuer>** with the common name of the issuer of the CA. This command outputs the main certificate details, including the validity dates.

# CHAPTER 7. ENABLING SSL/TLS ON OVERCLOUD PUBLIC ENDPOINTS

By default, the overcloud uses unencrypted endpoints for the overcloud services. To enable SSL/TLS in your overcloud, Red Hat recommends that you use a certificate authority (CA) solution.

When you use a CA solution, you have production ready solutions such as a certificate renewals, certificate revocation lists (CRLs), and industry accepted cryptography. For information on using Red Hat Identity Manager (IdM) as a CA, see Implementing TLS-e with Ansible .

You can use the following manual process to enable SSL/TLS for Public API endpoints only, the Internal and Admin APIs remain unencrypted. You must also manually update SSL/TLS certificates if you do not use a CA. For more information, see Manually updating SSL/TLS certificates .

### Prerequisites

- Network isolation to define the endpoints for the Public API.
- The **openssl-perl** package is installed.
- You have an SSL/TLS certificate. For more information see Configuring custom SSL/TLS certificates.

## 7.1. ENABLING SSL/TLS

To enable SSL/TLS in your overcloud, you must create an environment file that contains parameters for your SSL/TLS certiciates and private key.

### Procedure

1. Copy the **enable-tls.yaml** environment file from the heat template collection:

   ```
   $ cp -r /usr/share/openstack-tripleo-heat-templates/environments/ssl/enable-tls.yaml
   ~/templates/.
   ```

2. Edit this file and make the following changes for these parameters:

   **SSLCertificate**

   Copy the contents of the certificate file (**server.crt.pem**) into the **SSLCertificate** parameter:

   ```
   parameter_defaults:
     SSLCertificate: |
       -----BEGIN CERTIFICATE-----
       MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGS
       ...
       sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQ
       -----END CERTIFICATE-----
   ```

   **IMPORTANT**

   The certificate contents require the same indentation level for all new lines.

SSLIntermediateCertificate

If you have an intermediate certificate, copy the contents of the intermediate certificate into the **SSLIntermediateCertificate** parameter:

```
parameter_defaults:
  SSLIntermediateCertificate: |
    -----BEGIN CERTIFICATE-----
    sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvpBCwUAMFgxCzAJB
    ...
    MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQE
    -----END CERTIFICATE-----
```

IMPORTANT

The certificate contents require the same indentation level for all new lines.

SSLKey

Copy the contents of the private key (**server.key.pem**) into the **SSLKey** parameter:

```
parameter_defaults:
  ...
  SSLKey: |
    -----BEGIN RSA PRIVATE KEY-----
    MIIEowIBAAKCAQEAqVw8lnQ9RbeI1EdLN5PJP0lVO
    ...
    ctlKn3rAAdyumi4JDjESAXHIKFjJNOLrBmpQyES4X
    -----END RSA PRIVATE KEY-----
```

IMPORTANT

The private key contents require the same indentation level for all new lines.

## 7.2. INJECTING A ROOT CERTIFICATE

If the certificate signer is not in the default trust store on the overcloud image, you must inject the certificate authority into the overcloud image.

### Procedure

1. Copy the **inject-trust-anchor-hiera.yaml** environment file from the heat template collection:

```
$ cp -r /usr/share/openstack-tripleo-heat-templates/environments/ssl/inject-trust-anchor-hiera.yaml ~/templates/.
```

Edit this file and make the following changes for these parameters:

CAMap

Lists each certificate authority content (CA) to inject into the overcloud. The overcloud requires the

CA files used to sign the certificates for both the undercloud and the overcloud. Copy the contents of the root certificate authority file (**ca.crt.pem**) into an entry. For example, your **CAMap** parameter might look like the following:

```
parameter_defaults:
  CAMap:
    ...
    undercloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDlTCCAn2gAwIBAgIJAOnPtx2hHEhrMA0GCS
        BAYTAlVTMQswCQYDVQQIDAJOQzEQMA4GA1UEBw
        UmVkIEhhdDELMAkGA1UECwwCUUUxFDASBgNVBA
        -----END CERTIFICATE-----
    overcloud-ca:
      content: |
        -----BEGIN CERTIFICATE-----
        MIIDBzCCAe+gAwIBAgIJAIc75A7FD++DMA0GCS
        BAMMD3d3dy5leGFtcGxlLmNvbTAeFw0xOTAxMz
        Um54yGCARyp3LpkxvyfMXX1DokpS1uKi7s6CkF
        -----END CERTIFICATE-----
```

**IMPORTANT**

The certificate authority contents require the same indentation level for all new lines.

You can also inject additional CAs with the **CAMap** parameter.

## 7.3. CONFIGURING DNS ENDPOINTS

If you use a DNS hostname to access the overcloud through SSL/TLS, copy the **/usr/share/openstack-tripleo-heat-templates/environments/predictable-placement/custom-domain.yaml** file into the **/home/stack/templates** directory.

**NOTE**

It is not possible to redeploy with a TLS-everywhere architecture if this environment file is not included in the initial deployment.

Configure the host and domain names for all fields, adding parameters for custom networks if needed:

**CloudDomain**

the DNS domain for hosts.

**CloudName**

The DNS hostname of the overcloud endpoints.

**CloudNameCtlplane**

The DNS name of the provisioning network endpoint.

**CloudNameInternal**

The DNS name of the Internal API endpoint.

**CloudNameStorage**

The DNS name of the storage endpoint.

**CloudNameStorageManagement**

The DNS name of the storage management endpoint.

**Procedure**

- Use one of the following parameters to add the DNS servers to use:
  - **DEFAULT/undercloud_nameservers**
  - **%SUBNET_SECTION%/dns_nameservers**

## 7.4. ADDING ENVIRONMENT FILES DURING OVERCLOUD CREATION

Use the **-e** option with the deployment command **openstack overcloud deploy** to include environment files in the deployment process. Add the environment files from this section in the following order:

- The environment file to enable SSL/TLS (**enable-tls.yaml**)
- The environment file to set the DNS hostname (**custom-domain.yaml**)
- The environment file to inject the root certificate authority (**inject-trust-anchor-hiera.yaml**)
- The environment file to set the public endpoint mapping:
  - If you use a DNS name for accessing the public endpoints, use **/usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml**
  - If you use a IP address for accessing the public endpoints, use **/usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-ip.yaml**

**Procedure**

- Use the following deployment command snippet as an example of how to include your SSL/TLS environment files:

```
$ openstack overcloud deploy --templates \
[...]
-e /home/stack/templates/enable-tls.yaml \
-e ~/templates/custom-domain.yaml \
-e ~/templates/inject-trust-anchor-hiera.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-dns.yaml
```

## 7.5. MANUALLY UPDATING SSL/TLS CERTIFICATES

Complete the following steps if you are using your own SSL/TLS certificates that are not auto-generated from the TLS everywhere (TLS-e) process.

**Procedure**

1. Edit your heat templates with the following content:
   - Edit the **enable-tls.yaml** file and update the **SSLCertificate**, **SSLKey**, and **SSLIntermediateCertificate** parameters.

- If your certificate authority has changed, edit the **inject-trust-anchor-hiera.yaml** file and update the **CAMap** parameter.

2. Rerun the deployment command:

```
$ openstack overcloud deploy --templates \
[...]
-e /home/stack/templates/enable-tls.yaml \
-e ~/templates/custom-domain.yaml \
-e ~/templates/inject-trust-anchor-hiera.yaml \
-e /usr/share/openstack-tripleo-heat-templates/environments/ssl/tls-endpoints-public-
dns.yaml
```

3. On the director, run the following command for each Controller:

```
ssh tripleo-admin@<controller> sudo podman \
restart $(podman ps --format="{{.Names}}" | grep -w -E 'haproxy(-bundle-.*-[0-9]+)?')
```

# CHAPTER 8. USING FERNET KEYS FOR ENCRYPTION IN THE OVERCLOUD

Fernet is the default token provider, that replaces **uuid**. You can review your Fernet deployment and rotate the Fernet keys. Fernet uses three types of keys, which are stored in **/var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys**. The highest-numbered directory contains the primary key, which generates new tokens and decrypts existing tokens.

Fernet key rotation uses the following process:

1. The primary key becomes the secondary key.
2. The <system> issues a new primary key. The outgoing primary key is no longer valid. You can use secondary keys to decrypt tokens that were associated with previous primary keys, but you cannot issue new tokens.

When you decide the length of Fernet key rotation cycles, follow the security posture of your organization. If your organization does not have guidance, a monthly rotation cycle is good practice for security reasons.

## 8.1. REVIEWING THE FERNET DEPLOYMENT

To test that Fernet tokens are working correctly, retrieve the IP address of the Controller node, SSH into the Controller node, and review the settings of the token driver and provider.

### Procedure

1. Retrieve the IP address of the Controller node:

```
[stack@director ~]$ source ~/stackrc
[stack@director ~]$ openstack server list
--------------------------------------------------------------------------------------+
| ID                                   | Name                    | Status | Networks          |
--------------------------------------------------------------------------------------+
| 756fbd73-e47b-46e6-959c-e24d7fb71328 | overcloud-controller-0  | ACTIVE |
ctlplane=192.0.2.16 |
| 62b869df-1203-4d58-8e45-fac6cd4cfbee | overcloud-novacompute-0 | ACTIVE |
ctlplane=192.0.2.8  |
--------------------------------------------------------------------------------------+
```

2. SSH into the Controller node:

```
[tripleo-admin@overcloud-controller-0 ~]$ ssh tripleo-admin@192.0.2.16
```

3. Retrieve the values of the token driver and provider settings:

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo crudini --get /var/lib/config-data/puppet-
generated/keystone/etc/keystone/keystone.conf token driver
sql
[tripleo-admin@overcloud-controller-0 ~]$ sudo crudini --get /var/lib/config-data/puppet-
generated/keystone/etc/keystone/keystone.conf token provider
fernet
```

4. Test the Fernet provider:

```
[tripleo-admin@overcloud-controller-0 ~]$ exit
[stack@director ~]$ source ~/overcloudrc
[stack@director ~]$ openstack token issue
--------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
| Field | Value |
--------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
| expires | 2016-09-20 05:26:17+00:00 |
| id | gAAAAABX4LppE8vaiFZ992eah2i3edpO1aDFxlKZq6a_RJzxUx56QVKORrmW0-oZK3-
Xuu2wcnpYq_eek2SGLz250eLpZOzxKBR0GsoMfxJU8mEFF8NzfLNcbuS-iz7SV-
N1re3XEywSDG90JcgwjQfXW-8jtCm-n3LL5IaZexAYIw059T_-cd8 |
| project_id | 26156621d0d54fc39bf3adb98e63b63d |
| user_id | 397daf32cadd490a8f3ac23a626ac06c |
--------------------------------------------------------------------------------------------------------------
-----------------------------------------------------------------------------------+
```

The result includes the long Fernet token.

## 8.2. ROTATING THE FERNET KEYS BY USING THE WORKFLOW SERVICE

To ensure that the Fernet keys persist after stack updates, rotate the keys with the Workflow service (mistral). By default, director manages the overcloud Fernet keys in an environment file with the **ManageKeystoneFernetKeys** parameter. The Fernet keys are stored in the Workflow service, in the **KeystoneFernetKeys** section.

### Procedure

1. Review the existing Fernet keys:

    a. Identify the Fernet key location. Log in to a Controller node as the tripleo-admin user and use the **crudini** command to query the Fernet keys:

    ```
    [stack@<undercloud_host> ~]$ ssh tripleo-admin@overcloud-controller-o
    [tripleo-admin@overcloud-controller-0 ~]$ sudo crudini --get /var/lib/config-data/puppet-
    generated/keystone/etc/keystone/keystone.conf fernet_tokens key_repository
    /etc/keystone/fernet-keys
    ```

    NOTE

    The **/etc/keystone/** directory refers to the container file system path.

    b. Inspect the current Fernet key directories:

    ```
    [tripleo-admin@overcloud-controller-0 ~]$ sudo ls /var/lib/config-data/puppet-
    generated/keystone/etc/keystone/fernet-keys
    0  1  2
    ```

    - **0** - Contains the staged key, which becomes the next primary key and is always numbered **0**.

- **1** - Contains the secondary key.
- **2** - Contains the primary key. This number increments each time that the keys rotate. The highest number always serves as the primary key.

NOTE

- The maximum number of keys is set with **max_active_keys** property. The default is 5 keys.
- The keys propagate across all Controller nodes.

2. Rotate the Fernet keys by using the **workflow** command:

```
[stack@director ~]$ source ~/stackrc
[stack@director ~]$ openstack workflow execution create
tripleo.fernet_keys.v1.rotate_fernet_keys {"container": "overcloud"}
------------------------------------------------------------+
| Field             | Value                                  |
------------------------------------------------------------+
| ID                | 58c9c664-b966-4f82-b368-af5ed8de5b47   |
| Workflow ID       | 78f0990a-3d34-4bf2-a127-10c149bb275c   |
| Workflow name     | tripleo.fernet_keys.v1.rotate_fernet_keys |
| Description       |                                        |
| Task Execution ID | <none>                                 |
| State             | RUNNING                                |
| State info        | None                                   |
| Created at        | 2017-12-20 11:13:50                    |
| Updated at        | 2017-12-20 11:13:50                    |
------------------------------------------------------------+
```

**Verification**

1. Retrieve the ID and ensure that the workflow is successful.

```
[stack@director ~]$ openstack workflow execution show 58c9c664-b966-4f82-b368-
af5ed8de5b47
------------------------------------------------------------+
| Field             | Value                                  |
------------------------------------------------------------+
| ID                | 58c9c664-b966-4f82-b368-af5ed8de5b47   |
| Workflow ID       | 78f0990a-3d34-4bf2-a127-10c149bb275c   |
| Workflow name     | tripleo.fernet_keys.v1.rotate_fernet_keys |
| Description       |                                        |
| Task Execution ID | <none>                                 |
| State             | SUCCESS                                |
| State info        | None                                   |
| Created at        | 2017-12-20 11:13:50                    |
| Updated at        | 2017-12-20 11:15:00                    |
------------------------------------------------------------+
```

2. On the Controller node, review the number of Fernet keys, and compare with the previous result.

```
[tripleo-admin@overcloud-controller-0 ~]$ sudo ls /var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys
0  1  2  3
```

- **0** - Contains the staged key and always be numbered **0**. This key becomes a primary key during the next rotation.
- **1 & 2** - Contain the secondary keys.
- **3** - Contains the primary key. This number increments each time the keys rotate. The highest number always serves as the primary key.

NOTE

- The maximum number of keys is set with the **max_active_keys** property. The default is 5 keys.
- The keys propagate across all Controller nodes.

# CHAPTER 9. FEDERAL INFORMATION PROCESSING STANDARD ON RED HAT OPENSTACK PLATFORM

**IMPORTANT**

This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

The Federal Information Processing Standards (FIPS) is a set of security requirements developed by the National Institute of Standards and Technology (NIST). In Red Hat Enterprise Linux 9, the supported standard is FIPS publication 140-3: *Security Requirements for Cryptographic Modules*. For details about the supported standard, see the Federal Information Processing Standards Publication 140-3.

These security requirements define acceptable cryptographic algorithms and the use of those cryptographic algorithms, including security modules.

- FIPS 140-3 validation is achieved by using only those cryptographic algorithms approved through FIPS, in the manner prescribed, and through validated modules.
- FIPS 140-3 compatibility is achieved by using only those cryptographic algorithms approved through FIPS.

Red Hat OpenStack Platform 17 is **FIPS 140-3 compatible**. You can take advantage of FIPS compatibility by using images provided by Red Hat to deploy your overcloud.

**NOTE**

OpenStack 17.1 is based on Red Hat Enterprise Linux (RHEL) 9.2. RHEL 9.2 has not yet been submitted for FIPS validation. Red Hat expects, though cannot commit to a specific timeframe, to obtain FIPS validation for RHEL 9.0 and RHEL 9.2 modules, and later even minor releases of RHEL 9.x. Updates will be available in Compliance Activities and Government Standards.

## 9.1. ENABLING FIPS

When you enable FIPS, you must complete a series of steps during the installation of the undercloud and overcloud.

### Prerequisites

- You have installed Red Hat Enterprise Linux and are prepared to begin the installation of Red Hat OpenStack Platform director.

### Procedure

1. Enable FIPS on the undercloud:
    a. Enable FIPS on the system on which you plan to install the undercloud:

    ```
    fips-mode-setup --enable
    ```

NOTE

This step will add the **fips=1** kernel parameter to your GRUB configuration file. As a result, only cryptographic algorithms modules used by Red Hat Enterprise Linux are in FIPS mode and only cryptographic algorithms approved by the standard are used.

b. Reboot the system.

c. Verify that FIPS is enabled:

```
fips-mode-setup --check
```

d. Install and configure Red Hat OpenStack Platform director. For more information see: Installing director on the undercloud.

2. Prepare FIPS-enabled images for the overcloud.

a. Install images for the overcloud:

```
sudo dnf -y install rhosp-director-images-uefi-fips-x86_64
```

b. Create the **images** directory in the home directory of the **stack** user:

```
$ mkdir /home/stack/images
$ cd /home/stack/images
```

c. Extract the images to your home directory:

```
for i in /usr/share/rhosp-director-images/*fips*.tar; do tar -xvf $i; done
```

d. You must create symlinks before uploading the images:

```
ln -s ironic-python-agent-fips.initramfs      ironic-python-agent.initramfs
ln -s ironic-python-agent-fips.kernel         ironic-python-agent.kernel
ln -s overcloud-hardened-uefi-full-fips.qcow2  overcloud-hardened-uefi-full.qcow2
```

e. Upload the FIPS-enabled overcloud images to the Image service:

```
 openstack overcloud image upload --update-existing --whole-disk
```

NOTE

You must use the **--update-existing** flag even if there are no images currently in the OpenStack Image service.

3. Enable FIPS on the overcloud.
Configure templates for an overcloud deployment specific to your environment. Include all configuration templates in the deployment command, including fips.yaml:

```
openstack overcloud deploy
...
-e /usr/share/openstack-tripleo-heat-templates/environents/fips.yaml
```

# CHAPTER 10. IMPROVING USER ACCESS SECURITY

IMPORTANT

This feature is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment. For more information about Technology Preview features, see Scope of Coverage Details.

You can enable secure role-based access control (SRBAC) in Red Hat OpenStack Platform 17. The SRBAC model has three personas, based on three roles existing within the project scope.

## 10.1. SRBAC PERSONAS

Personas are a combination of roles and the scope to which they belong. When you deploy Red Hat OpenStack Platform 17, you can assign any of the personas from the project scope.

### 10.1.1. Red Hat OpenStack Platform SRBAC roles

Currently, three different roles are available within the project scope.

admin

The **admin** role includes all create, read, update, or delete operations on a resource or API.

member

The **member** role is allowed to create, read, update, and delete resources that are owned by the scope in which they are a member.

reader

The **reader** role is for read-only operations, regardless of the scope it is applied to. This role can view resources across the entirety of the scope to which it is applied.

### 10.1.2. Red Hat OpenStack Platform SRBAC scope

The scope is the context in which operations are performed. Only the **project** scope is available in Red Hat OpenStack Platform 17. The **project** scope is a contained subset of APIs for isolated self-service resources within OpenStack.

### 10.1.3. Red Hat OpenStack Platform SRBAC personas

Project admin

Because the project admin persona is the only administrative persona available, Red Hat OpenStack Platform 17 includes modified policies that grant the project admin persona the highest level of authorization. This persona includes create, read, update and delete operations on resources across projects, which includes adding and removing users and other projects.

NOTE

This persona is expected to change in scope with future development. This role implies all permissions granted to project members and project readers.

Project member

The project member persona is for users who are granted permission to consume resources within the project scope. This persona can create, list, update, and delete resources within the project to which they are assigned. This persona implies all permissions granted to project readers.

**Project reader**

The project reader persona is for users who are granted permission to view non-sensitive resources in the project. On projects, assign the reader role to end users who need to inspect or view resources, or to auditors, who only need to view project-specific resources within a single project for the purposes of an audit The project-reader persona will not address all auditing use cases.

NOTE

Additional personas based on the **system** or **domain** scopes are in development and are not available for use.

## 10.2. ACTIVATING SECURE ROLE-BASED ACCESS CONTROL

When you activate secure role-based Authentication, you are activating a new set of policy files that define the scope of permissions assigned to users in your Red Hat OpenStack Platform environment.

### Prerequisites

- You have an installed Red Hat OpenStack Platform director environment.

### Procedure

- Include the **enable-secure-rbac.yaml** environment file in the deployment script when deploying Red Hat OpenStack Platform:

