OpenStack Security Guide

current (2015-05-01)

This book provides best practices and conceptual information about securing an OpenStack cloud.

Table of Contents

List of Figures

Preface

Conventions

The OpenStack documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note

A handy tip or reminder.

Important

Something you must be aware of before proceeding.

Warning

Critical information about the risk of data loss or security issues.

Command prompts

$ prompt Any user, including the `root` user, can run commands that are prefixed with the $ prompt.

 # prompt The `root` user must run commands that are prefixed with the # prompt. You can also prefix these commands with the **sudo** command, if available, to run them.

Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

Revision Date	Summary of Changes
April 29, 2015	• Final prep for Kilo release.
February 11, 2015	• Chapter on Data processing added.
October 16, 2014	• This book has been extensively reviewed and updated. Chapters have been rearranged and a glossary has been added.
December 2, 2013	• Chapter on Object Storage added.
October 17, 2013	• Havana release.
July 2, 2013	• Initial creation...

1. Introduction

The OpenStack Security Guide is the result of a five day sprint of collaborative work of many individuals. The purpose of this document is to provide the best practice guidelines for deploying a secure OpenStack cloud. It is a living document that is updated as new changes are merged into the repository, and is meant to reflect the current state of security within the OpenStack community and provide frameworks for decision making where listing specific security controls are not feasible due to complexity or other environment specific details.

Acknowledgments

The OpenStack Security Group would like to acknowledge contributions from the following organizations that were instrumental in making this book possible. The organizations are:

Why and how we wrote this book

As OpenStack adoption continues to grow and the product matures, security has become a priority. The OpenStack Security Group has recognized the need for a comprehensive and authoritative security guide. The **Open-**

Stack Security Guide has been written to provide an overview of security best practices, guidelines, and recommendations for increasing the security of an OpenStack deployment. The authors bring their expertise from deploying and securing OpenStack in a variety of environments.

This guide augments the *OpenStack Operations Guide* and can be referenced to harden existing OpenStack deployments or to evaluate the security controls of OpenStack cloud providers.

Objectives

- Identify the security domains in OpenStack
- Provide guidance to secure your OpenStack deployment
- Highlight security concerns and potential mitigations in present day OpenStack
- Discuss upcoming security features
- To provide a community driven facility for knowledge capture and dissemination

How

As with the OpenStack Operations Guide, we followed the book sprint methodology. The book sprint process allows for rapid development and production of large bodies of written work. Coordinators from the OpenStack Security Group re-enlisted the services of Adam Hyde as facilitator. Corporate support was obtained and the project was formally announced during the OpenStack summit in Portland, Oregon.

The team converged in Annapolis, MD due to the close proximity of some key members of the group. This was a remarkable collaboration between public sector intelligence community members, silicon valley startups and some large, well-known technology companies. The book sprint ran during the last week in June 2013 and the first edition was created in five days.

The team included:

- **Bryan D. Payne**, Nebula

  Dr. Bryan D. Payne is the Director of Security Research at Nebula and co-founder of the OpenStack Security Group (OSSG). Prior to joining Nebula, he worked at Sandia National Labs, the National Security Agency, BAE Systems, and IBM Research. He graduated with a Ph.D. in Computer Science from the Georgia Tech College of Computing, specializing in systems security.

- **Robert Clark**, HP

  Robert Clark is the Lead Security Architect for HP Cloud Services and co-founder of the OpenStack Security Group (OSSG). Prior to being recruited by HP, he worked in the UK Intelligence Community. Robert has a strong background in threat modeling, security architecture and virtualization technology. Robert has a master's degree in Software Engineering from the University of Wales.

- **Keith Basil**, Red Hat

  Keith Basil is a Principal Product Manager for Red Hat OpenStack and is focused on Red Hat's OpenStack product management, development and strategy. Within the US public sector, Basil brings previous experience from the design of an authorized, secure, high-performance cloud architecture for Federal civilian agencies and contractors.

- **Cody Bunch**, Rackspace

Cody Bunch is a Private Cloud architect with Rackspace. Cody has co-authored an update to "The OpenStack Cookbook" as well as books on VMware automation.

- **Malini Bhandaru**, Intel

  Malini Bhandaru is a security architect at Intel. She has a varied background, having worked on platform power and performance at Intel, speech products at Nuance, remote monitoring and management at ComBrio, and web commerce at Verizon. She has a Ph.D. in Artificial Intelligence from the University of Massachusetts, Amherst.

- **Gregg Tally**, Johns Hopkins University Applied Physics Laboratory

  Gregg Tally is the Chief Engineer at JHU/APL's Cyber Systems Group within the Asymmetric Operations Department. He works primarily in systems security engineering. Previously, he has worked at SPARTA, McAfee, and Trusted Information Systems where he was involved in cyber security research projects.

- **Eric Lopez**, VMware

  Eric Lopez is Senior Solution Architect at VMware's Networking and Security Business Unit where he helps customers implement OpenStack and VMware NSX (formerly known as Nicira's Network Virtualization Platform). Prior to joining VMware (through the company's acquisition of Nicira), he worked for Q1 Labs, Symantec, Vontu, and Brightmail. He has a B.S in Electrical Engineering/Computer Science and Nuclear Engineering from U.C. Berkeley and MBA from the University of San Francisco.

- **Shawn Wells**, Red Hat

  Shawn Wells is the Director, Innovation Programs at Red Hat, focused on improving the process of adopting, contributing to, and managing open source technologies within the U.S. Government. Additionally, Shawn is an upstream maintainer of the SCAP Security Guide project which forms virtualization and operating system hardening policy with the U.S. Military, NSA, and DISA. Formerly an NSA civilian, Shawn developed SIGINT collection systems utilizing large distributed computing infrastructures.

- **Ben de Bont**, HP

Ben de Bont is the CSO for HP Cloud Services. Prior to his current role Ben led the information security group at MySpace and the incident response team at MSN Security. Ben holds a master's degree in Computer Science from the Queensland University of Technology.

- **Nathanael Burton**, National Security Agency

  Nathanael Burton is a Computer Scientist at the National Security Agency. He has worked for the Agency for over 10 years working on distributed systems, large-scale hosting, open source initiatives, operating systems, security, storage, and virtualization technology. He has a B.S. in Computer Science from Virginia Tech.

- **Vibha Fauver**

  Vibha Fauver, GWEB, CISSP, PMP, has over fifteen years of experience in Information Technology. Her areas of specialization include software engineering, project management and information security. She has a B.S. in Computer & Information Science and a M.S. in Engineering Management with specialization and a certificate in Systems Engineering.

- **Eric Windisch**, Cloudscaling

  Eric Windisch is a Principal Engineer at Cloudscaling where he has been contributing to OpenStack for over two years. Eric has been in the trenches of hostile environments, building tenant isolation and infrastructure security through more than a decade of experience in the web hosting industry. He has been building cloud computing infrastructure and automation since 2007.

- **Andrew Hay**, CloudPassage

  Andrew Hay is the Director of Applied Security Research at CloudPassage, Inc. where he leads the security research efforts for the company and its server security products purpose-built for dynamic public, private, and hybrid cloud hosting environments.

- **Adam Hyde**

  Adam facilitated this Book Sprint. He also founded the Book Sprint methodology and is the most experienced Book Sprint facilitator around. Adam founded FLOSS Manuals—a community of some 3,000 individuals developing Free Manuals about Free Software. He is also the founder and project manager for Booktype, an open source project for writing, editing, and publishing books online and in print.

During the sprint we also had help from Anne Gentle, Warren Wang, Paul McMillan, Brian Schott and Lorin Hochstein.

This Book was produced in a 5 day book sprint. A book sprint is an intensely collaborative, facilitated process which brings together a group to produce a book in 3-5 days. It is a strongly facilitated process with a specific methodology founded and developed by Adam Hyde. For more information visit the book sprint web page at http://www.booksprints.net.

After initial publication, the following added new content:

- **Rodney D. Beede**, Seagate Technology

  Rodney D. Beede is the Cloud Security Engineer for Seagate Technology. He contributed the missing chapter on securing OpenStack Object Storage (swift). He holds a M.S. in Computer Science from the University of Colorado.