```
openstack overcloud deploy --templates
…
-e /usr/share/openstack-tripleo-heat-templates/environments/enable-secure-rbac.yaml
```

## 10.3. ASSIGNING ROLES IN AN SRBAC ENVIRONMENT

With SRBAC on Red Hat OpenStack Platform, you can assign users to the role of **project-admin**, **project-member**, or **project-reader**.

### Prerequisites

- You have deployed Red Hat OpenStack Platform with secure role based authentication (SRBAC).

### Procedure

- Use the **openstack role add** command using the following syntax:
  - Assign the **admin** role:

    ```
    $ openstack role add --user <user> --user-domain <domain> --project <project> --
    project-domain <project-domain> admin
    ```

  - Assign the **member** role:

```
$ openstack role add --user <user> --user-domain <domain> --project <project>  --
project-domain <project-domain> member
```

- Assign the **reader** role:

```
$ openstack role add --user <user> --user-domain <domain> --project <project> --
project-domain <project-domain> reader
```

- Replace **<user>** with an existing user to apply the role.
- Replace **<domain>** with the domain to which the role applies.
- Replace **<project>** with the project for which the user is being granted the role.
- Replace **<project-domain>** with the domain that the project is in.

# CHAPTER 11. POLICIES

Each OpenStack service contains resources that are managed by access policies. For example, a resource might include the following functions:

- Permission to create and start instances
- The ability to attach a volume to an instance

If you are a Red Hat OpenStack Platform (RHOSP) administrator, you can create custom policies to introduce new roles with varying levels of access, or to change the default behavior of existing roles.

**IMPORTANT**

Red Hat does not support customized roles or policies. Syntax errors or misapplied authorization can negatively impact security or usability. If you need customized roles or policies in your production environment, contact Red Hat support for a support exception.

## 11.1. REVIEWING EXISTING POLICIES

Policy files for services traditionally existed in the **/etc/$service** directory. For example, the full path of the **policy.json** file for Compute (nova) was **/etc/nova/policy.json**.

There are two important architectural changes that affect how you can find existing policies:

- Red Hat OpenStack Platform is now containerized.
  - Policy files, if present, are in the traditional path if you view them from inside the service container:
    /etc/$service/policy.json
  - Policy files, if present, are in the following path if you view them from outside the service container:
    /var/lib/config-data/puppet-generated/$service/etc/$service/policy.json
- Each service has default policies that are provided in code, with files that are available only if you created them manually, or if they are generated with **oslopolicy** tooling. To generate a policy file, use the **oslopolicy-policy-generator** from within a container, as in the following example:

```
podman exec -it keystone oslopolicy-policy-generator --namespace keystone
```

By default, generated policies are pushed to stdout by osly.policy CLI tools.

## 11.2. UNDERSTANDING SERVICE POLICIES

Service policy file statements are either alias definitions or rules. Alias definitions exist at the top of the file. The following list contains an explanation of the alias definitions from the generated **policy.json** file for Compute (nova):

- "context_is_admin": "role:admin"
  When **rule:context_is_admin** appears after a target, the policy checks that the user is operating with an administrative context before it allows that action.

- "admin_or_owner": "is_admin:True or project_id:%(project_id)s"
  When **admin_or_owner** appears after a target, the policy checks that the user is either an admin, or that their project ID matches the owning project ID of the target object before it allows that action.
- "admin_api": "is_admin:True
  When **admin_api** appears after a target, the policy checks that the user is an admin before it allows that action.

## 11.3. POLICY SYNTAX

Policy.json files support certain operators so that you can control the target scope of these settings. For example, the following keystone setting contains the rule that only admin users can create users:

```
"identity:create_user": "rule:admin_required"
```

The section to the left of the **:** character describes the privilege, and the section to the right defines who can use the privilege. You can also use operators to the right side to further control the scope:

- **!** - No user (including admin) can perform this action.
- @ and **""** - Any user can perform this action.
- **not**, **and**, **or** - Standard operator functions are available.

For example, the following setting means that no users have permission to create new users:

```
"identity:create_user": "!"
```

## 11.4. USING POLICY FILES FOR ACCESS CONTROL

IMPORTANT

Red Hat does not support customized roles or policies. Syntax errors or misapplied authorization can negatively impact security or usability. If you need customized roles or policies in your production environment, contact Red Hat support for a support exception.

To override the default rules, edit the **policy.json** file for the appropriate OpenStack service. For example, the Compute service has a **policy.json** in the nova directory, which is the correct location of the file for containerized services when you view it from inside the container.

NOTE

- You must thoroughly test changes to policy files in a staging environment before implementing them in production.
- You must check that any changes to the access control policies do not unintentionally weaken the security of any resource. In addition, any changes to a **policy.json** file are effective immediately and do not require a service restart.

**Example: Creating a power user role**

To customize the permissions of a keystone role, update the **policy.json** file of a service. This means that you can more granularly define the permissions that you assign to a class of users. This example creates a *power user* role for your deployment with the following privileges:

- Start an instance.
- Stop an instance.
- Manage the volumes that are attached to instances.

The intention of this role is to grant additional permissions to certain users, without the need to then grant **admin** access. To use these privileges, you must grant the following permissions to a custom role:

- Start an instance: **"os_compute_api:servers:start": "role:PowerUsers"**
- Stop an instance: **"os_compute_api:servers:stop": "role:PowerUsers"**
- Configure an instance to use a particular volume: **"os_compute_api:servers:create:attach_volume": "role:PowerUsers"**
- List the volumes that are attached to an instance: **"os_compute_api:os-volumes-attachments:index": "role:PowerUsers"**
- Attach a volume: **"os_compute_api:os-volumes-attachments:create": "role:PowerUsers"**
- View the details of an attached volume: **"os_compute_api:os-volumes-attachments:show": "role:PowerUsers"**
- Change the volume that is attached to an instance: **"os_compute_api:os-volumes-attachments:update": "role:PowerUsers"**
- Delete a volume that is attached to an instance: **"os_compute_api:os-volumes-attachments:delete": "role:PowerUsers"**

NOTE

When you modify the **policy.json** file, you override the default policy. As a result, members of **PowerUsers** are the only users that can perform these actions. To allow **admin** users to retain these permissions, you can create rules for *admin_or_power_user*. You can also use some basic conditional logic to define **role:PowerUsers or role:Admin**.

1. Create the custom keystone role:

```
$ openstack role create PowerUsers
+-----------+----------------------------------+
| Field     | Value                            |
+-----------+----------------------------------+
| domain_id | None                             |
| id        | 7061a395af43455e9057ab631ad49449 |
| name      | PowerUsers                       |
+-----------+----------------------------------+
```

2. Add an existing user to the role, and assign the role to a project:

```
$ openstack role add --project [PROJECT_NAME] --user [USER_ID] [PowerUsers-ROLE_ID]
```

NOTE

A role assignment is paired exclusively with one project. This means that when you assign a role to a user, you also define the target project at the same time. If you want the user to receive the same role but for a different project, you must assign the role to them again separately but target the different project.

3. View the default nova policy settings:

```
$ oslopolicy-policy-generator --namespace nova
```

4. Create custom permissions for the new **PowerUsers** role by adding the following entries to **/var/lib/config-data/puppet-generated/nova/etc/nova/policy.json**:

NOTE

Test your policy changes before deployment to verify that they work as you expect.

```
{
"os_compute_api:servers:start": "role:PowerUsers",
"os_compute_api:servers:stop": "role:PowerUsers",
"os_compute_api:servers:create:attach_volume": "role:PowerUsers",
"os_compute_api:os-volumes-attachments:index": "role:PowerUsers",
"os_compute_api:os-volumes-attachments:create": "role:PowerUsers",
"os_compute_api:os-volumes-attachments:show": "role:PowerUsers",
"os_compute_api:os-volumes-attachments:update": "role:PowerUsers",
"os_compute_api:os-volumes-attachments:delete": "role:PowerUsers"
}
```

You implement the changes when you save this file and restart the nova container. Users that are added to the **PowerUsers** keystone role receive these privileges.

## 11.5. EXAMPLE: LIMITING ACCESS BASED ON ATTRIBUTES

IMPORTANT

Red Hat does not support customized roles or policies. Syntax errors or misapplied authorization can negatively impact security or usability. If you need customized roles or policies in your production environment, contact Red Hat support for a support exception.

You can create policies that will restrict access to API calls based on the attributes of the user making that API call. For example, the following default rule states that keypair deletion is allowed if run from an administrative context, or if the user ID of the token matches the user ID associated with the target.

```
"os_compute_api:os-keypairs:delete": "rule:admin_api or user_id:%(user_id)s"
```

NOTE: * Newly implemented features are not guaranteed to be in every service with each release. Therefore, it is important to write rules using the conventions of the target service's existing policies. For details on viewing these policies, see Reviewing existing policies. * All policies should be rigorously tested in a non-production environment for every version on which they will be deployed, as policies are not guaranteed for compatibility across releases.

Based on the above example, you can craft API rules to expand or restrict access to users based on whether or not they own a resource. Additionally, attributes can be combined with other restrictions to form rules as seen in the example below:

```
"admin_or_owner": "is_admin:True or project_id:%(project_id)s"
```

Considering the examples above, you can create a unique rule limited to administrators and users, and then use that rule to further restrict actions:

```
"admin_or_user": "is_admin:True or user_id:%(user_id)s"
"os_compute_api:os-instance-actions": "rule:admin_or_user"
```

**Additional resources**

- Policy syntax.

## 11.6. MODIFYING POLICIES WITH HEAT

**IMPORTANT**

Red Hat does not support customized roles or policies. Syntax errors or misapplied authorization can negatively impact security or usability. If you need customized roles or policies in your production environment, contact Red Hat support for a support exception.

You can use heat to configure access policies for certain services in the overcloud. Use the following parameters to set policies on the respective services:

**Table 11.1. Policy Parameters**

| Parameter | Description |
|---|---|
| KeystonePolicies | A hash of policies to configure for OpenStack Identity (keystone). |
| IronicApiPolicies | A hash of policies to configure for OpenStack Bare Metal (ironic) API. |
| BarbicanPolicies | A hash of policies to configure for OpenStack Key Manager (barbican). |
| NeutronApiPolicies | A hash of policies to configure for OpenStack Networking (neutron) API. |
| SaharaApiPolicies | A hash of policies to configure for OpenStack Clustering (sahara) API. |

| Parameter | Description |
|---|---|
| NovaApiPolicies | A hash of policies to configure for OpenStack Compute (nova) API. |
| CinderApiPolicies | A hash of policies to configure for OpenStack Block Storage (cinder) API. |
| GlanceApiPolicies | A hash of policies to configure for OpenStack Image Storage (glance) API. |
| HeatApiPolicies | A hash of policies to configure for OpenStack Orchestration (heat) API. |

To configure policies for a service, give the policy parameter a hash value that contains the service's policies For example:

- OpenStack Identity (keystone) uses the **KeystonePolicies** parameter. Set this parameter in the **parameter_defaults** section of an environment file:

```
parameter_defaults:
  KeystonePolicies: { keystone-context_is_admin: { key: context_is_admin, value: 'role:admin' } }
```

- OpenStack Compute (nova) uses the **NovaApiPolicies** parameter. Set this parameter in the **parameter_defaults** section of an environment file:

```
parameter_defaults:
  NovaApiPolicies: { nova-context_is_admin: { key: 'compute:get_all', value: '@' } }
```

## 11.7. AUDITING YOUR USERS AND ROLES

You can use tools available in Red Hat OpenStack Platform to build a report of role assignments per user and associated privileges.

### Prerequisites

- You have an installed Red Hat OpenStack Platform environment.
- You are logged into the director as stack.

### Procedure

1. Run the **openstack role list** command to see the roles currently in your environment:

```
openstack role list -c Name -f value

swiftoperator
ResellerAdmin
```

```
admin
_member_
heat_stack_user
```

2. Run the **openstack role assignment list** command to list all users that are members of a particular role. For example, to see all users that have the admin role, run the following:

```
$ openstack role assignment list --names --role admin
+-------+-----------------------------------+-------+-----------------+------------+--------+-----------+
| Role  | User                              | Group | Project         | Domain     | System | Inherited |
+-------+-----------------------------------+-------+-----------------+------------+--------+-----------+
| admin | heat-cfn@Default                  |       | service@Default |            |        | False     |
| admin | placement@Default                 |       | service@Default |            |        | False     |
| admin | neutron@Default                   |       | service@Default |            |        | False     |
| admin | zaqar@Default                     |       | service@Default |            |        | False     |
| admin | swift@Default                     |       | service@Default |            |        | False     |
| admin | admin@Default                     |       | admin@Default   |            |        | False     |
| admin | zaqar-websocket@Default           |       | service@Default |            |        | False     |
| admin | heat@Default                      |       | service@Default |            |        | False     |
| admin | ironic-inspector@Default          |       | service@Default |            |        | False     |
| admin | nova@Default                      |       | service@Default |            |        | False     |
| admin | ironic@Default                    |       | service@Default |            |        | False     |
| admin | glance@Default                    |       | service@Default |            |        | False     |
| admin | mistral@Default                   |       | service@Default |            |        | False     |
| admin | heat_stack_domain_admin@heat_stack |       |                 | heat_stack |        | False
|
| admin | admin@Default                     |       |                 |            | all    | False     |
+-------+-----------------------------------+-------+-----------------+------------+--------+-----------+
```

**NOTE**

You can use the **-f {csv,json,table,value,yaml}** parameter to export these results.

## 11.8. AUDITING API ACCESS

You can audit the API calls a given role can access. Repeating this process for each role will result in a comprehensive report on the accessible APIs for each role.

### Prerequisites

- An authentication file to source as a user in the target role.
- An access token in JSON format.
- A policy file for each service's API you wish to audit.

### Procedure

1. Start by sourcing an authentication file of a user in the desired role.
2. Capture a Keystone generated token and save it to a file. You can do this by running any openstack-cli command and using the --debug option, which prints the provided token to stdout. You can copy this token and save it to an access file. Use the following command to do

this as a single step:

```
openstack token issue --debug 2>&1 | egrep ^'{\"token\":' > access.file.json
```

3. Create a policy file. This can be done on an overcloud node that hosts the containerized service of interest. The following example creates a policy file for the cinder service:

```
ssh tripleo-admin@CONTROLLER-1 sudo podman exec cinder_api \
oslopolicy-policy-generator \
--config-file /etc/cinder/cinder.conf \
--namespace cinder > cinderpolicy.json
```

4. Using these files, you can now audit the role in question for access to cinder’s APIs:

```
oslopolicy-checker --policy cinderpolicy.json --access access.file.json
```

# CHAPTER 12. NETWORK TIME PROTOCOL

You need to ensure that systems within your Red Hat OpenStack Platform cluster have accurate and consistent timestamps between systems.

Red Hat OpenStack Platform on Red Hat Enterprise Linux 8 supports Chrony for time management. For more information, see Using the Chrony suite to configure NTP .

## 12.1. WHY CONSISTENT TIME IS IMPORTANT

Consistent time throughout your organization is important for both operational and security needs:

**Identifying a security event**

Consistent timekeeping helps you correlate timestamps for events on affected systems so that you can understand the sequence of events.

**Authentication and security systems**

Security systems can be sensitive to time skew, for example:

- A kerberos-based authentication system might refuse to authenticate clients that are affected by seconds of clock skew.
- Transport layer security (TLS) certificates depend on a valid source of time. A client to server TLS connection fails if the difference between client and server system times exceeds the *Valid From* date range.

**Red Hat OpenStack Platform services**

Some core OpenStack services are especially dependent on accurate timekeeping, including High Availability (HA) and Ceph.

## 12.2. NTP DESIGN

Network time protocol (NTP) is organized in a hierarchical design. Each layer is called a stratum. At the top of the hierarchy are stratum 0 devices such as atomic clocks. In the NTP hierarchy, stratum 0 devices provide reference for publicly available stratum 1 and stratum 2 NTP time servers.

Do not connect your data center clients directly to publicly available NTP stratum 1 or 2 servers. The number of direct connections would put unnecessary strain on the public NTP resources. Instead, allocate a dedicated time server in your data center, and connect the clients to that dedicated server.

Configure instances to receive time from your dedicated time servers, not the host on which they reside.

**NOTE**

Service containers running within the Red Hat OpenStack Platform environment still receive time from the host on which they reside.

# CHAPTER 13. HARDENING INFRASTRUCTURE AND VIRTUALIZATION

You can harden your physical and virtual environment to better protect against internal and external threats.

## 13.1. HARWARE FOR RED HAT OPENSTACK PLATFORM

When you add hardware for use in your cloud environment, ensure that it is supported that hardware virtualization is supported. Disable hardware features that you do not use.

**Procedure**

1. Check Certified hardware for Red Hat OpenStack to ensure your hardware is supported.
2. Check that hardware virtualization is available and enabled:

   ```
   cat /proc/cpuinfo | egrep "vmx|svm"
   ```

3. Ensure that all firmware is up-to-date on your hardware platform. See hardware vendor documentation for details.

## 13.2. SOFTWARE UPDATES IN A CLOUD ENVIRONMENT

Keep Red Hat OpenStack Platform (RHOSP) updated for security, performance and supportability.

- If there are kernel updates included when you update, you must reboot the physical system or instance that you updated.
- Update OpenStack Image (glance) images to ensure that newly created instances have the latest updates.
- If you selectively update packages on RHOSP, ensure that all security updates are included. For more information about the latest vulnerabilities and security updates, see:
  - RHSA-announce
  - Errata notifications
  - Security Advisories

## 13.3. LIMITING HARDWARE AND SOFTWARE FEATURES

Enable only the hardware and software features that you use, so that less code is exposed to the possibility of attack. There are some features that you should only enable in trusted environments.

**PCI passthrough**

PCI passthrough allows an instance to have direct access to a PCI device on a node. An instance with PCI device access may allow a malicious actor to make modifications to the firmware. Additionally, some PCI devices have direct memory access (DMA). When you give an instance control over a device with DMA, it can gain arbitrary physical memory access.

You must enable PCI passthrough for specific use cases, such as Network Functions Virtualization (NFV). Do not enable PCI passthrough unless it is necessary for your deployment.

**Kernel same-page merging**

Kernel same-page merging (KSM) is a feature that reduces the use of memory through the deduplication and sharing of memory pages. When two or more virtual machines have identical pages in memory, those pages can be shared allowing for higher density. Memory deduplication strategies are vulnerable to side-channel attacks and should only be used in trusted environments. In Red Hat OpenStack Platform, KSM is disabled by default.

## 13.4. SELINUX ON RED HAT OPENSTACK PLATFORM

Security-Enhanced Linux (SELinux) is an implementation of mandatory access control (MAC). MAC limits the impact of an attack by restricting what a process or application is permitted to do on a system. For more information on SELinux, see What is SELinux?.

SELinux policies have been pre-configured for Red Hat OpenStack Platform (RHOSP) services. On RHOSP, SELinux is configured to run each QEMU process under a separate security context. In RHOSP, SELinux policies help protect hypervisor hosts and instances against the following threats:

**Hypervisor threats**

A compromised application running within an instance attacks the hypervisor to access underlying resources. If an instance is able to access the hypervisor OS, physical devices and other applications can become targets. This threat represents considerable risk. A compromise on a hypervisor can also compromise firmware, other instances, and network resources.

**Instance threats**

A compromised application running within an instance attacks the hypervisor to access or control another instance and its resources, or instance file images. The administrative strategies for protecting real networks do not apply directly to virtual environments. Because every instance is a process labeled by SELinux, there is a security boundary around each instance, enforced by the Linux kernel.

In RHOSP, instance image files on disk are labeled with SELinux data type **svirt_image_t**. When the instance is powered on, SELinux appends a random numerical identifier to the image. A random numerical identifier can prevent a compromised OpenStack instance from gaining unauthorized access to other containers. SELinux is capable of assigning up to 524,288 numeric identifiers on each hypervisor node.

## 13.5. INVESTIGATING CONTAINERIZED SERVICES

The OpenStack services that come with Red Hat OpenStack Platform run within containers. Containerization allows for the development and upgrade of services without dependency related conflicts. When a service runs within a container, potential vulnerabilities to that service are also contained.

You can get information about the service that are running in your environment by using the following steps:

**Procedure**

- Use `podman inspect to get information, such as bind mounted host directories:
  Example:

```
$ sudo podman inspect <container_name> | less
```

Replace <container_name> with the name of your container. For example, **nova compute**.

- Check the logs for the service located in **/var/log/containers**:
  Example:

  ```
  sudo less /var/log/containers/nova/nova-compute.log
  ```

- Run an interactive CLI session within the container:
  Example:

  ```
  podman exec -it nova_compute /bin/bash
  ```

NOTE

You can make changes to the service for testing purposes directly within the container. All changes are lost when the container is restarted.

## 13.6. MAKING TEMPORARY CHANGES TO CONTAINERIZED SERVICES

You can make changes to containerized services that persist when the container is restarted, but that do not affect the permanent configuration of your Red Hat OpenStack Platform (RHOSP) cluster. This is useful for testing configuration changes, or enabling debug-level logs when troubleshooting. You can revert changes manually. Alternatively, running a redeploy on your RHOSP cluster resets all parameters to their permanent configurations.

Use configuration files that are located in **/var/lib/config-data/puppet-generated/[service]** to make temporary changes to a service. The following example enables debugging on the nova service:

### Procedure

1. Edit the **nova.conf** configuration file that is bind mounted to the **nova_compute** container. Set the value of the **debug** parameter to **True**:

   ```
   $ sudo sed -i 's/^debug=.*/debug=True' \
   /var/lib/config-data/puppet-generated/nova/etc/nova/nova.conf
   ```

   WARNING

   Configuration files for OpenStack files are **ini** files with multiple sections, such as **[DEFAULT]** and **[database]**. Parameters that are unique to each section might not be unique across the entire file. Use **sed** with caution. You can check to see if a **parameter** appears more than once in a configuration file by running **egrep -v "^$|^#" [configuration_file] | grep [parameter]**.

2. Restart the nova container:

```
sudo podman restart nova_compute
```

## 13.7. MAKING PERMANENT CHANGES TO CONTAINERIZED SERVICES

You can make permanent changes to containerized services in Red Hat OpenStack Platform (RHOSP) services with heat. Use an existing template that you used when you first deployed RHOSP, or create a new template to add to your deployment script. In the following example, the private key size for libvirt is increased to **4096**.

### Procedure

1. Create a new **yaml** template called **libvirt-keysize.yaml**, and use the **LibvirtCertificateKeySize** parameter to increase the default value from **2048** to **4096**.

   ```
   cat > /home/stack/templates/libvirt-keysize.yaml
   parameter_defaults:
           LibvirtCertificateKeySize: 4096
   EOF
   ```

2. Add the **libvirt-keysize.yaml** configuration file to your deployment script:

   ```
   openstack overcloud deploy --templates \
   ...
   -e /home/stack/templates/libvirt-keysize.yaml
   ...
   ```

3. Rerun the deployment script:

   ```
   ./deploy.sh
   ```

## 13.8. FIRMWARE UPDATES

Physical servers use complex firmware to enable and operate server hardware and lights-out management cards, which can have their own security vulnerabilities, potentially allowing system access and interruption. To address these, hardware vendors will issue firmware updates, which are installed separately from operating system updates. You will need an operational security process that retrieves, tests, and implements these updates on a regular schedule, noting that firmware updates often require a reboot of physical hosts to become effective.

## 13.9. USE SSH BANNER TEXT

You can set a banner that displays a console message to all users that connect over SSH. You can add banner text to **/etc/issue** using the following parameters in an environment file. Consider customizing this sample text to suit your requirements.

```
resource_registry:
  OS::TripleO::Services::Sshd:
    /usr/share/openstack-tripleo-heat-templates/deployment/sshd/sshd-baremetal-puppet.yaml

parameter_defaults:
  BannerText: |
   ******************************************************************
```

```
 * This system is for the use of authorized users only. Usage of  *
 * this system may be monitored and recorded by system personnel. *
 * Anyone using this system expressly consents to such monitoring *
 * and is advised that if such monitoring reveals possible        *
 * evidence of criminal activity, system personnel may provide    *
 * the evidence from such monitoring to law enforcement officials.*
 ******************************************************************
```

To apply this change to your deployment, save the settings as a file called **ssh_banner.yaml**, and then pass it to the **overcloud deploy** command as follows. The **<full environment>** indicates that you must still include all of your original deployment parameters. For example:

```
  openstack overcloud deploy --templates \
    -e <full environment> -e  ssh_banner.yaml
```

## 13.10. AUDIT FOR SYSTEM EVENTS

Maintaining a record of all audit events helps you establish a system baseline, perform troubleshooting, or analyze the sequence of events that led to a certain outcome. The audit system is capable of logging many types of events, such as changes to the system time, changes to Mandatory/Discretionary Access Control, and creating/deleting users or groups.

Rules can be created using an environment file, which are then injected by director into **/etc/audit/audit.rules**. For example:

```
  resource_registry:
    OS::TripleO::Services::AuditD: /usr/share/openstack-tripleo-heat-
templates/deployment/auditd/auditd-baremetal-puppet.yaml
  parameter_defaults:
    AuditdRules:
      'Record Events that Modify User/Group Information':
        content: '-w /etc/group -p wa -k audit_rules_usergroup_modification'
        order  : 1
      'Collects System Administrator Actions':
        content: '-w /etc/sudoers -p wa -k actions'
        order  : 2
      'Record Events that Modify the Systems Mandatory Access Controls':
        content: '-w /etc/selinux/ -p wa -k MAC-policy'
        order  : 3
```

## 13.11. MANAGE FIREWALL RULES

Firewall rules are automatically applied on overcloud nodes during deployment, and are intended to only expose the ports required to get OpenStack working. You can specify additional firewall rules as needed. For example, to add rules for a Zabbix monitoring system:

```
parameter_defaults:
  ControllerExtraConfig:
    ExtraFirewallRules:
      '301 allow zabbix':
      dport: 10050
      proto: tcp
      source: 10.0.0.8
```

NOTE

When you do not set the **action** parameter, the result is **accept**. You can only set the **action** parameter to **drop**, **insert**, or **append**.

You can also add rules that restrict access. The number used during rule definition will determine the rule's precedence. For example, RabbitMQ's rule number is **109** by default. If you want to restrain it, you switch it to use a lower value:

```
parameter_defaults:
  ControllerParameters
    ExtraFirewallRules:
      '098 allow rabbit from internalapi network':
        dport: [4369,5672,25672]
        proto: tcp
        source: 10.0.0.0/24
      '099 drop other rabbit access:
        dport: [4369,5672,25672]
        proto: tcp
        action: drop
```

In this example, **098** and **099** are arbitrarily chosen numbers that are lower than RabbitMQ's rule number **109**. To determine a rule's number, you can inspect the iptables rule on the appropriate node; for RabbitMQ, you would check the controller:

```
iptables-save
[...]
-A INPUT -p tcp -m multiport --dports 4369,5672,25672 -m comment --comment "109 rabbitmq" -m
state --state NEW -j ACCEPT
```

Alternatively, you can extract the port requirements from the puppet definition. For example, RabbitMQ's rules are stored in **puppet/services/rabbitmq.yaml**:

```
    ExtraFirewallRules:
      '109 rabbitmq':
        dport:
          - 4369
          - 5672
          - 25672
```

The following parameters can be set for a rule:

- **dport**: The destination port associated to the rule.
- **sport**: The source port associated to the rule.
- **proto**: The protocol associated to the rule. Defaults to *tcp*
- **action**: The action policy associated to the rule. Defaults to *INSERT* and sets the jump to *ACCEPTS*.
- **state**: Array of states associated to the rule. Default to [ *NEW*]
- **source**: The source IP address associated to the rule.

- **interface**: The network interface associated to the rule.
- **chain**: The chain associated to the rule. Default to *INPUT*
- **destination**: The destination cidr associated to the rule.

## 13.12. INTRUSION DETECTION WITH AIDE

AIDE (Advanced Intrusion Detection Environment) is a file and directory integrity checker. It is used to detect incidents of unauthorized file tampering or changes. For example, AIDE can alert you if system password files are changed.

AIDE works by analyzing system files and then compiling an integrity database of file hashes. The database then serves as a comparison point to verify the integrity of the files and directories and detect changes.

The director includes the AIDE service, allowing you to add entries into an AIDE configuration, which is then used by the AIDE service to create an integrity database. For example:

```
resource_registry:
  OS::TripleO::Services::Aide:
    /usr/share/openstack-tripleo-heat-templates/deployment/aide/aide-baremetal-ansible.yaml

parameter_defaults:
  AideRules:
    'TripleORules':
      content: 'TripleORules = p+sha256'
      order: 1
    'etc':
      content: '/etc/ TripleORules'
      order: 2
    'boot':
      content: '/boot/ TripleORules'
      order: 3
    'sbin':
      content: '/sbin/ TripleORules'
      order: 4
    'var':
      content: '/var/ TripleORules'
      order: 5
    'not var/log':
      content: '!/var/log.*'
      order: 6
    'not var/spool':
      content: '!/var/spool.*'
      order: 7
    'not nova instances':
      content: '!/var/lib/nova/instances.*'
      order: 8
```

### NOTE

The above example is not actively maintained or benchmarked, so you should select the AIDE values that suit your requirements.

1. An alias named **TripleORules** is declared to avoid having to repeatedly out the same attributes each time.

2. The alias receives the attributes of **p+sha256**. In AIDE terms, this reads as the following instruction: monitor all file permissions **p** with an integrity checksum of **sha256**.

For a complete list of attributes available for AIDE's config files, see the AIDE MAN page at https://aide.github.io/.

Complete the following to apply changes to your deployment:

1. Save the settings as a file called **aide.yaml** in the **/home/stack/templates/** directory.

2. Edit the **aide.yaml** environment file to have the parameters and values suitable for your environment.

3. Include the **/home/stack/templates/aide.yaml** environment file in the **openstack overcloud deploy** command, along with all other necessary heat templates and environment files specific to your environment:

```
openstack overcloud deploy --templates
...
-e /home/stack/templates/aide.yaml
```

### 13.12.1. Using complex AIDE rules

Complex rules can be created using the format described previously. For example:

```
MyAlias = p+i+n+u+g+s+b+m+c+sha512
```

The above would translate as the following instruction: monitor permissions, inodes, number of links, user, group, size, block count, mtime, ctime, using sha256 for checksum generation.

Note, the alias should always have an order position of **1**, which means that it is positioned at the top of the AIDE rules and is applied recursively to all values below.

Following after the alias are the directories to monitor. Note that regular expressions can be used. For example we set monitoring for the **var** directory, but overwrite with a not clause using **!** with **'!/var/log.*'** and **'!/var/spool.*'**.

### 13.12.2. Additional AIDE values

The following AIDE values are also available:

**AideConfPath**: The full POSIX path to the aide configuration file, this defaults to **/etc/aide.conf**. If no requirement is in place to change the file location, it is recommended to stick with the default path.

**AideDBPath**: The full POSIX path to the AIDE integrity database. This value is configurable to allow operators to declare their own full path, as often AIDE database files are stored off node perhaps on a read only file mount.

**AideDBTempPath**: The full POSIX path to the AIDE integrity temporary database. This temporary files is created when AIDE initializes a new database.

**AideHour**: This value is to set the hour attribute as part of AIDE cron configuration.

**AideMinute**: This value is to set the minute attribute as part of AIDE cron configuration.

**AideCronUser**: This value is to set the linux user as part of AIDE cron configuration.

**AideEmail**: This value sets the email address that receives AIDE reports each time a cron run is made.

**AideMuaPath**: This value sets the path to the Mail User Agent that is used to send AIDE reports to the email address set within **AideEmail**.

### 13.12.3. Cron configuration for AIDE

The AIDE director service allows you to configure a cron job. By default, it will send reports to **/var/log/audit/**; if you want to use email alerts, then enable the **AideEmail** parameter to send the alerts to the configured email address. Note that a reliance on email for critical alerts can be vulnerable to system outages and unintentional message filtering.

### 13.12.4. Considering the effect of system upgrades

When an upgrade is performed, the AIDE service will automatically regenerate a new integrity database to ensure all upgraded files are correctly recomputed to possess an updated checksum.

If **openstack overcloud deploy** is called as a subsequent run to an initial deployment, and the AIDE configuration rules are changed, the director AIDE service will rebuild the database to ensure the new config attributes are encapsulated in the integrity database.

## 13.13. REVIEW SECURETTY

SecureTTY allows you to disable root access for any console device (tty). This behavior is managed by entries in the **/etc/securetty** file. For example:

```
resource_registry:
  OS::TripleO::Services::Securetty: ../puppet/services/securetty.yaml

parameter_defaults:
  TtyValues:
    - console
    - tty1
    - tty2
    - tty3
    - tty4
    - tty5
    - tty6
```

## 13.14. CADF AUDITING FOR IDENTITY SERVICE

A thorough auditing process can help you review the ongoing security posture of your OpenStack deployment. This is especially important for keystone, due to its role in the security model.

Red Hat OpenStack Platform has adopted Cloud Auditing Data Federation (CADF) as the data format for audit events, with the keystone service generating CADF events for Identity and Token operations. You can enable CADF auditing for keystone using **KeystoneNotificationFormat**:

```
parameter_defaults:
  KeystoneNotificationFormat: cadf
```

-

## 13.15. REVIEW THE LOGIN.DEFS VALUES

To enforce password requirements for new system users (non-keystone), director can add entries to **/etc/login.defs** by following these example parameters:

```
resource_registry:
  OS::TripleO::Services::LoginDefs: ../puppet/services/login-defs.yaml

parameter_defaults:
  PasswordMaxDays: 60
  PasswordMinDays: 1
  PasswordMinLen: 5
  PasswordWarnAge: 7
  FailDelay: 4
```

# CHAPTER 14. HARDENING THE DASHBOARD SERVICE

The Dashboard service (horizon) gives users a self-service portal for provisioning their own resources within the limits set by administrators. Manage the security of the Dashboard service with the same sensitivity as the OpenStack APIs.

## 14.1. DEBUGGING THE DASHBOARD SERVICE

The default value for the **DEBUG** parameter is **False**. Keep the default value in your production environment. Change this setting only during investigation. When you change the value of the **DEBUG** parameter to **True**, Django can output stack straces to browser users that contain sensitive web server state information.

When the value of the **DEBUG** parameter is **True**, the **ALLOWED_HOSTS** settings are also disabled. For more information on configuring **ALLOWED_HOSTS**, see Configure ALLOWED_HOSTS.

## 14.2. SELECTING A DOMAIN NAME

It is a best practice to deploy the Dashboard service (horizon) to a second level domain, as opposed to a shared domain on any level. Examples of each are provided below:

- Second level domain: **https://example.com**
- Shared subdomain: **https://example.public-url.com**

Deploying the Dashboard service to a dedicated second level domain isolates cookies and security tokens from other domains, based on browsers' **same-origin** policy. When deployed on a subdomain, the security of the Dashboard service is equivalent to the least secure application deployed on the same second-level domain.

You can further mitigate this risk by avoiding a cookie-backed session store, and configuring HTTP Strict Transport Security (HSTS) (described in this guide).

NOTE

Deploying the Dashboard service on a bare domain, like **https://example/**, is unsupported.

## 14.3. CONFIGURE ALLOWED_HOSTS

Horizon is built on the python Django web framework, which requires protection against security threats associated with misleading HTTP Host headers. To apply this protection, configure the **ALLOWED_HOSTS** setting to use the FQDN that is served by the OpenStack dashboard.

When you configure the **ALLOWED_HOSTS** setting, any HTTP request with a Host header that does not match the values in this list is denied, and an error is raised.

### Procedure

1. Under **parameter_defaults** in your templates, set the value of the **HorizonAllowedHosts** parameter:

```
parameter_defaults:
  HorizonAllowedHosts: <value>
```

Replace **<value>** with the FQDN that is served by the OpenStack dashboard.

2. Deploy the overcloud with the modified template, and all other templates required for your environment.

## 14.4. CROSS SITE SCRIPTING (XSS)

The OpenStack Dashboard accepts the entire Unicode character set in most fields. Malicious actors can attempt to use this extensibility to test for cross-site scripting (XSS) vulnerabilities. The OpenStack Dashboard service (horizon) has tools that harden against XSS vulnerabilites. It is important to ensure the correct use of these tools in custom dashboards. When you perform an audit against custom dashboards, pay attention to the following:

- The **mark_safe** function.
- **is_safe** - when used with custom template tags.
- The **safe** template tag.
- Anywhere auto escape is turned off, and any JavaScript which might evaluate improperly escaped data.

## 14.5. CROSS SITE REQUEST FORGERY (CSRF)

Dashboards that use multiple JavaScript instances should be audited for vulnerabilities such as inappropriate use of the **@csrf_exempt** decorator. Evaluate any dashboard that does not follow recommended security settings before lowering CORS (Cross Origin Resource Sharing) restrictions. Configure your web server to send a restrictive CORS header with each response. Allow only the dashboard domain and protocol, for example:**Access-Control-Allow-Origin: https://example.com/**. You should never allow the wild card origin.

## 14.6. ALLOW IFRAME EMBEDDING

The **DISALLOW_IFRAME_EMBED** setting disallows Dashboard from being embedded within an iframe. Legacy browsers can still be vulnerable to Cross-Frame Scripting (XFS) vulnerabilities, so this option adds extra security hardening for deployments that do not require iframes. The setting is set to **True** by default, however it can be disabled using an environment file, if needed.

**Procedure**

- You can allow iframe embedding using the following parameter:

```
parameter_defaults:
  ControllerExtraConfig:
    horizon::disallow_iframe_embed: false
```

NOTE

These settings should only be set to **False** once the potential security impacts are fully understood.

## 14.7. USING HTTPS ENCRYPTION FOR DASHBOARD TRAFFIC

It is recommended you use HTTPS to encrypt Dashboard traffic. You can do this by configuring it to use a valid, trusted certificate from a recognized certificate authority (CA). Private organization-issued certificates are only appropriate when the root of trust is pre-installed in all user browsers.

Configure HTTP requests to the dashboard domain to redirect to the fully qualified HTTPS URL.

See Chapter 7, *Enabling SSL/TLS on overcloud public endpoints* . for more information.

## 14.8. HTTP STRICT TRANSPORT SECURITY (HSTS)

HTTP Strict Transport Security (HSTS) prevents browsers from making subsequent insecure connections after they have initially made a secure connection. If you have deployed your HTTP services on a public or an untrusted zone, HSTS is especially important.

For director-based deployments, this setting is enabled by default in the **/usr/share/openstack-tripleo-heat-templates/deployment/horizon/horizon-container-puppet.yaml** file:

```
horizon::enable_secure_proxy_ssl_header: true
```

### Verification

After the overcloud is deployed, check the **local_settings** file for Red Hat OpenStack Dashboard (horizon) for verification.

1. Use **ssh** to connect to a controller:

   ```
   $ ssh tripleo-admin@controller-0
   ```

2. Check that the **SECURE_PROXY_SSL_HEADER** parameter has a value of **('HTTP_X_FORWARDED_PROTO', 'https')**:

   ```
   sudo egrep ^SECURE_PROXY_SSL_HEADER /var/lib/config-data/puppet-generated/horizon/etc/openstack-dashboard/local_settings
   SECURE_PROXY_SSL_HEADER = ('HTTP_X_FORWARDED_PROTO', 'https')
   ```

## 14.9. FRONT-END CACHING

It is not recommended to use front-end caching tools with the Dashboard, as it renders dynamic content resulting directly from OpenStack API requests. As a result, front-end caching layers such as **varnish** can prevent the correct content from being displayed. The Dashboard uses Django, which serves static media directly served from the web service and already benefits from web host caching.

## 14.10. SESSION BACKEND

For director-based deployments, the default session backend for horizon is **django.contrib.sessions.backends.cache**, which is combined with memcached. This approach is preferred to local-memory cache for performance reasons, is safer for highly-available and load balanced installs, and has the ability to share cache over multiple servers, while still treating it as a single cache.

You can review these settings in director's **horizon.yaml** file:

```
horizon::cache_backend: django.core.cache.backends.memcached.MemcachedCache
horizon::django_session_engine: 'django.contrib.sessions.backends.cache'
```

## 14.11. REVIEWING THE SECRET KEY

The Dashboard depends on a shared **SECRET_KEY** setting for some security functions. The secret key should be a randomly generated string at least 64 characters long, which must be shared across all active dashboard instances. Compromise of this key might allow a remote attacker to execute arbitrary code. Rotating this key invalidates existing user sessions and caching. Do not commit this key to public repositories.

For director deployments, this setting is managed as the **HorizonSecret** value.

## 14.12. CONFIGURING SESSION COOKIES

The Dashboard session cookies can be open to interaction by browser technologies, such as JavaScript. For director deployments with TLS everywhere, you can harden this behavior using the **HorizonSecureCookies** setting.

NOTE

Never configure CSRF or session cookies to use a wildcard domain with a leading dot.

## 14.13. STATIC MEDIA

The dashboard's static media should be deployed to a subdomain of the dashboard domain and served by the web server. The use of an external content delivery network (CDN) is also acceptable. This subdomain should not set cookies or serve user-provided content. The media should also be served with HTTPS.

Dashboard's default configuration uses **django_compressor** to compress and minify CSS and JavaScript content before serving it. This process should be statically done before deploying the dashboard, rather than using the default in-request dynamic compression and copying the resulting files along with deployed code or to the CDN server. Compression should be done in a non-production build environment. If this is not practical, consider disabling resource compression entirely. Online compression dependencies (less, Node.js) should not be installed on production machines.

## 14.14. VALIDATING PASSWORD COMPLEXITY

The OpenStack Dashboard (horizon) can use a password validation check to enforce password complexity.

### Procedure

1. Specify a regular expression for password validation, as well as help text to be displayed for failed tests. The following example requires users to create a password of between 8 to 18 characters in length:

```
parameter_defaults:
  HorizonPasswordValidator: '^.{8,18}$'
  HorizonPasswordValidatorHelp: 'Password must be between 8 and 18 characters.'
```

1. Apply this change to your deployment. Save the settings as a file called **horizon_password.yaml**, and then pass it to the **overcloud deploy** command as follows. The **<full environment>** indicates that you must still include all of your original deployment parameters. For example:

```
openstack overcloud deploy --templates \
  -e <full environment> -e  horizon_password.yaml
```

## 14.15. ENFORCE THE ADMINISTRATOR PASSWORD CHECK

The following setting is set to **True** by default, however it can be disabled using an environment file, if needed.

NOTE

These settings should only be set to **False** once the potential security impacts are fully understood.

### Procedure

The **ENFORCE_PASSWORD_CHECK** setting in Dashboard's **local_settings.py** file displays an *Admin Password* field on the *Change Password* form, which helps verify that an administrator is initiating the password change.

- You can disable **ENFORCE_PASSWORD_CHECK** using an environment file:

```
parameter_defaults:
  ControllerExtraConfig:
    horizon::enforce_password_check: false
```

## 14.16. DISABLE PASSWORD REVEAL

The **disable_password_reveal** parameter is set to **True** by default, however it can be disabled using an environment file, if needed. The password reveal button allows a user at the Dashboard to view the password they are about to enter.

### Procedure

- Under the **ControllerExtraConfig** parameter, include **horizon::disable_password_reveal: false**. Save this to a heat environment file and include it with your deployment command.

### Example

```
parameter_defaults:
  ControllerExtraConfig:
    horizon::disable_password_reveal: false
```

NOTE

These settings should only be set to **False** once the potential security impacts are fully understood.

## 14.17. DISPLAYING A LOGON BANNER FOR THE DASHBOARD

Regulated industries such as HIPAA, PCI-DSS, and the US Government require you to display a user logon banner. The Red Hat OpenStack Platform (RHOSP) dashboard (horizon) uses a default theme (RCUE), which is stored inside the horizon container.

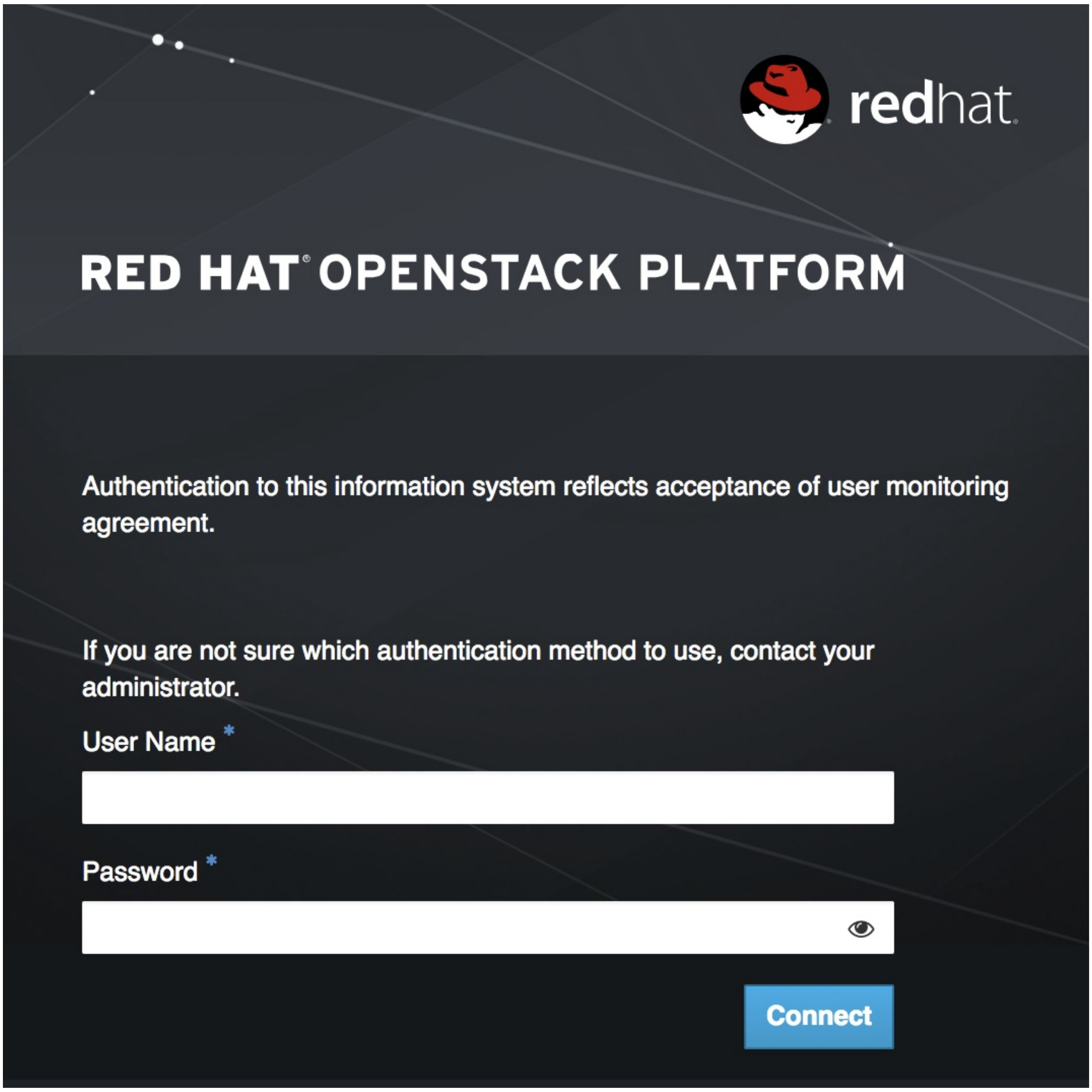
Within the custom Dashboard container, you can create a logon banner by manually editing the **/usr/share/openstack-dashboard/openstack_dashboard/themes/rcue/templates/auth/login.html** file:

### Procedure

1. Enter the required logon banner just before the **{% include 'auth/_login.html' %}** section. HTML tags are allowed:

```
<snip>
<div class="container">
  <div class="row-fluid">
    <div class="span12">
      <div id="brand">
        <img src="../../static/themes/rcue/images/RHOSP-Login-Logo.svg">
      </div><!--/#brand-->
    </div><!--/.span*-->

    <!-- Start of Logon Banner -->
    <p>Authentication to this information system reflects acceptance of user monitoring
agreement.</p>
    <!-- End of Logon Banner -->

    {% include 'auth/_login.html' %}
  </div><!--/.row-fluid→
</div><!--/.container-->

{% block js %}
  {% include "horizon/_scripts.html" %}
{% endblock %}

  </body>
</html>
```

The above example produces a dashboard similar to the following:


RED HAT® OPENSTACK PLATFORM

Authentication to this information system reflects acceptance of user monitoring agreement.

If you are not sure which authentication method to use, contact your administrator.

User Name *

Password *

Additional resources

- Customizing the dashboard

## 14.18. LIMITING THE SIZE OF FILE UPLOADS

You can optionally configure the dashboard to limit the size of file uploads; this setting might be a requirement for various security hardening policies.

**LimitRequestBody** - This value (in bytes) limits the maximum size of a file that you can transfer using the Dashboard, such as images and other large files.

IMPORTANT

This setting has not been formally tested by Red Hat. It is recommended that you thoroughly test the effect of this setting before deploying it to your production environment.

NOTE

File uploads will fail if the value is too small.

For example, this setting limits each file upload to a maximum size of 10 GB (**10737418240**). You will need to adjust this value to suit your deployment.

- **/var/lib/config-data/puppet-generated/horizon/etc/httpd/conf/httpd.conf**

```
<Directory />
  LimitRequestBody 10737418240
</Directory>
```

- **/var/lib/config-data/puppet-generated/horizon/etc/httpd/conf.d/10-horizon_vhost.conf**

```
<Directory "/var/www">
  LimitRequestBody 10737418240
</Directory>
```

- **/var/lib/config-data/puppet-generated/horizon/etc/httpd/conf.d/15-horizon_ssl_vhost.conf**

```
<Directory "/var/www">
  LimitRequestBody 10737418240
</Directory>
```

NOTE

These configuration files are managed by Puppet, so any unmanaged changes are overwritten whenever you run the **openstack overcloud deploy** process.

# CHAPTER 15. HARDENING THE NETWORKING SERVICE

The Networking service (neutron) is the software-defined networking (SDN) component of Red Hat OpenStack Platform (RHOSP). The RHOSP Networking service manages internal and external traffic to and from virtual machine instances and provides core services such as routing, segmentation, DHCP, and metadata. It provides the API for virtual networking capabilities and management of switches, routers, ports, and firewalls.

For more information about the Red Hat OpenStack Platfrom Networking service, see the Networking Guide.

This section discusses OpenStack Networking configuration good practices as they apply to project network security within your OpenStack deployment.

## 15.1. RESTRICT BIND ADDRESS OF THE API SERVER: NEUTRON-SERVER

To restrict the interface or IP address on which the OpenStack Networking API service binds a network socket for incoming client connections, specify the **bind_host** and **bind_port** in the **/var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf** file:

```
# Address to bind the API server
bind_host = IP ADDRESS OF SERVER

# Port the bind the API server to
bind_port = 9696
```

## 15.2. PROJECT NETWORK SERVICES WORKFLOW

OpenStack Networking provides users self-service configuration of network resources. It is important that cloud architects and operators evaluate their design use cases in providing users the ability to create, update, and destroy available network resources.

## 15.3. NETWORKING RESOURCE POLICY ENGINE

A policy engine and its configuration file (**policy.json**) within OpenStack Networking provides a method to provide finer grained authorization of users on project networking methods and objects. The OpenStack Networking policy definitions affect network availability, network security and overall OpenStack security. Cloud architects and operators should carefully evaluate their policy towards user and project access to administration of network resources.

**NOTE**

It is important to review the default networking resource policy, as this policy can be modified to suit your security posture.

If your deployment of OpenStack provides multiple external access points into different security zones it is important that you limit the project’s ability to attach multiple vNICs to multiple external access points – this would bridge these security zones and could lead to unforeseen security compromise. You can help mitigate this risk by using the host aggregates functionality provided by Compute, or by splitting the project instances into multiple projects with different virtual network configurations. For more information on host aggregates, see Creating and managing host aggregates .

## 15.4. SECURITY GROUPS

A security group is a collection of security group rules. Security groups and their rules allow administrators and projects the ability to specify the type of traffic and direction (ingress/egress) that is allow ed to pass through a virtual interface port. When a virtual interface port is created in OpenStack Networking it is associated with a security group. Rules can be added to the default security group in order to change the behavior on a per-deployment basis.

When using the Compute API to modify security groups, the updated security group applies to all virtual interface ports on an instance. This is due to the Compute security group APIs being instance-based rather than port-based, as found in neutron.

## 15.5. MITIGATE ARP SPOOFING

OpenStack Networking has a built-in feature to help mitigate the threat of ARP spoofing for instances. This should not be disabled unless careful consideration is given to the resulting risks.

## 15.6. USE A SECURE PROTOCOL FOR AUTHENTICATION

In **/var/lib/config-data/puppet-generated/neutron/etc/neutron/neutron.conf** check that the value of **auth_uri** under the **[keystone_authtoken]** section is set to an Identity API endpoint that starts with `https:

# CHAPTER 16. HARDENING BLOCK STORAGE ON RED HAT OPENSTACK PLATFORM

OpenStack Block Storage (cinder) is a service that provides software (services and libraries) to self-service manage persistent block-level storage devices. This creates on-demand access to Block Storage resources for use with Compute (nova) instances. This creates software-defined storage through abstraction by virtualizing pools of block storage to a variety of back-end storage devices which can be either software implementations or traditional hardware storage products. The primary functions of this is to manage the creation, attachment, and detachment of the block devices. The consumer requires no knowledge of the type of back-end storage equipment or where it is located.

Compute instances store and retrieve block storage using industry-standard storage protocols such as iSCSI, ATA over Ethernet, or Fibre-Channel. These resources are managed and configured using OpenStack native standard HTTP RESTful API.

## 16.1. SET THE MAX SIZE FOR THE BODY OF A REQUEST

If the maximum body size per request is not defined, the attacker can craft an arbitrary OSAPI request of large size, causing the service to crash and finally resulting in a Denial Of Service attack. Assigning t he maximum value ensures that any malicious oversized request gets blocked ensuring continued availability of the service.

Review whether **max_request_body_size** under the **[oslo_middleware]** section in **cinder.conf** is set to **114688**.

## 16.2. ENABLE VOLUME ENCRYPTION

Unencrypted volume data makes volume-hosting platforms especially high-value targets for attackers, as it allows the attacker to read the data for many different VMs. In addition, the physical storage medium could be stolen, remounted, and accessed from a different machine. Encrypting volume data and volume backups can help mitgate these risks and provides defense-in-depth to volume-hosting platforms. Block Storage (cinder) is able to encrypt volume data before it is written to disk, so consider enabling volume encryption, and using Barbican for private key storage.

## 16.3. VOLUME WIPING

There are multiple ways to wipe a block storage device. The traditional approach is to set the **lvm_type** to thin, and then use the **volume_clear** parameter. Alternatively, if the volume encryption feature is used, then volume wiping is not necessary if the volume encryption key is deleted.

NOTE

Previously, **lvm_type=default** was used to signify a wipe. While this method still works, **lvm_type=default** is not recommended for setting secure delete.

The **volume_clear** parameter can accept either **zero** or **shred** as arguments. **zero** will write a single pass of zeroes to the device. The **shred** operation will write three passes of predetermined bit patterns.

# CHAPTER 17. HARDENING THE SHARED FILE SYSTEM (MANILA)

The Shared File Systems service (manila) provides a set of services for managing shared file systems in a multi-project cloud environment. With manila, you can create a shared file system and manage its properties, such as visibility, accessibility, and quotas.

For more information on manila, see the Storage Guide: https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/17.0/html-single/storage_guide/

## 17.1. SECURITY CONSIDERATIONS FOR MANILA

Manila is registered with keystone, allowing you to the locate the API using the **manila endpoints** command. For example:

```
$ manila endpoints
+-------------+-----------------------------------------+
| manila      | Value                                   |
+-------------+-----------------------------------------+
| adminURL    | http://172.18.198.55:8786/v1/20787a7b...|
| region      | RegionOne                               |
| publicURL   | http://172.18.198.55:8786/v1/20787a7b...|
| internalURL | http://172.18.198.55:8786/v1/20787a7b...|
| id          | 82cc5535aa444632b64585f138cb9b61        |
+-------------+-----------------------------------------+

+-------------+-----------------------------------------+
| manilav2    | Value                                   |
+-------------+-----------------------------------------+
| adminURL    | http://172.18.198.55:8786/v2/20787a7b...|
| region      | RegionOne                               |
| publicURL   | http://172.18.198.55:8786/v2/20787a7b...|
| internalURL | http://172.18.198.55:8786/v2/20787a7b...|
| id          | 2e8591bfcac4405fa7e5dc3fd61a2b85        |
+-------------+-----------------------------------------+
```

By default, the manila API service only listens on port **8786** with **tcp6**, which supports both IPv4 and IPv6.

Manila uses multiple configurations files; these are stored in **/var/lib/config-data/puppet-generated/manila/**:

```
api-paste.ini
manila.conf
policy.json
rootwrap.conf
rootwrap.d

./rootwrap.d:
share.filters
```

It is recommended that you configure manila to run under a non-root service account, and change file permissions so that only the system administrator can modify them. Manila expects that only administrators can write to configuration files, and services can only read them through their group membership in the **manila** group. Other users must not be able to read these files, as they contain service account passwords.

NOTE

Only the root user should own be able to write to the configuration for **manila-rootwrap** in **rootwrap.conf**, and the **manila-rootwrap** command filters for share nodes in **rootwrap.d/share.filters**.

## 17.2. NETWORK AND SECURITY MODELS FOR MANILA

A share driver in manila is a Python class that can be set for the back end to manage share operations, some of which are vendor-specific. The back end is an instance of the **manila-share** service. Manila has share drivers for many different storage systems, supporting both commercial vendors and open source solutions. Each share driver supports one or more back end modes: **share servers** and **no share servers**. An administrator selects a mode by specifying it in **manila.conf**, using **driver_handles_share_servers**.

A share server is a logical Network Attached Storage (NAS) server that exports shared file systems. Back-end storage systems today are sophisticated and can isolate data paths and network paths between different OpenStack projects.

A share server provisioned by a manila share driver would be created on an isolated network that belongs to the project user creating it. The **share servers** mode can be configured with either a flat network, or a segmented network, depending on the network provider.

It is possible to have separate drivers for different modes use the same hardware. Depending on the chosen mode, you might need to provide more configuration details through the configuration file.

## 17.3. SHARE BACKEND MODES

Each share driver supports at least one of the available driver modes:

- Share servers - **driver_handles_share_servers = True** - The share driver creates share servers and manages the share server life cycle.
- No share servers - **driver_handles_share_servers = False** - An administrator (rather than a share driver) manages the bare metal storage with a network interface, instead of relying on the presence of the share servers.

**No share servers mode** - In this mode, drivers will not set up share servers, and consequently will not need to set up any new network interfaces. It is assumed that storage controller being managed by the driver has all of the network interfaces it is going to need. Drivers create shares directly without previously creating a share server. To create shares using drivers operating in this mode, manila does not require users to create any private share networks either.

NOTE

In **no share servers mode**, manila will assume that the network interfaces through which any shares are exported are already reachable by all projects.

In the **no share servers** mode a share driver does not handle share server life cycle. An administrator is expected to handle the storage, networking, and other host-side configuration that might be necessary to provide project isolation. In this mode an administrator can set storage as a host which exports shares.

All projects within the OpenStack cloud share a common network pipe. Lack of isolation can impact security and quality of service. When using share drivers that do not handle share servers, cloud users cannot be sure that their shares cannot be accessed by untrusted users by a tree walk over the top directory of their file systems. In public clouds it is possible that all network bandwidth is used by one client, so an administrator should care for this not to happen. Network balancing can be done by any means, and not necessarily just with OpenStack tools.

**Share servers mode** - In this mode, a driver is able to create share servers and plug them to existing OpenStack networks. Manila determines if a new share server is required, and provides all the networking information necessary for the share drivers to create the requisite share server.

When creating shares in the driver mode that handles share servers, users must provide a share network that they expect their shares to be exported upon. Manila uses this network to create network ports for the share server on this network.

Users can configure **security services** in both **share servers** and **no share servers** back end modes. But with the **no share servers** back end mode, an administrator must set the required authentication services manually on the host. And in **share servers** mode manila can configure security services identified by the users on the share servers it spawns.

## 17.4. NETWORKING REQUIREMENTS FOR MANILA

Manila can integrate with different network types: **flat**, **GRE**, **VLAN**, **VXLAN**.

NOTE

Manila is only storing the network information in the database, with the real networks being supplied by the network provider. Manila supports using the OpenStack Networking service (neutron) and also "standalone" pre-configured networking.

In the **share servers** back end mode, a share driver creates and manages a share server for each share network. This mode can be divided in two variations:

- Flat network in **share servers** backend mode
- Segmented network in **share servers** backend mode

Users can use a network and subnet from the OpenStack Networking (neutron) service to create share networks. If the administrator decides to use the **StandAloneNetworkPlugin**, users need not provide any networking information since the administrator pre-configures this in the configuration file.

NOTE

Share servers spawned by some share drivers are Compute servers created with the Compute service. A few of these drivers do not support network plugins.

After a share network is created, manila retrieves network information determined by a network provider: network type, segmentation identifier (if the network uses segmentation) and the IP block in CIDR notation from which to allocate the network.

Users can create security services that specify security requirements such as AD or LDAP domains or a Kerberos realm. Manila assumes that any hosts referred to in security service are reachable from a subnet where a share server is created, which limits the number of cases where this mode could be used.

NOTE

Some share drivers might not support all types of segmentation, for more details see the specification for the driver you are using.

## 17.5. SECURITY SERVICES WITH MANILA

Manila can restrict access to file shares by integrating with network authentication protocols. Each project can have its own authentication domain that functions separately from the cloud's keystone authentication domain. This project domain can be used to provide authorization (AuthZ) services to applications that run within the OpenStack cloud, including manila. Available authentication protocols include LDAP, Kerberos, and Microsoft Active Directory authentication service.

## 17.6. INTRODUCTION TO SECURITY SERVICES

After creating a share and getting its export location, users have no permissions to mount it and operate with files. Users need to explicitly grant access to the new share.

The client authentication and authorization (authN/authZ) can be performed in conjunction with security services. Manila can use LDAP, Kerberos, or Microsoft Active directory if they are supported by the share drivers and back ends.

NOTE

In some cases, it is required to explicitly specify one of the security services, for example, NetApp, EMC and Windows drivers require Active Directory for the creation of shares with the CIFS protocol.

## 17.7. SECURITY SERVICES MANAGEMENT

A *security service* is a manila entity that abstracts a set of options that define a security zone for a particular shared file system protocol, such as an Active Directory domain or a Kerberos domain. The security service contains all of the information necessary for manila to create a server that joins a given domain.

Using the API, users can create, update, view, and delete a security service. Security Services are designed on the following assumptions:

- Projects provide details for the security service.
- Administrators care about security services: they configure the server side of such security services.
- Inside the manila API, a **security_service** is associated with the **share_networks**.
- Share drivers use data in the security service to configure newly created share servers.

When creating a security service, you can select one of these authentication services:

- **LDAP** - The Lightweight Directory Access Protocol. An application protocol for accessing and maintaining distributed directory information services over an IP network.
- **Kerberos** - The network authentication protocol which works on the basis of tickets to allow nodes communicating over a non-secure network to prove their identity to one another in a secure manner.

- **Active Directory** - A directory service that Microsoft developed for Windows domain networks. Uses LDAP, Microsoft's version of Kerberos, and DNS.

Manila allows you to configure a security service with these options:

- A DNS IP address that is used inside the project network.
- An IP address or hostname of a security service.
- A domain of a security service.
- A user or group name that is used by a project.
- A password for a user, if you specify a username.

An existing security service entity can be associated with share network entities that inform manila about security and network configuration for a group of shares. You can also see the list of all security services for a specified share network and disassociate them from a share network.

An administrator and users as share owners can manage access to the shares by creating access rules with authentication through an IP address, user, group, or TLS certificates. Authentication methods depend on which share driver and security service you configure and use. You can then configure a back end to use a specific authentication service, which can operate with clients without manila and keystone.

**NOTE**

Different authentication services are supported by different share drivers. For details of supporting of features by different drivers, see https://docs.openstack.org/manila/latest/admin/share_back_ends_feature_support_mappi

Support for a specific authentication service by a driver does not mean that it can be configured with any shared file system protocol. Supported shared file systems protocols are NFS, CEPHFS, CIFS, GlusterFS, and HDFS. See the driver vendor's documentation for information on a specific driver and its configuration for security services.

Some drivers support security services and other drivers do not support any of the security services mentioned above. For example, Generic Driver with the NFS or the CIFS shared file system protocol supports only authentication method through the IP address.

**NOTE**

In most cases, drivers that support the CIFS shared file system protocol can be configured to use Active Directory and manage access through the user authentication.

- Drivers that support the GlusterFS protocol can be used with authentication using TLS certificates.
- With drivers that support NFS protocol authentication using an IP address is the only supported option.
- Since the HDFS shared file system protocol uses NFS access it also can be configured to authenticate using an IP address.

The recommended configuration for production manila deployments is to create a share with the CIFS share protocol and add to it the Microsoft Active Directory directory service. With this configuration you will get the centralized database and the service that integrates the Kerberos and LDAP approaches.

## 17.8. SHARE ACCESS CONTROL

Users can specify which specific clients have access to the shares they create. Due to the keystone service, shares created by individual users are only visible to themselves and other users within the same project. Manila allows users to create shares that are "publicly" visible. These shares are visible in dashboards of users that belong to other OpenStack projects if the owners grant them access, they might even be able to mount these shares if they are made accessible on the network.

While creating a share, use key **--public** to make your share public for other projects to see it in a list of shares and see its detailed information.

According to the **policy.json** file, an administrator and the users as share owners can manage access to shares by means of creating access rules. Using the **manila access-allow**, **manila access-deny**, and **manila access-list** commands, you can grant, deny and list access to a specified share correspondingly.

NOTE

Manila does not provide end-to-end management of the storage system. You will still need to separately protect the backend system from unauthorized access. As a result, the protection offered by the manila API can still be circumvented if someone compromises the backend storage device, thereby gaining out of band access.

When a share is just created there are no default access rules associated with it and permission to mount it. This could be seen in mounting config for export protocol in use. For example, there is an NFS command **exportfs** or **/etc/exports** file on the storage which controls each remote share and defines hosts that can access it. It is empty if nobody can mount a share. For a remote CIFS server there is **net conf list** command which shows the configuration. The **hosts deny** parameter should be set by the share driver to **0.0.0.0/0** which means that any host is denied to mount the share.

Using manila, you can grant or deny access to a share by specifying one of these supported share access levels:

- **rw** - Read and write (RW) access. This is the default value.
- **ro**- Read-only (RO) access.

NOTE

The RO access level can be helpful in public shares when the administrator gives read and write (RW) access for some certain editors or contributors and gives read-only (RO) access for the rest of users (viewers).

You must also specify one of these supported authentication methods:

- **ip** - Uses an IP address to authenticate an instance. IP access can be provided to clients addressable by well-formed IPv4 or IPv6 addresses or subnets denoted in CIDR notation.
- **cert** - Uses a TLS certificate to authenticate an instance. Specify the TLS identity as the **IDENTKEY**. A valid value is any string up to 64 characters long in the common name (CN) of the certificate.

- **user** - Authenticates by a specified user or group name. A valid value is an alphanumeric string that can contain some special characters and is from 4 to 32 characters long.

NOTE

Supported authentication methods depend on which share driver, security service and shared file system protocol you use. Supported shared file system protocols are MapRFS, CEPHFS, NFS, CIFS, GlusterFS, and HDFS. Supported security services are LDAP, Kerberos protocols, or Microsoft Active Directory service.

To verify that access rules (ACL) were configured correctly for a share, you can list its permissions.

NOTE

When selecting a security service for your share, you will need to consider whether the share driver is able to create access rules using the available authentication methods. Supported security services are LDAP, Kerberos, and Microsoft Active Directory.

## 17.9. SHARE TYPE ACCESS CONTROL

A share type is an administrator-defined *type of service*, comprised of a project visible description, and a list of non-project-visible key-value pairs called *extra specifications*. The **manila-scheduler** uses extra specifications to make scheduling decisions, and drivers control the share creation.

An administrator can create and delete share types, and can also manage extra specifications that give them meaning inside manila. Projects can list the share types and can use them to create new shares. Share types can be created as **public** and **private**. This is the level of visibility for the share type that defines whether other projects can or cannot see it in a share types list and use it to create a new share.

By default, share types are created as public. While creating a share type, use **--is_public** parameter set to **False** to make your share type private which will prevent other projects from seeing it in a list of share types and creating new shares with it. On the other hand, **public** share types are available to every project in a cloud.

Manila allows an administrator to grant or deny access to the **private** share types for projects. You can also get information about the access for a specified private share type.

NOTE

Since share types due to their extra specifications help to filter or choose back ends before users create a share, using access to the share types you can limit clients in choice of specific back ends.

For example, an administrator user in the **admin** project can create a private share type named **my_type** and see it in the list. In the console examples below, the logging in and out is omitted, and environment variables are provided to show the currently logged in user.

```
$ env | grep OS_
...
OS_USERNAME=admin
OS_TENANT_NAME=admin
...
$ manila type-list --all
+----+--------+-----------+-----------+----------------------------------+----------------------+
| ID | Name   | Visibility| is_default| required_extra_specs             | optional_extra_specs  |
+----+--------+-----------+-----------+----------------------------------+----------------------+
| 4..| my_type| private   | -         | driver_handles_share_servers:False| snapshot_support:True |
| 5..| default| public    | YES       | driver_handles_share_servers:True | snapshot_support:True |
+----+--------+-----------+-----------+----------------------------------+----------------------+
```

The **demo** user in the **demo** project can list the types and the private share type named **my_type** is not visible for him.

```
$ env | grep OS_
...
OS_USERNAME=demo
OS_TENANT_NAME=demo
...
$ manila type-list --all
+----+--------+-----------+-----------+----------------------------------+----------------------+
| ID | Name   | Visibility| is_default| required_extra_specs             | optional_extra_specs |
+----+--------+-----------+-----------+----------------------------------+----------------------+
| 5..| default| public    | YES       | driver_handles_share_servers:True| snapshot_support:True|
+----+--------+-----------+-----------+----------------------------------+----------------------+
```

The administrator can grant access to the private share type for the demo project with the project ID equal to **df29a37db5ae48d19b349fe947fada46**:

```
$ env | grep OS_
...
OS_USERNAME=admin
OS_TENANT_NAME=admin
...
$ openstack project list
+----------------------------------+--------------------+
| ID                               | Name               |
+----------------------------------+--------------------+
| ...                              | ...                |
| df29a37db5ae48d19b349fe947fada46 | demo               |
+----------------------------------+--------------------+
$ manila type-access-add my_type df29a37db5ae48d19b349fe947fada46
```

As a result, users in the **demo** project can see the private share type and use it in the share creation:

```
$ env | grep OS_
...
OS_USERNAME=demo
OS_TENANT_NAME=demo
...
$ manila type-list --all
+----+--------+-----------+-----------+----------------------------------+----------------------+
| ID | Name   | Visibility| is_default| required_extra_specs             | optional_extra_specs  |
+----+--------+-----------+-----------+----------------------------------+----------------------+
| 4..| my_type| private   | -         | driver_handles_share_servers:False| snapshot_support:True |
| 5..| default| public    | YES       | driver_handles_share_servers:True | snapshot_support:True |
+----+--------+-----------+-----------+----------------------------------+----------------------+
```

To deny access for a specified project, use **manila type-access-remove <share_type> <project_id>**.

NOTE

For an example that demonstrates the purpose of the share types, consider a situation where you have two back ends: LVM as a public storage and Ceph as a private storage. In this case you can grant access to certain projects and control access with **user/group** authentication method.

## 17.10. POLICIES

The Shared File Systems service API is gated with role-based access control policies. These policies determine which user can access certain APIs in a certain way, and are defined in the service's **policy.json** file.

NOTE

The configuration file **policy.json** may be placed anywhere. The path **/var/lib/config-data/puppet-generated/manila/etc/manila/policy.json** is expected by default.

Whenever an API call is made to manila, the policy engine uses the appropriate policy definitions to determine if the call can be accepted. A policy rule determines under which circumstances the API call is permitted. The **/var/lib/config-data/puppet-generated/manila/etc/manila/policy.json** file has rules where an action is always permitted, when the rule is an empty string: **""**; the rules based on the user role or rules; rules with boolean expressions. Below is a snippet of the **policy.json** file for manila. It can be expected to change between OpenStack releases.

```
{
    "context_is_admin": "role:admin",
    "admin_or_owner": "is_admin:True or project_id:%(project_id)s",
    "default": "rule:admin_or_owner",
    "share_extension:quotas:show": "",
    "share_extension:quotas:update": "rule:admin_api",
    "share_extension:quotas:delete": "rule:admin_api",
    "share_extension:quota_classes": "",
}
```

Users must be assigned to groups and roles that you refer to in your policies. This is done automatically by the service when user management commands are used.

NOTE

Any changes to **/var/lib/config-data/puppet-generated/manila/etc/manila/policy.json** are effective immediately, which allows new policies to be implemented while manila is running. Manual modification of the policy can have unexpected side effects and is not encouraged. Manila does not provide a default policy file; all the default policies are within the code base. You can generate the default policies from the manila code by executing: **oslopolicy-sample-generator --config-file=var/lib/config-data/puppet-generated/manila/etc/manila/manila-policy-generator.conf**

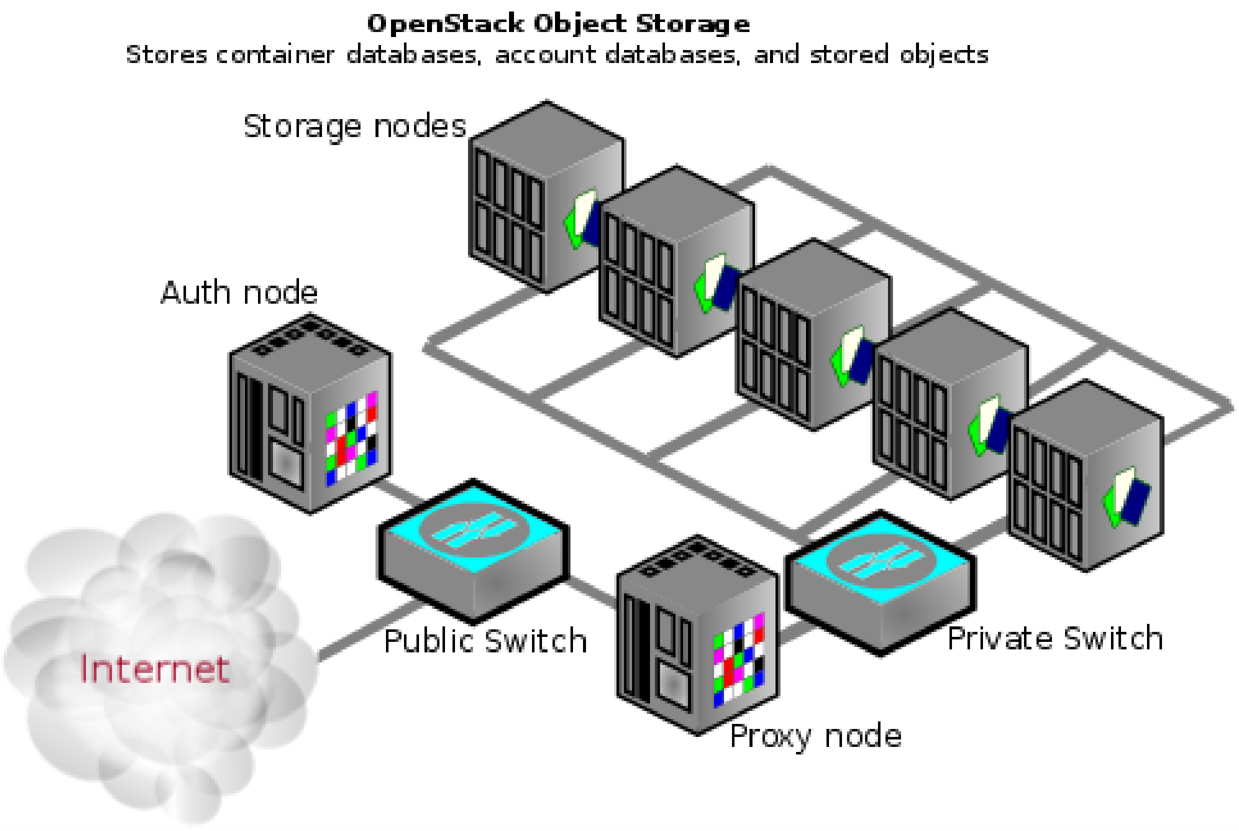
# CHAPTER 18. OBJECT STORAGE

The Object Storage (swift) service stores and retrieves data over HTTP. Objects (blobs of data) are stored in an organizational hierarchy that can be configured to offer anonymous read-only access, ACL defined access, or even temporary access. Swift supports multiple token-based authentication mechanisms implemented through middleware.

Applications store and retrieve data in Object Storage using an industry-standard HTTP RESTful API. The back end swift components follow the same RESTful model, although some APIs (such as those managing durability) are kept private to the cluster.

The components of swift fall into the following primary groups:

- Proxy services
- Auth services
- Storage services
    - Account service
    - Container service
    - Object service



NOTE

An Object Storage installation does not have to be internet-facing and could also be a private cloud with the public switch a part of the organization's internal network infrastructure.

## 18.1. NETWORK SECURITY

Security hardening for swift begins with securing the networking component. See the networking chapter for more information.

For high availability, the rsync protocol is used to replicate data between storage service nodes. In addition, the proxy service communicates with the storage service when relaying data between the client end-point and the cloud environment.

**NOTE**

Swift does not use encryption or authentication with inter-node communications. This is because swift uses the native rsync protocol for performance reasons, and does not use SSH for rsync communications.This is why you see a private switch or private network ([V]LAN) in the architecture diagrams. This data zone should be separate from other OpenStack data networks as well.

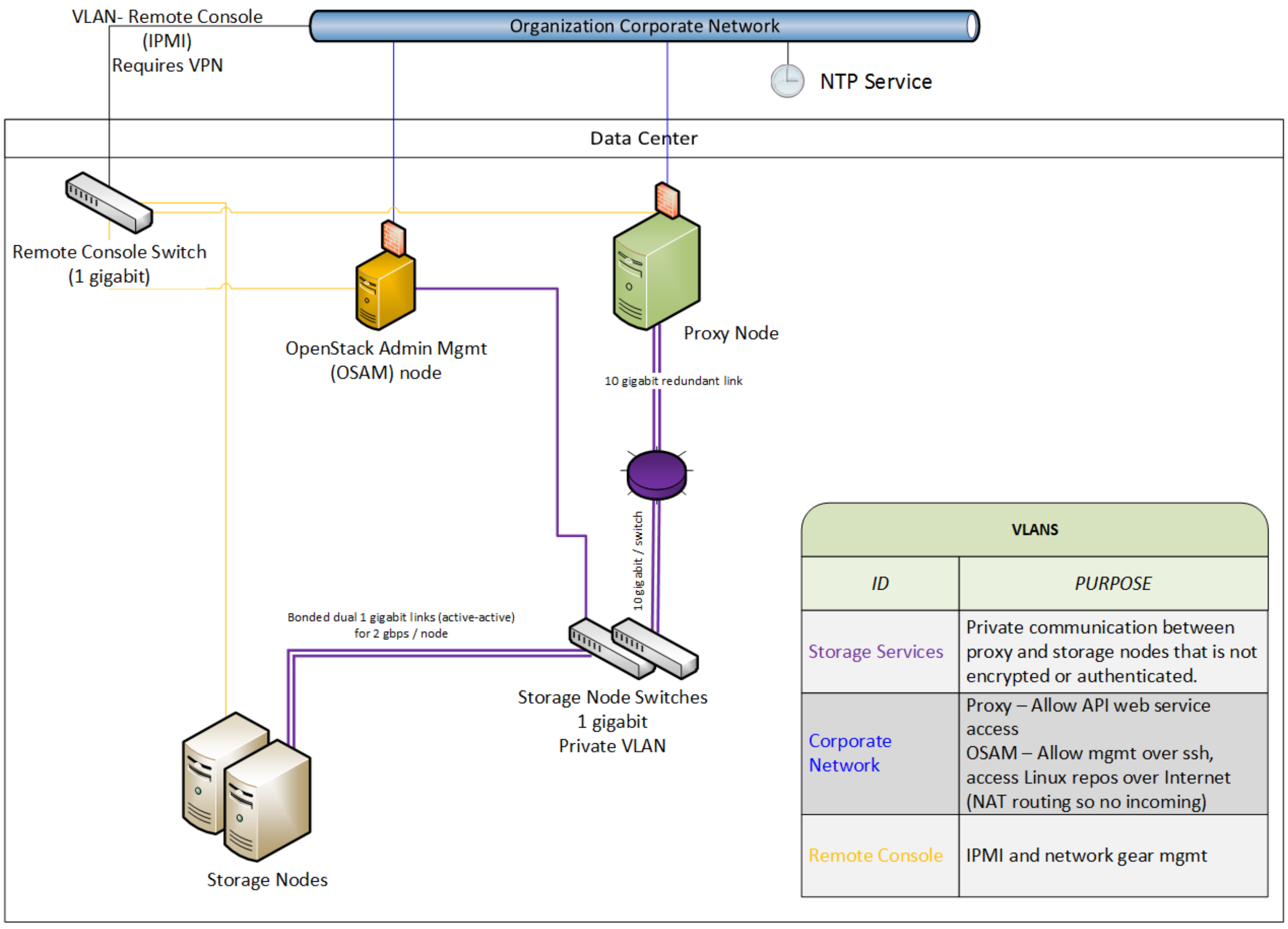
**NOTE**

Use a private (V)LAN network segment for your storage nodes in the data zone.

This requires that the proxy nodes have dual interfaces (physical or virtual):

- One interface as a public interface for consumers to reach.
- Another interface as a private interface with access to the storage nodes.

The following figure demonstrates one possible network architecture, using the Object Storage network architecture with a management node (OSAM):



| VLANS | |
|---|---|
| *ID* | *PURPOSE* |
| Storage Services | Private communication between proxy and storage nodes that is not encrypted or authenticated. |
| Corporate Network | Proxy – Allow API web service access<br>OSAM – Allow mgmt over ssh, access Linux repos over Internet (NAT routing so no incoming) |
| Remote Console | IPMI and network gear mgmt |

## 18.2. RUN SERVICES AS NON-ROOT USER

It is recommend that you configure swift to run under a non-root (**UID 0**) service account. One recommendation is the username **swift** with the primary group **swift**, as deployed by director. Object Storage services include, for example, **proxy-server**, **container-server**, **account-server**.

## 18.3. FILE PERMISSIONS

The **/var/lib/config-data/puppet-generated/swift/etc/swift/** directory contains information about the ring topology and environment configuration. The following permissions are recommended:

```
chown -R root:swift /var/lib/config-data/puppet-generated/swift/etc/swift/*
find /var/lib/config-data/puppet-generated/swift/etc/swift/ -type f -exec chmod 640 {} \;
find /var/lib/config-data/puppet-generated/swift/etc/swift/ -type d -exec chmod 750 {} \;
```

This restriction only allows root to modify configuration files, while still allowing the services to read them, due to their membership in the **swift** group.

## 18.4. SECURING STORAGE SERVICES

The following are the default listening ports for the various storage services:

- Account service - **TCP/6002**
- Container service - **TCP/6001**
- Object Service - **TCP/6000**

- Rsync - **TCP/873**

NOTE

If ssync is used instead of rsync, the object service port is used for maintaining durability.

NOTE

Authentication does not occur at the storage nodes. If you are able to connect to a storage node on one of these ports, you can access or modify data without authentication. To help mitigate this issue, you should follow the recommendations given previously about using a private storage network.

## 18.5. OBJECT STORAGE ACCOUNT TERMINOLOGY

A swift account is not a user account or credential. The following distinctions exist:

- Swift account - A collection of containers (not user accounts or authentication). The authentication system you use will determine which users are associated with the account and how they might access it.
- Swift containers - A collection of objects. Metadata on the container is available for ACLs. The usage of ACLs is dependent on the authentication system used.
- Swift objects - The actual data objects. ACLs at the object level are also available with metadata, and are dependent on the authentication system used.

At each level, you have ACLs that control user access; ACLs are interpreted based on the authentication system in use. The most common type of authentication provider is the Identity Service (keystone); custom authentication providers are also available.

## 18.6. SECURING PROXY SERVICES

A proxy node should have at least two interfaces (physical or virtual): one public and one private. You can use firewalls or service binding to help protect the public interface. The public-facing service is an HTTP web server that processes end-point client requests, authenticates them, and performs the appropriate action. The private interface does not require any listening services, but is instead used to establish outgoing connections to storage nodes on the private storage network.

## 18.7. HTTP LISTENING PORT

Director configures the web services to run under a non-root (no UID 0) user. Using port numbers higher than **1024** help avoid running any part of the web container as root. Normally, clients that use the HTTP REST API (and perform automatic authentication) will retrieve the full REST API URL they require from the authentication response. The OpenStack REST API allows a client to authenticate to one URL and then be redirected to use a completely different URL for the actual service. For example, a client can authenticate to **https://identity.cloud.example.org:55443/v1/auth** and get a response with their authentication key and storage URL (the URL of the proxy nodes or load balancer) of **https://swift.cloud.example.org:44443/v1/AUTH_8980**.

## 18.8. LOAD BALANCER

If the option of using Apache is not feasible, or for performance you wish to offload your TLS work, you might employ a dedicated network device load balancer. This is a common way to provide redundancy and load balancing when using multiple proxy nodes.

If you choose to offload your TLS, ensure that the network link between the load balancer and your proxy nodes are on a private (V)LAN segment such that other nodes on the network (possibly compromised) cannot wiretap (sniff) the unencrypted traffic. If such a breach was to occur, the attacker could gain access to endpoint client or cloud administrator credentials and access the cloud data.

The authentication service you use will determine how you configure a different URL in the responses to endpoint clients, allowing them to use your load balancer instead of an individual proxy node.

## 18.9. OBJECT STORAGE AUTHENTICATION

Object Storage (swift) uses a WSGI model to provide for a middleware capability that not only provides general extensibility, but is also used for authentication of endpoint clients. The authentication provider defines what roles and user types exist. Some use traditional username and password credentials, while others might leverage API key tokens or even client-side x.509 certificates. Custom providers can be integrated using custom middleware.

Object Storage comes with two authentication middleware modules by default, either of which can be used as sample code for developing a custom authentication middleware.

## 18.10. ENCRYPT AT-REST SWIFT OBJECTS

Swift can integrate with Barbican to transparently encrypt and decrypt your stored (at-rest) objects. At-rest encryption is distinct from in-transit encryption, and refers to the objects being encrypted while being stored on disk.

Swift performs these encryption tasks transparently, with the objects being automatically encrypted when uploaded to swift, then automatically decrypted when served to a user. This encryption and decryption is done using the same (symmetric) key, which is stored in Barbican.

**Additional resources**

- Manage secrets with OpenStack Key Manager

## 18.11. ADDITIONAL ITEMS

In **/var/lib/config-data/puppet-generated/swift/etc/swift/swift.conf** on every node, there is a **swift_hash_path_prefix** setting and a **swift_hash_path_suffix** setting. These are provided to reduce the chance of hash collisions for objects being stored and avert one user overwriting the data of another user.

This value should be initially set with a cryptographically secure random number generator and consistent across all nodes. Ensure that it is protected with proper ACLs and that you have a backup copy to avoid data loss.

# CHAPTER 19. MONITORING AND LOGGING

Log management is an important component of monitoring the security status of your OpenStack deployment. Logs provide insight into the BAU actions of administrators, projects, and instances, in addition to the component activities that comprise your OpenStack deployment.

Logs are not only valuable for proactive security and continuous compliance activities, but they are also a valuable information source for investigation and incident response. For example, analyzing the keystone access logs could alert you to failed logins, their frequency, origin IP, and whether the events are restricted to select accounts, among other pertinent information.

The director includes intrusion detection capabilities using AIDE, and CADF auditing for keystone. For more information, see Hardening infrastructure and virtualization.

## 19.1. HARDEN THE MONITORING INFRASTRUCTURE

Centralized logging systems are a high value target for intruders, as a successful breach could allow them to erase or tamper with the record of events. It is recommended you harden the monitoring platform with this in mind. In addition, consider making regular backups of these systems, with failover planning in the event of an outage or DoS.

## 19.2. EXAMPLE EVENTS TO MONITOR

Event monitoring is a more proactive approach to securing an environment, providing real-time detection and response. Multiple tools exist which can aid in monitoring. For an OpenStack deployment, you will need to monitor the hardware, the OpenStack services, and the cloud resource usage.

This section describes some example events you might need to be aware of.

**IMPORTANT**

This list is not exhaustive. You will need to consider additional use cases that might apply to your specific network, and that you might consider anomalous behavior.

- Detecting the absence of log generation is an event of high value. Such a gap might indicate a service failure, or even an intruder who has temporarily switched off logging or modified the log level to hide their tracks.
- Application events, such as start or stop events, that were unscheduled might have possible security implications.
- Operating system events on the OpenStack nodes, such as user logins or restarts. These can provide valuable insight into distinguishing between proper and improper usage of systems.
- Networking bridges going down. This would be an actionable event due to the risk of service outage.
- IPtables flushing events on Compute nodes, and the resulting loss of access to instances.

To reduce security risks from orphaned instances on a user, project, or domain deletion in the Identity service there is discussion to generate notifications in the system and have OpenStack components respond to these events as appropriate such as terminating instances, disconnecting attached volumes, reclaiming CPU and storage resources and so on.

Security monitoring controls such as intrusion detection software, antivirus software, and spyware detection and removal utilities can generate logs that show when and how an attack or intrusion took place. These tools can provide a layer of protection when deployed on the OpenStack nodes. Project users might also want to run such tools on their instances.

# CHAPTER 20. DATA PRIVACY FOR PROJECTS

OpenStack is designed to support multi-tenancy between projects with different data requirements. A cloud operator will need to consider their applicable data privacy concerns and regulations. This chapter addresses aspects of data residency and disposal for OpenStack deployments.

## 20.1. DATA RESIDENCY

The privacy and isolation of data has consistently been cited as the primary barrier to cloud adoption over the past few years. Concerns over who owns data in the cloud and whether the cloud operator can be ultimately trusted as a custodian of this data have been significant issues in the past.

Certain OpenStack services have access to data and metadata belonging to projects or reference project information. For example, project data stored in an OpenStack cloud might include the following items:

- Object Storage objects.
- Compute instance ephemeral filesystem storage.
- Compute instance memory.
- Block Storage volume data.
- Public keys for Compute access.
- Virtual machine images in the Image service.
- Instance snapshots.
- Data passed to Compute’s configuration-drive extension.

Metadata stored by an OpenStack cloud includes the following items (this list is non-exhaustive):

- Organization name.
- User’s “Real Name”.
- Number or size of running instances, buckets, objects, volumes, and other quota-related items.
- Number of hours running instances or storing data.
- IP addresses of users.
- Internally generated private keys for compute image bundling.

## 20.2. DATA DISPOSAL

Good practices suggest that the operator must sanitize cloud system media (digital and non-digital) prior to disposal, prior to release out of organization control, or prior to release for reuse. Sanitization methods should implement an appropriate level of strength and integrity given the specific security domain and sensitivity of the information.

NOTE

The NIST Special Publication 800-53 Revision 4 takes a particular view on this topic:

> The sanitization process removes information from the media such that the information cannot be retrieved or reconstructed. Sanitization techniques, including clearing, purging, cryptographic erase, and destruction, prevent the disclosure of information to unauthorized individuals when such media is reused or released for disposal.

Cloud operators should consider the following when developing general data disposal and sanitization guidelines (as per the NIST recommended security controls):

- Track, document and verify media sanitization and disposal actions.
- Test sanitation equipment and procedures to verify proper performance.
- Sanitize portable, removable storage devices prior to connecting such devices to the cloud infrastructure.
- Destroy cloud system media that cannot be sanitized.

As a result, an OpenStack deployment will need to address the following practices (among others):

- Secure data erasure
- Instance memory scrubbing
- Block Storage volume data
- Compute instance ephemeral storage
- Bare metal server sanitization

### 20.2.1. Data not securely erased

Within OpenStack some data might be deleted, but not securely erased in the context of the NIST standards outlined above. This is generally applicable to most or all of the above-defined metadata and information stored in the database. This might be remediated with database and/or system configuration for auto vacuuming and periodic free-space wiping.

### 20.2.2. Instance memory scrubbing

Specific to various hypervisors is the treatment of instance memory. This behavior is not defined in Compute, although it is generally expected of hypervisors that they will make a best effort to scrub memory either upon deletion of an instance, upon creation of an instance, or both.

## 20.3. ENCRYPTING CINDER VOLUME DATA

Use of the OpenStack volume encryption feature is highly encouraged. This is discussed below in the Data Encryption section under Volume Encryption. When this feature is used, destruction of data is accomplished by securely deleting the encryption key. The end user can select this feature while creating a volume, but note that an admin must perform a one-time set up of the volume encryption feature first.

If the OpenStack volume encryption feature is not used, then other approaches generally would be more difficult to enable. If a back-end plug-in is being used, there might be independent ways of doing encryption or non-standard overwrite solutions. Plug-ins to OpenStack Block Storage store data in a variety of ways. Many plug-ins are specific to a vendor or technology, whereas others are more DIY solutions around file systems (such as LVM or ZFS). Methods for securely destroying data varies between plug-ins, vendors, and file systems.

Some back ends (such as ZFS) support copy-on-write to prevent data exposure. In these cases, reads from unwritten blocks always return zero. Other back ends (such as LVM) might not natively support this, so the cinder plug-in takes the responsibility to override previously written blocks before handing them to users. It is important to review what assurances your chosen volume back end provides and to see what remediation might be available for those assurances not provided.

## 20.4. IMAGE SERVICE DELAY DELETE FEATURES

Image Service has a delayed delete feature, which will pend the deletion of an image for a defined time period. Consider disabling this feature if this behavior is a security concern; you can do this by editing **glance-api.conf** file and setting the **delayed_delete** option to **False**.

## 20.5. COMPUTE SOFT DELETE FEATURES

Compute has a soft-delete feature, which enables an instance that is deleted to be in a soft-delete state for a defined time period. The instance can be restored during this time period. To disable the soft-delete feature, edit the **/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf** file and leave the **reclaim_instance_interval** option empty.

## 20.6. SECURITY HARDENING FOR BARE METAL PROVISIONING

For your bare metal provisioning infrastructure, you should consider security hardening the baseboard management controllers (BMC) in general, and IPMI in particular. For example, you might isolate these systems within a provisioning network, configure non-default and strong passwords, and disable unwanted management functions. For more information, you can refer to the vendor’s guidance on security hardening these components.

NOTE

If possible, consider evaluating Redfish-based BMCs over legacy ones.

## 20.7. HARDWARE IDENTIFICATION

When deploying a server, there might not always have a reliable way to distinguish it from an attacker’s server. This capability might be dependent on the hardware/BMC to some extent, but generally it seems that there is no verifiable means of identification built into servers.

## 20.8. DATA ENCRYPTION

The option exists for implementers to encrypt project data wherever it is stored on disk or transported over a network, such as the OpenStack volume encryption feature described below. This is above and beyond the general recommendation that users encrypt their own data before sending it to their provider.

The importance of encrypting data on behalf of projects is largely related to the risk assumed by a provider that an attacker could access project data. There might be requirements here in government, as well as requirements per-policy, in private contract, or even in case law in regard to private contracts for public cloud providers. Consider getting a risk assessment and legal advice before choosing project encryption policies.

Per-instance or per-object encryption is preferable over, in descending order, per-project, per-host, and per-cloud aggregations. This recommendation is inverse to the complexity and difficulty of implementation. Presently, in some projects it is difficult or impossible to implement encryption as loosely granular as even per-project. Implementers should give serious consideration to encrypting project data.

Often, data encryption relates positively to the ability to reliably destroy project and per-instance data, simply by throwing away the keys. It should be noted that in doing so, it becomes of great importance to destroy those keys in a reliable and secure manner.

Opportunities to encrypt data for users are present:

- Object Storage objects
- Network data

### 20.8.1. Volume encryption

A volume encryption feature in OpenStack supports privacy on a per-project basis. The following features are supported:

- Creation and usage of encrypted volume types, initiated through the dashboard or a command line interface
- Enable encryption and select parameters such as encryption algorithm and key size
- Volume data contained within iSCSI packets is encrypted
- Supports encrypted backups if the original volume is encrypted
- Dashboard indication of volume encryption status. Includes indication that a volume is encrypted, and includes the encryption parameters such as algorithm and key size
- Interface with the Key management service

### 20.8.2. Object Storage objects

Object Storage (swift) supports the optional encryption of object data at rest on storage nodes. The encryption of object data is intended to mitigate the risk of user’s data being read if an unauthorized party were to gain physical access to a disk.

Encryption of data at rest is implemented by middleware that may be included in the proxy server WSGI pipeline. The feature is internal to a swift cluster and not exposed through the API. Clients are unaware that data is encrypted by this feature internally to the swift service; internally encrypted data should never be returned to clients through the swift API.

The following data are encrypted while at rest in swift:

- Object content, for example, the content of an object **PUT** request’s body.
- The entity tag (**ETag**) of objects that have non-zero content.
- All custom user object metadata values. For example, metadata sent using **X-Object-Meta-** prefixed headers with **PUT** or **POST** requests.

Any data or metadata not included in the list above is not encrypted, including:

- Account, container, and object names
- Account and container custom user metadata values
- All custom user metadata names
- Object Content-Type values
- Object size
- System metadata

### 20.8.3. Block Storage performance and back ends

When enabling the operating system, you can enhance the OpenStack Volume Encryption performance by using the hardware acceleration features available in both Intel and AMD processors.

The OpenStack volume encryption feature uses either **dm-crypt** on the host or native **QEMU** encryption support to secure volume data. Red Hat recommends that you use the **LUKS** volume encryption type when creating encrypted volumes.

### 20.8.4. Network data

Project data for Compute nodes could be encrypted over IPsec or other tunnels. This practice is not common or standard in OpenStack, but is an option available to motivated and interested implementers. Likewise, encrypted data remains encrypted as it is transferred over the network.

## 20.9. KEY MANAGEMENT

To address the often mentioned concern of project data privacy, there is significant interest within the OpenStack community to make data encryption more ubiquitous. It is relatively easy for an end-user to encrypt their data prior to saving it to the cloud, and this is a viable path for project objects such as media files, database archives among others. In some instances, client-side encryption is used to encrypt data held by the virtualization technologies which requires client interaction, such as presenting keys, to decrypt data for future use.

Barbican can help projects more seamlessly encrypt the data and have it accessible without burdening the user with key management. Providing encryption and key management services as part of OpenStack eases data-at-rest security adoption and can help address customer concerns about privacy or misuse of data.

The volume encryption feature relies on a key management service, such as the Key Manager service (barbican), for the creation and security-hardened storage of keys.

# CHAPTER 21. MANAGING INSTANCE SECURITY

One of the benefits of running instances in a virtualized environment is the new opportunities for security controls that are not typically available when deploying onto bare metal. Certain technologies can be applied to the virtualization stack that bring improved information assurance for OpenStack deployments. Operators with strong security requirements might want to consider deploying these technologies, however, not all are applicable in every situation. In some cases, technologies might be ruled out for use in a cloud because of prescriptive business requirements. Similarly some technologies inspect instance data such as run state which might be undesirable to the users of the system.

This chapter describes these technologies and the situations where they can be used to help improve security for instances or the underlying nodes. Possible privacy concerns are also highlighted, which can include data passthrough, introspection, or entropy sources.

## 21.1. SUPPLYING ENTROPY TO INSTANCES

Entropy refers to the quality and source of random data that is available to an instance. Cryptographic technologies typically rely on randomness, which requires drawing from a pool of entropy. Entropy starvation occurs when an instance cannot get enough entropy to support the randomness required by cryptographic technologies. Entropy starvation can manifest in instances as something seemingly unrelated. For example, slow boot time might be caused by the instance waiting for SSH key generation. The potential for entropy starvation can also motivate cloud users to use poor quality entropy sources from within the instance, which makes applications running in the cloud less secure.

To provide a high quality source of entropy to the instances you need enough hardware random number generators (HRNG) in the cloud to support the instances. For everyday operations, a modern HRNG can produce enough entropy to support 50-100 Compute nodes. High bandwidth HRNGs can handle more nodes. You must identify the application requirements for a cloud to ensure that sufficient entropy is available.

The VirtIO RNG is a random number generator that uses **/dev/urandom** as the source of entropy by default, to ensure instances are not starved of entropy at boot time. It can also can be configured to use a HRNG, or a tool such as the entropy gathering daemon (EGD) to provide a way to distribute entropy through a deployment. Virtio RNG devices are enabled by default for instances. To disable Virtio RNG devices for instances you must set **hw_rng:allowed** to **False** on the instance flavor.

## 21.2. SCHEDULING INSTANCES TO NODES

Before an instance is created, a host for the image instantiation must be selected. This selection is performed by the **nova-scheduler** which determines how to dispatch compute and volume requests.

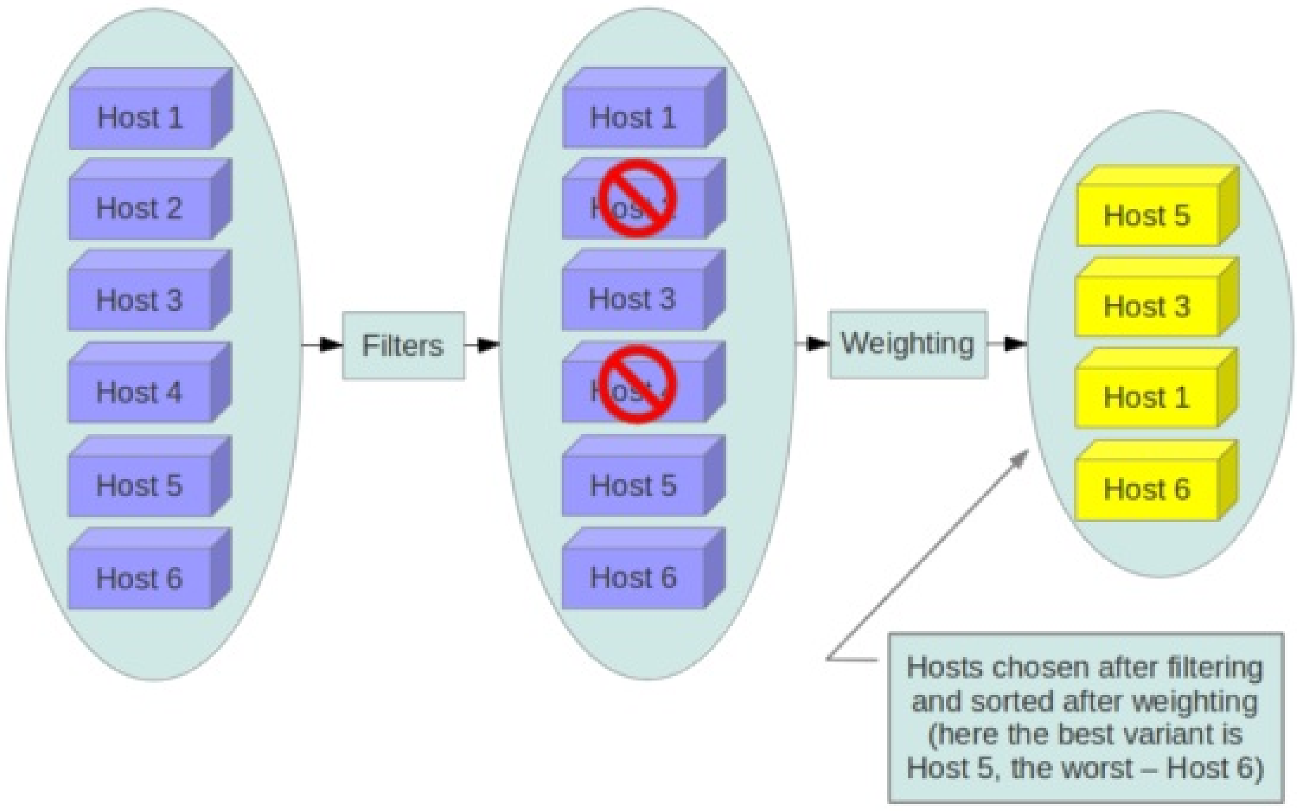
The **FilterScheduler** is the default scheduler for Compute, although other schedulers exist. This capability works in collaboration with *filter hints* to determine where an instance should be started. This process of host selection allows administrators to fulfill many different security and compliance requirements. If data isolation is a primary concern, you could choose to have project instances reside on the same hosts whenever possible. Conversely, you could attempt to have instances reside on as many different hosts as possible for availability or fault tolerance reasons.

Filter schedulers fall under the following main categories:

- *Resource based filters* - Determines the placement of an instance, based on the system resource usage of the hypervisor host sets, and can trigger on free or used properties such as RAM, IO, or CPU utilization.

- *Image based filters* - Delegates instance creation based on the image metadata used, such as the operating system of the VM or type of image used.
- *Environment based filters* - Determines the placement of an instance based on external details, such as within a specific IP range, across availability zones, or on the same host as another instance.
- *Custom criteria* - Delegates instance creation based on user or administrator-provided criteria such as trusts or metadata parsing.

Multiple filters can be applied at once. For example, the **ServerGroupAffinity** filter checks that an instance is created on a member of a specific set of hosts, and the **ServerGroupAntiAffinity** filter checks that same instance is not created on another specific set of hosts. Note that these two filters would usually be both enabled at the same time, and can never conflict with each other as they each check for the value of a given property, and cannot both be true at the same time.



**IMPORTANT**

Consider disabling filters that parse objects that are provided by users, or could be manipulated (such as metadata).

## 21.3. USING TRUSTED IMAGES

In a cloud environment, users work with either pre-installed images or images they upload themselves. In both cases, users should be able to ensure the image they are using has not been tampered with. The ability to verify images is a fundamental imperative for security. A chain of trust is needed from the source of the image to the destination where it is used. This can be accomplished by signing images obtained from trusted sources and by verifying the signature prior to use. Various ways to obtain and create verified images will be discussed below, followed by a description of the image signature verification feature.

## 21.4. CREATING IMAGES

For guidance on how to create and upload images to Red Hat OpenStack Image service (glance), see Creating and Managing Images. Use trusted images for your environment for increased security, and use the hardening guidelines of your organization for further protection. You can get images for your environment in one of several ways:

**Download instance media**

To obtain boot media from a trusted source, download images from the official Red Hat source and use the SHA256SUM for validation.

**Create an image from an ISO**

For details on creating an image from an installation process see Creating a Red Hat Enterprise Linux 9 image.

**Use an image builder**

You can use **disk-image-builder** to produce minimal systems that have just the required components for their purpose within OpenStack. For information on creating a custom image with **disk-image-builder**, see Composing a customized RHEL system image .

## 21.5. VERIFYING IMAGE SIGNATURES

You can enable image signature verification to ensure that your Image service (glance) images do not contain unauthorized changes before the Compute service (nova) starts the instance. With this feature enabled, you prevent a new instance from starting that may include malware or security vulnerabilities.

### Prerequisites

- You have an installed Red Hat OpenStack Platform director environment.
- You are logged into the director as stack.

### Procedure

1. In your heat templates, enable instance signature verfication by setting the value of **True** to the **VerifyGlanceSignatures** parameter:

   ```
   parameter_defaults:
     VerifyGlanceSignatures: True
   ```

2. Ensure that the template that you use to modify the **VerifyGlanceSignatures** parameter is included in your **openstack overcloud deploy** script, and rerun the deploy script.

**NOTE**

If you create an instance with an image that you have not signed, the image fails verification and the instance does not start. For more information on signing your images, see Signing Image service images .

## 21.6. MIGRATING INSTANCES

OpenStack and the underlying virtualization layers provide for the live migration of images between OpenStack nodes, allowing you to seamlessly perform rolling upgrades of your Compute nodes without instance downtime. However, live migrations also carry significant risk. To understand the risks involved, the following are the high-level steps performed during a live migration:

1. Start instance on destination host
2. Transfer memory
3. Stop the guest and sync disks
4. Transfer the state
5. Start the guest

**NOTE**

Certain operations, such as cold migration, resize, and shelve can all result in some amount of transferring the instance's data to other services, across the network, among others.

### 21.6.1. Live migration risks

At various stages of the live migration process, the contents of an instance's run time memory and disk are transmitted over the network in plain text. Consequently there are multiple risks that need to be addressed when using live migration. The following non-exhaustive list details some of these risks:

- Denial of Service (DoS): If something fails during the migration process, the instance could be lost.
- Data exposure: Memory or disk transfers must be handled securely.
- Data manipulation: If memory or disk transfers are not handled securely, then an attacker could manipulate user data during the migration.
- Code injection: If memory or disk transfers are not handled securely, then an attacker could manipulate executables, either on disk or in memory, during the migration.

### 21.6.2. Disable live migration

Currently, live migration is enabled in OpenStack by default. Live migrations are admin-only tasks by default, so a user cannot initiate this operation, only administrators (which are presumably trusted). Live migrations can be disabled by adding the following lines to the nova **policy.json** file:

```
"compute_extension:admin_actions:migrate": "!",
"compute_extension:admin_actions:migrateLive": "!",
```

Alternatively, live migration can be expected to fail when blocking TCP ports **49152** through **49261**, or ensuring that the nova user does not have passwordless SSH access between compute hosts.

Note that SSH configuration for live migration is significantly locked down: A new user is created (nova_migration) and the SSH keys are restricted to that user, and only for use on the allowed networks. A wrapper script then restricts the commands that can be run (for example, netcat on the libvirt socket).

### 21.6.3. Encrypted live migration

Live migration traffic transfers the contents of disk and memory of a running instance in plain text, and is currently hosted on the Internal API network by default.

If there is a sufficient requirement (such as upgrades) for keeping live migration enabled, then libvirtd can provide encrypted tunnels for the live migrations. However, this feature is not exposed in either the OpenStack Dashboard or nova-client commands, and can only be accessed through manual configuration of libvirtd. The live migration process then changes to the following high-level steps:

1. Instance data is copied from the hypervisor to libvirtd.
2. An encrypted tunnel is created between libvirtd processes on both source and destination hosts.
3. The destination libvirtd host copies the instances back to an underlying hypervisor.

NOTE

For Red Hat OpenStack Platform 13, the recommended approach is to use tunnelled migration, which is enabled by default when using Ceph as the back end. For more information, see https://docs.openstack.org/nova/queens/configuration/config.html#libvirt.live_migration_t

## 21.7. MONITORING, ALERTING, AND REPORTING

Instances are a server image capable of being replicated across hosts. Consequently, it would be a good practice to apply logging similarly between physical and virtual hosts. Operating system and application events should be logged, including access events to hosts and data, user additions and removals, privilege changes, and others as dictated by your requirements. Consider exporting the results to a log aggregator that collects log events, correlates them for analysis, and stores them for reference or further action. One common tool to do this is an ELK stack, or Elasticsearch, Logstash, and Kibana.

NOTE

These logs should be reviewed regularly, or even monitored within a live view performed by a network operations center (NOC).

You will need to further determine which events will trigger an alert that is subsequently sent to a responder for action.

**Additional resources**

- Monitoring Tools Configuration Guide

## 21.8. UPDATES AND PATCHES

A hypervisor runs independent virtual machines. This hypervisor can run in an operating system or directly on the hardware (called bare metal). Updates to the hypervisor are not propagated down to the virtual machines. For example, if a deployment is using KVM and has a set of CentOS virtual machines, an update to KVM will not update anything running on the CentOS virtual machines.

Consider assigning clear ownership of virtual machines to owners, who are then responsible for the hardening, deployment, and continued functionality of the virtual machines. You should also have a plan to regularly deploy updates, while first testing them in an environment that resembles production.

## 21.9. FIREWALLS AND INSTANCE PROFILES

Most common operating systems include host-based firewalls for an additional layer of security. While instances should run as few applications as possible (to the point of being single-purpose instances, if possible), all applications running on an instance should be profiled to determine which system resources the application needs access to, the lowest level of privilege required for it to run, and what the expected network traffic is that will be going into and coming from the virtual machine. This expected traffic should be added to the host-based firewall as allowed traffic, along with any necessary logging and management communication such as SSH or RDP. All other traffic should be explicitly denied in the firewall configuration.

On Linux instances, the application profile above can be used in conjunction with a tool like **audit2allow** to build an SELinux policy that will further protect sensitive system information on most Linux distributions. SELinux uses a combination of users, policies and security contexts to compartmentalize the resources needed for an application to run, and segmenting it from other system resources that are not needed.

**NOTE**

Red Hat OpenStack Platform has SELinux enabled by default, with policies that are customized for OpenStack services. Consider reviewing these polices regularly, as required.

## 21.10. SECURITY GROUPS

OpenStack provides security groups for both hosts and the network to add defense-in-depth to the instances in a given project. These are similar to host-based firewalls as they allow or deny incoming traffic based on port, protocol, and address. However, security group rules are applied to incoming traffic only, while host-based firewall rules can be applied to both incoming and outgoing traffic. It is also possible for host and network security group rules to conflict and deny legitimate traffic. Consider checking that security groups are configured correctly for the networking being used. See Security groups in this guide for more detail.

**NOTE**

You should keep security groups and port security enabled unless you specifically need them to be disabled. To build on the defense-in-depth approach, it is recommended that you apply granular rules to instances.

## 21.11. ACCESSING THE INSTANCE CONSOLE

By default, an instance's console is remotely accessible through a virtual console. This can be useful for troubleshooting purposes. Red Hat OpenStack Platform uses VNC for remote console access.

- Consider locking down the VNC port using firewall rules. By default, **nova_vnc_proxy** uses **6080** and **13080**.
- Confirm that the VNC traffic is encrypted by TLS. For director-based deployments, start with **UseTLSTransportForVnc**.

## 21.12. CERTIFICATE INJECTION

If you need to SSH into your instances, you can configure Compute to automatically inject the required SSH key into the instance upon creation.

### Additional resources

- Creating images.

# CHAPTER 22. MESSAGE QUEUING

Message queuing services facilitate inter-process communication in OpenStack. This is done using these message queuing service back ends:

- RabbitMQ - Red Hat OpenStack Platform uses RabbitMQ by default.
- Qpid

Both RabbitMQ and Qpid are Advanced Message Queuing Protocol (AMQP) frameworks, which provide message queues for peer-to-peer communication. Queue implementations are typically deployed as a centralized or decentralized pool of queue servers.

Message queues effectively facilitate command and control functions across OpenStack deployments. Once access to the queue is permitted, no further authorization checks are performed. Services accessible through the queue do validate the contexts and tokens within the actual message payload. However, you must note the expiration date of the token because tokens are potentially re-playable and can authorize other services in the infrastructure.

OpenStack does not support message-level confidence, such as message signing. Consequently, you must secure and authenticate the message transport itself. For high-availability (HA) configurations, you must perform queue-to-queue authentication and encryption.

## 22.1. MESSAGING TRANSPORT SECURITY

AMQP based solutions (Qpid and RabbitMQ) support transport-level security using TLS.

Consider enabling transport-level cryptography for your message queue. Using TLS for the messaging client connections provides protection of the communications from tampering and eavesdropping in-transit to the messaging server. Guidance is included below on how TLS is typically configured for the two popular messaging servers: Qpid and RabbitMQ. When configuring the trusted certificate authority (CA) bundle that your messaging server uses to verify client connections, it is recommended that this be limited to only the CA used for your nodes, preferably an internally managed CA. The bundle of trusted CAs will determine which client certificates will be authorized and pass the client-server verification step of the setting up the TLS connection.

**NOTE**

When installing the certificate and key files, ensure that the file permissions are restricted, for example using **chmod 0600**, and the ownership is restricted to the messaging server daemon user to prevent unauthorized access by other processes and users on the messaging server.

### 22.1.1. RabbitMQ server SSL configuration

The following lines should be added to the system-wide RabbitMQ configuration file, typically **/etc/rabbitmq/rabbitmq.config**:

```
[
  {rabbit, [
    {tcp_listeners, [] },
    {ssl_listeners, [{"<IP address or hostname of management network interface>", 5671}] },
    {ssl_options, [{cacertfile,"/etc/ssl/cacert.pem"},
                   {certfile,"/etc/ssl/rabbit-server-cert.pem"},
                   {keyfile,"/etc/ssl/rabbit-server-key.pem"},
                {verify,verify_peer},
                {fail_if_no_peer_cert,true}]}
   ]}
].
```

NOTE

The **tcp_listeners** option is set to **[]** to prevent it from listening on a non-SSL port. The **ssl_listeners** option should be restricted to only listen on the management network for the services.

## 22.2. QUEUE AUTHENTICATION AND ACCESS CONTROL

RabbitMQ and Qpid offer authentication and access control mechanisms for controlling access to queues.

Simple Authentication and Security Layer (SASL) is a framework for authentication and data security in Internet protocols. Both RabbitMQ and Qpid offer SASL and other pluggable authentication mechanisms beyond simple usernames and passwords that allow for increased authentication security. While RabbitMQ supports SASL, support in OpenStack does not currently allow for requesting a specific SASL authentication mechanism. RabbitMQ support in OpenStack allows for either username and password authentication over an unencrypted connection or user name and password in conjunction with X.509 client certificates to establish the secure TLS connection.

Consider configuring X.509 client certificates on all the OpenStack service nodes for client connections to the messaging queue and where possible (currently only Qpid) perform authentication with X.509 client certificates. When using usernames and passwords, accounts should be created per-service and node for finer grained auditability of access to the queue.

Before deployment, consider the TLS libraries that the queuing servers use. Qpid uses Mozilla's NSS library, whereas RabbitMQ uses Erlang's TLS module which uses OpenSSL.

## 22.3. OPENSTACK SERVICE CONFIGURATION FOR RABBITMQ

This section describes the typical RabbitMQ configuration for OpenStack services:

```
[DEFAULT]
rpc_backend = nova.openstack.common.rpc.impl_kombu
rabbit_use_ssl = True
rabbit_host = RABBIT_HOST
rabbit_port = 5671
rabbit_user = compute01
rabbit_password = RABBIT_PASS
kombu_ssl_keyfile = /etc/ssl/node-key.pem
kombu_ssl_certfile = /etc/ssl/node-cert.pem
kombu_ssl_ca_certs = /etc/ssl/cacert.pem
```

NOTE

Replace **RABBIT_PASS** with a suitable password.

## 22.4. OPENSTACK SERVICE CONFIGURATION FOR QPID

This section describes the typical Qpid configuration for OpenStack services:

```
[DEFAULT]
rpc_backend = nova.openstack.common.rpc.impl_qpid
qpid_protocol = ssl
qpid_hostname = <IP or hostname of management network interface of messaging server>
qpid_port = 5671
qpid_username = compute01
qpid_password = QPID_PASS
```

NOTE

Replace **QPID_PASS** with a suitable password.

Optionally, if using SASL with Qpid specify the SASL mechanisms in use by adding:

```
qpid_sasl_mechanisms = <space separated list of SASL mechanisms to use for auth>
```

## 22.5. MESSAGE QUEUE PROCESS ISOLATION AND POLICY

Each project provides a number of services which send and consume messages. Each binary which sends a message is expected to consume messages, if only replies, from the queue.

Message queue service processes should be isolated from each other and other processes on a machine.

## 22.6. NAMESPACES

Linux uses namespaces to assign processes into independent domains. Other parts of this guide cover system compartmentalization in more detail.

# CHAPTER 23. SECURING ENDPOINTS IN RED HAT OPENSTACK PLATFORM

The process of engaging with an OpenStack cloud begins by querying an API endpoint. While there are different challenges for public and private endpoints, these are high value assets that can pose a significant risk if compromised.

This chapter recommends security enhancements for both public and private-facing API endpoints.

## 23.1. INTERNAL API COMMUNICATIONS

OpenStack provides both public-facing, internal admin, and private API endpoints. By default, OpenStack components use the publicly defined endpoints. The recommendation is to configure these components to use the API endpoint within the proper security domain. The internal admin endpoint allows further elevated access to keystone, so it might be desirable to further isolate this.

Services select their respective API endpoints based on the OpenStack service catalog. These services might not obey the listed public or internal API endpoint values. This can lead to internal management traffic being routed to external API endpoints.

## 23.2. CONFIGURE INTERNAL URLS IN THE IDENTITY SERVICE CATALOG

The Identity service catalog should be aware of your internal URLs. While this feature is not used by default, it may be available through configuration. In addition, it should be forward-compatible with expectant changes once this behavior becomes the default.

Consider isolating the configured endpoints from a network level, given that they have different levels of access. The Admin endpoint is intended for access by cloud administrators, as it provides elevated access to keystone operations not available on the internal or public endpoints. The internal endpoints are intended for uses internal to the cloud (for example, by OpenStack services), and usually would not be accessible outside of the deployment network. The public endpoints should be TLS-enabled, and the only API endpoints accessible outside of the deployment for cloud users to operate on.

Registration of an internal URL for an endpoint is automated by director. For more information, see https://github.com/openstack/tripleo-heat-templates/blob/a7857d6dfcc875eb2bc611dd9334104c18fe8ac6/network/endpoints/build_endpoint_map

## 23.3. CONFIGURE APPLICATIONS FOR INTERNAL URLS

You can force some services to use specific API endpoints. As a result, it is recommended that any OpenStack service that contacts the API of another service must be explicitly configured to access the proper internal API endpoint.

Each project might present an inconsistent way of defining target API endpoints. Future releases of OpenStack seek to resolve these inconsistencies through consistent use of the Identity service catalog.

## 23.4. PASTE AND MIDDLEWARE

Most API endpoints and other HTTP services in OpenStack use the Python Paste Deploy library. From a security perspective, this library enables manipulation of the request filter pipeline through the application’s configuration. Each element in this chain is referred to as middleware. Changing the order of filters in the pipeline or adding additional middleware might have unpredictable security impact.

Commonly, implementers add middleware to extend OpenStack's base functionality. Consider giving careful consideration to the potential exposure introduced by the addition of non-standard software components to the HTTP request pipeline.

## 23.5. API ENDPOINT PROCESS ISOLATION AND POLICY

You can isolate API endpoint processes for increased security. Consider deploying API endpoints within the public security domain on separate hosts for increased isolation.

### 23.5.1. Configuring a policy to restrict metadef APIs

In Red Hat OpenStack Platform (RHOSP), users can define key value pairs and tag metadata with metadata definition (metadef) APIs. Currently, there is no limit on the number of metadef namespaces, objects, properties, resources, or tags that users can create.

Metadef APIs can leak information to unauthorized users. A malicious user can exploit the lack of restrictions and fill the Image service (glance) database with unlimited resources, which can create a Denial of Service (DoS) style attack.

Image service policies control metadef APIs. However, the default policy setting for metadef APIs allows all users to create or read the metadef information. Because metadef resources are not isolated to the owner, metadef resources with potentially sensitive names, such as internal infrastructure details or customer names, can expose that information to malicious users.

To make the Image service (glance) more secure, restrict metadef modification APIs to admin-only access by default in your Red Hat OpenStack Platform (RHOSP) deployments.

**Procedure**

1. As a cloud administrator, create a separate heat template environment file, such as **lock-down-glance-metadef-api.yaml**, to contain policy overrides for the Image service metadef API:

```
...
parameter_defaults:
  GlanceApiPolicies: {
      glance-metadef_default: { key: 'metadef_default', value: '' },
      glance-metadef_admin: { key: 'metadef_admin', value: 'role:admin' },
      glance-get_metadef_namespace: { key: 'get_metadef_namespace', value:
'rule:metadef_default' },
      glance-get_metadef_namespaces: { key: 'get_metadef_namespaces', value:
'rule:metadef_default' },
      glance-modify_metadef_namespace: { key: 'modify_metadef_namespace', value:
'rule:metadef_admin' },
      glance-add_metadef_namespace: { key: 'add_metadef_namespace', value:
'rule:metadef_admin' },
      glance-delete_metadef_namespace: { key: 'delete_metadef_namespace', value:
'rule:metadef_admin' },
      glance-get_metadef_object: { key: 'get_metadef_object', value: 'rule:metadef_default' },
      glance-get_metadef_objects: { key: 'get_metadef_objects', value: 'rule:metadef_default'
},
      glance-modify_metadef_object: { key: 'modify_metadef_object', value:
'rule:metadef_admin' },
      glance-add_metadef_object: { key: 'add_metadef_object', value: 'rule:metadef_admin' },
      glance-delete_metadef_object: { key: 'delete_metadef_object', value:
'rule:metadef_admin' },
        glance-list_metadef_resource_types: { key: 'list_metadef_resource_types', value:
'rule:metadef_default' },
        glance-get_metadef_resource_type: { key: 'get_metadef_resource_type', value:
'rule:metadef_default' },
        glance-add_metadef_resource_type_association: { key:
'add_metadef_resource_type_association', value: 'rule:metadef_admin' },
        glance-remove_metadef_resource_type_association: { key:
'remove_metadef_resource_type_association', value: 'rule:metadef_admin' },
        glance-get_metadef_property: { key: 'get_metadef_property', value:
'rule:metadef_default' },
        glance-get_metadef_properties: { key: 'get_metadef_properties', value:
'rule:metadef_default' },
        glance-modify_metadef_property: { key: 'modify_metadef_property', value:
'rule:metadef_admin' },
        glance-add_metadef_property: { key: 'add_metadef_property', value:
'rule:metadef_admin' },
        glance-remove_metadef_property: { key: 'remove_metadef_property', value:
'rule:metadef_admin' },
        glance-get_metadef_tag: { key: 'get_metadef_tag', value: 'rule:metadef_default' },
        glance-get_metadef_tags: { key: 'get_metadef_tags', value: 'rule:metadef_default' },
        glance-modify_metadef_tag: { key: 'modify_metadef_tag', value: 'rule:metadef_admin' },
        glance-add_metadef_tag: { key: 'add_metadef_tag', value: 'rule:metadef_admin' },
        glance-add_metadef_tags: { key: 'add_metadef_tags', value: 'rule:metadef_admin' },
        glance-delete_metadef_tag: { key: 'delete_metadef_tag', value: 'rule:metadef_admin' },
        glance-delete_metadef_tags: { key: 'delete_metadef_tags', value: 'rule:metadef_admin' }
  }

…
```

2. Include the environment file that contains the policy overrides in the deployment command with the **-e** option when you deploy the overcloud:

```
$ openstack overcloud deploy -e lock-down-glance-metadef-api.yaml
```

### 23.5.2. Enabling metadef APIs

If you previously restricted metadata definition (metadef) APIs or want to relax the new defaults, you can override metadef modification policies to allow users to update their respective resources.

**IMPORTANT**

Cloud administrators with users who depend on write access to the metadef APIs can make those APIs accessible to all users. In this type of configuration, however, there is the potential to unintentionally leak sensitive resource names, such as customer names and internal projects. Administrators must audit their systems to identify previously created resources that might be vulnerable even if only read access is enabled for all users.

**Procedure**

1. As a cloud administrator, log in to the undercloud and create a file for policy overrides. For example:

```
$ cat open-up-glance-api-metadef.yaml
```

2. Configure the policy override file to allow metadef API read-write access to all users:

```
GlanceApiPolicies: {
    glance-metadef_default: { key: 'metadef_default', value: '' },
    glance-get_metadef_namespace: { key: 'get_metadef_namespace', value:
'rule:metadef_default' },
    glance-get_metadef_namespaces: { key: 'get_metadef_namespaces', value:
'rule:metadef_default' },
    glance-modify_metadef_namespace: { key: 'modify_metadef_namespace', value:
'rule:metadef_default' },
    glance-add_metadef_namespace: { key: 'add_metadef_namespace', value:
'rule:metadef_default' },
    glance-delete_metadef_namespace: { key: 'delete_metadef_namespace', value:
'rule:metadef_default' },
    glance-get_metadef_object: { key: 'get_metadef_object', value: 'rule:metadef_default' },
    glance-get_metadef_objects: { key: 'get_metadef_objects', value: 'rule:metadef_default' },
    glance-modify_metadef_object: { key: 'modify_metadef_object', value:
'rule:metadef_default' },
    glance-add_metadef_object: { key: 'add_metadef_object', value: 'rule:metadef_default' },
    glance-delete_metadef_object: { key: 'delete_metadef_object', value: 'rule:metadef_default'
},
    glance-list_metadef_resource_types: { key: 'list_metadef_resource_types', value:
'rule:metadef_default' },
    glance-get_metadef_resource_type: { key: 'get_metadef_resource_type', value:
'rule:metadef_default' },
    glance-add_metadef_resource_type_association: { key:
'add_metadef_resource_type_association', value: 'rule:metadef_default' },
    glance-remove_metadef_resource_type_association: { key:
'remove_metadef_resource_type_association', value: 'rule:metadef_default' },
    glance-get_metadef_property: { key: 'get_metadef_property', value: 'rule:metadef_default'
},
    glance-get_metadef_properties: { key: 'get_metadef_properties', value:
'rule:metadef_default' },
    glance-modify_metadef_property: { key: 'modify_metadef_property', value:
'rule:metadef_default' },
    glance-add_metadef_property: { key: 'add_metadef_property', value: 'rule:metadef_default'
},
    glance-remove_metadef_property: { key: 'remove_metadef_property', value:
'rule:metadef_default' },
    glance-get_metadef_tag: { key: 'get_metadef_tag', value: 'rule:metadef_default' },
    glance-get_metadef_tags: { key: 'get_metadef_tags', value: 'rule:metadef_default' },
    glance-modify_metadef_tag: { key: 'modify_metadef_tag', value: 'rule:metadef_default' },
    glance-add_metadef_tag: { key: 'add_metadef_tag', value: 'rule:metadef_default' },
    glance-add_metadef_tags: { key: 'add_metadef_tags', value: 'rule:metadef_default' },
    glance-delete_metadef_tag: { key: 'delete_metadef_tag', value: 'rule:metadef_default' },
    glance-delete_metadef_tags: { key: 'delete_metadef_tags', value: 'rule:metadef_default' }
  }
```

NOTE

You must configure all metadef policies to use **rule:metadeta_default**.

3. Include the new policy file in the deployment command with the **-e** option when you deploy the overcloud:

```
$ openstack overcloud deploy -e open-up-glance-api-metadef.yaml
```

## 23.6. CHANGING THE SSL/TLS CIPHER AND RULES FOR HAPROXY

If you enabled SSL/TLS in the overcloud, consider hardening the SSL/TLS ciphers and rules that are used with the HAProxy configuration. By hardening the SSL/TLS ciphers, you help avoid SSL/TLS vulnerabilities, such as the POODLE vulnerability.

1. Create a heat template environment file called **tls-ciphers.yaml**:

```
touch ~/templates/tls-ciphers.yaml
```

2. Use the **ExtraConfig** hook in the environment file to apply values to the **tripleo::haproxy::ssl_cipher_suite** and **tripleo::haproxy::ssl_options** hieradata:

```
parameter_defaults:
  ExtraConfig:
    tripleo::haproxy::ssl_cipher_suite:: `DHE-RSA-AES128-CCM:DHE-RSA-AES256-
CCM:DHE-RSA-AES128-GCM-SHA256:DHE-RSA-AES256-GCM-SHA384:ECDHE-ECDSA-
AES128-CCM:ECDHE-ECDSA-AES256-CCM:ECDHE-ECDSA-AES128-GCM-
SHA256:ECDHE-ECDSA-AES256-GCM-SHA384:ECDHE-ECDSA-CHACHA20-
POLY1305:ECDHE-ECDSA-CHACHA20-POLY1305:ECDHE-RSA-AES128-GCM-
SHA256:ECDHE-RSA-AES256-GCM-SHA384:ECDHE-RSA-CHACHA20-POLY1305`

    tripleo::haproxy::ssl_options:: no-sslv3 no-tls-tickets
```

NOTE

The cipher collection is one continuous line.

3. Include the **tls-ciphers.yaml** environment file with the overcloud deploy command when deploying the overcloud:

```
openstack overcloud deploy --templates \
...
-e /home/stack/templates/tls-ciphers.yaml
...
```

## 23.7. NETWORK POLICY

API endpoints will typically span multiple security zones, so you must pay particular attention to the separation of the API processes. For example, at the network design level, you can consider restricting access to specified systems only. See the guidance on security zones for more information.

With careful modeling, you can use network ACLs and IDS technologies to enforce explicit point-to-point communication between network services. As a critical cross domain service, this type of explicit enforcement works well for OpenStack's message queue service.

To enforce policies, you can configure services, host-based firewalls (such as iptables), local policy (SELinux), and optionally global network policy.

## 23.8. MANDATORY ACCESS CONTROLS

You should isolate API endpoint processes from each other and other processes on a machine. The configuration for those processes should be restricted to those processes by Discretionary Access Controls (DAC) and Mandatory Access Controls (MAC). The goal of these enhanced access controls is to aid in the containment of API endpoint security breaches.

## 23.9. API ENDPOINT RATE-LIMITING

Rate Limiting is a means to control the frequency of events received by a network based application. When robust rate limiting is not present, it can result in an application being susceptible to various denial of service attacks. This is especially true for APIs, which by their nature are designed to accept a high frequency of similar request types and operations.

It is recommended that all endpoints (especially public) are give an extra layer of protection, for example, using physical network design, a rate-limiting proxy, or web application firewall.

It is key that the operator carefully plans and considers the individual performance needs of users and services within their OpenStack cloud when configuring and implementing any rate limiting functionality.

**NOTE**

For Red Hat OpenStack Platform deployments, all services are placed behind load balancing proxies.

# CHAPTER 24. IMPLEMENTING FEDERATION

WARNING

Red Hat does not support federation at this time. This feature should only be used for testing, and should not be deployed in a production environment.

## 24.1. FEDERATE WITH IDM USING RED HAT SINGLE SIGN-ON

You can use Red Hat Single Sign-On (RH-SSO) to federate your IdM users for OpenStack authentication (authN). Federation allows your IdM users to login to the OpenStack Dashboard without revealing their credentials to any OpenStack services. Instead, when Dashboard needs a user's credentials, it will forward the user to Red Hat Single Sign-On (RH-SSO) and allow them to enter their IdM credentials there. As a result, RH-SSO asserts back to Dashboard that the user has successfully authenticated, and Dashboard then allows the user to access the project.

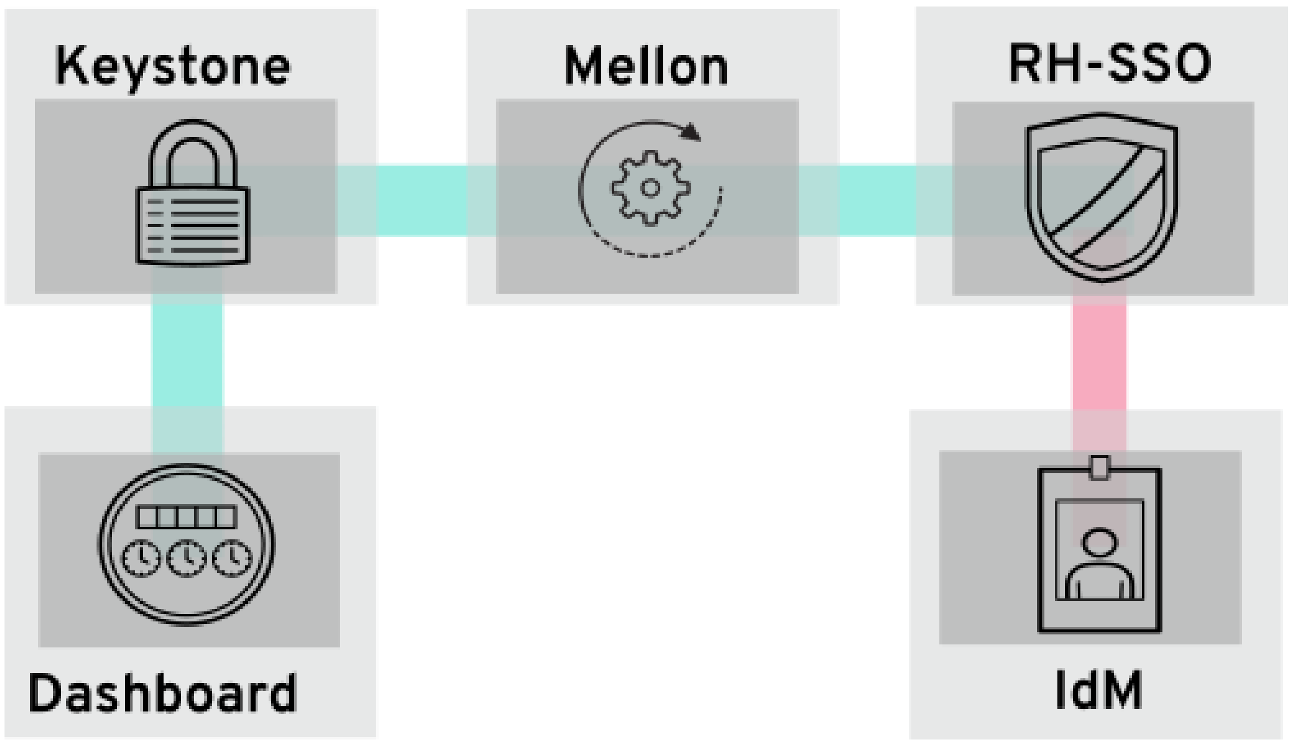
In the diagram below, keystone (the SP) communicates with RH-SSO[id=*the-federation-workflow_implementing-federation*] = The federation workflow

This section describes how the Identity service (keystone), RH-SSO and IdM interact with each other. Federation in OpenStack uses the concept of Identity Providers and Service Providers:

**Identity Provider** (IdP) - the service that stores the user accounts. In this case, the user accounts held in IdM, are presented to Keystone using RH-SSO.

**Service Provider** (SP) - the service that requires authentication from the users in the IdP. In this case, keystone is the service provider that grants Dashboard access to IdM users.

In the diagram below, keystone (the SP) communicates with RH-SSO (the IdP), which is also able to serve as a universal adapter for other IdPs. In this configuration, you can point keystone at RH-SSO, and RH-SSO will forward requests on to the Identity Providers that it supports (known as authentication modules), these currently include IdM and Active Directory. This is done by having the Service Provider (SP) and Identity Provider (IdP) exchange metadata, which each sysadmin then makes a decision to trust. The result is that the IdP can confidently make assertions, and the SP can then receive these assertions.



**Additional resources**

- Federate with Identity Service