How to contribute to this book

The initial work on this book was conducted in an overly air-conditioned room that served as our group office for the entirety of the documentation sprint.

Learn more about how to contribute to the OpenStack docs: http://wiki.openstack.org/Documentation/HowTo.

Introduction to OpenStack

This guide provides security insight into *OpenStack* deployments. The intended audience is cloud architects, deployers, and administrators. In addition, cloud users will find the guide both educational and helpful in provider selection, while auditors will find it useful as a reference document to support their compliance certification efforts. This guide is also recommended for anyone interested in cloud security.

Each OpenStack deployment embraces a wide variety of technologies, spanning Linux distributions, database systems, messaging queues, OpenStack components themselves, access control policies, logging services, security monitoring tools, and much more. It should come as no surprise that the security issues involved are equally diverse, and their in-depth analysis would require several guides. We strive to find a balance, providing enough context to understand OpenStack security issues and their handling, and provide external references for further information. The guide could be read from start to finish or sampled as necessary like a reference.

We briefly introduce the kinds of clouds: private, public, and hybrid before presenting an overview of the OpenStack components and their related security concerns in the remainder of the chapter.

Cloud types

OpenStack is a key enabler in adoption of cloud technology and has several common deployment use cases. These are commonly known as Public, Private, and Hybrid models. The following sections use the National Institute of Standards and Technology (NIST) definition of cloud to introduce these different types of cloud as they apply to OpenStack.

Public cloud

According to NIST, a public cloud is one in which the infrastructure is open to the general public for consumption. OpenStack public clouds are typically run by a service provider and can be consumed by individuals, corporations, or any paying customer. A public cloud provider may expose a full set of features such as software-defined networking, block storage, in addition to multiple instance types. Due to the nature of public clouds, they are exposed to a higher degree of risk. As a consumer of a public cloud you should validate that your selected provider has the necessary certifications, attestations, and other regulatory considerations. As a public cloud provider, depending on your target customers, you may be subject to one or more regulations. Additionally, even if not required to meet regulatory requirements, a provider should ensure tenant isolation as well as protecting management infrastructure from external attacks.

Private cloud

At the opposite end of the spectrum is the private cloud. As NIST defines it, a private cloud is provisioned for exclusive use by a single organization comprising multiple consumers, such as business units. It may be owned, managed, and operated by the organization, a third-party, or some combination of them, and it may exist on or off premises. Private cloud use cases are diverse, as such, their individual security concerns vary.

Community cloud

NIST defines a community cloud as one whose infrastructure is provisioned for the exclusive use by a specific community of consumers from organizations that have shared concerns. For example, mission, security requirements, policy, and compliance considerations. It may be owned, managed, and operated by one or more of the organizations in the communi-

ty, a third-party, or some combination of them, and it may exist on or off premises.

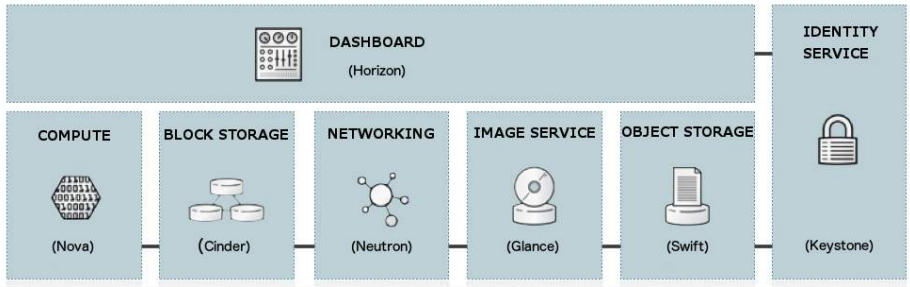
Hybrid cloud

A hybrid cloud is defined by NIST as a composition of two or more distinct cloud infrastructures, such as private, community, or public, that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability, such as cloud bursting for load balancing between clouds. For example an online retailer may have their advertising and catalogue presented on a public cloud that allows for elastic provisioning. This would enable them to handle seasonal loads in a flexible, cost-effective fashion. Once a customer begins to process their order, they are transferred to the more secure private cloud back end that is PCI compliant.

For the purposes of this document, we treat Community and Hybrid similarly, dealing explicitly only with the extremes of Public and Private clouds from a security perspective. Your security measures depend where your deployment falls upon the private public continuum.

OpenStack service overview

OpenStack embraces a modular architecture to provide a set of core services that facilitates scalability and elasticity as core design tenets. This chapter briefly reviews OpenStack components, their use cases and security considerations.



Compute

OpenStack *Compute* service (*nova*) provides services to support the management of virtual machine instances at scale, instances that host multi-tiered applications, dev/test environments, "Big Data" crunching Hadoop clusters, and/or high performance computing.

The Compute service facilitates this management through an abstraction layer that interfaces with supported hypervisors, which we address later on in more detail.

Later in the guide, we focus generically on the virtualization stack as it relates to hypervisors.

For information about the current state of feature support, see OpenStack Hypervisor Support Matrix.

The security of Compute is critical for an OpenStack deployment. Hardening techniques should include support for strong instance isolation, secure communication between Compute sub-components, and resiliency of public-facing *API* endpoints.

Object Storage

The OpenStack *Object Storage* service (*swift*) provides support for storing and retrieving arbitrary data in the cloud. The Object Storage service provides both a native API and an Amazon Web Services S3 compatible API. The service provides a high degree of resiliency through data replication and can handle petabytes of data.

It is important to understand that object storage differs from traditional file system storage. It is best used for static data such as media files (MP3s, images, videos), virtual machine images, and backup files.

Object security should focus on access control and encryption of data in transit and at rest. Other concerns may relate to system abuse, illegal or malicious content storage, and cross authentication attack vectors.

Block Storage

The OpenStack *Block Storage* service (*cinder*) provides persistent block storage for compute instances. The Block Storage service is responsible for managing the life-cycle of block devices, from the creation and attachment of volumes to instances, to their release.

Security considerations for block storage are similar to that of object storage.

Networking

The OpenStack *Networking* service (*neutron*, previously called quantum) provides various networking services to cloud users (tenants) such as IP ad-

dress management, *DNS*, *DHCP*, load balancing, and security groups (network access rules, like firewall policies). It provides a framework for software defined networking (SDN) that allows for pluggable integration with various networking solutions.

OpenStack Networking allows cloud tenants to manage their guest network configurations. Security concerns with the networking service include network traffic isolation, availability, integrity and confidentiality.

Dashboard

The OpenStack *dashboard* (*horizon*) provides a web-based interface for both cloud administrators and cloud tenants. Through this interface administrators and tenants can provision, manage, and monitor cloud resources. Horizon is commonly deployed in a public facing manner with all the usual security concerns of public web portals.

Identity service

The OpenStack *Identity* service (*keystone*) is a **shared service** that provides authentication and authorization services throughout the entire cloud infrastructure. The Identity service has pluggable support for multiple forms of authentication.

Security concerns here pertain to trust in authentication, management of authorization tokens, and secure communication.

Image service

The OpenStack *Image service* (*glance*) provides disk image management services. The Image service provides image discovery, registration, and delivery services to the Compute service, as needed.

Trusted processes for managing the life cycle of disk images are required, as are all the previously mentioned issues with respect to data security.

Data processing service

The *Data processing service* for OpenStack (*sahara*) provides a platform for the provisioning, management, and usage of clusters running popular processing frameworks.

Security considerations for data processing should focus on data privacy and secure communications to provisioned clusters.

Other supporting technology

OpenStack relies on messaging for internal communication between several of its services. By default, OpenStack uses message queues based on the Advanced Message Queue Protocol (*AMQP*). Similar to most OpenStack services, it supports pluggable components. Today the implementation back end could be *RabbitMQ*, *Qpid*, or *ZeroMQ*.

As most management commands flow through the message queuing system, it is a primary security concern for any OpenStack deployment. Message queuing security is discussed in detail later in this guide.

Several of the components use databases though it is not explicitly called out. Securing the access to the databases and their contents is yet another security concern, and consequently discussed in more detail later in this guide.

Security boundaries and threats

A cloud can be abstracted as a collection of logical components by virtue of their function, users, and shared security concerns, which we call security domains. Threat actors and vectors are classified based on their motivation and access to resources. Our goal is to provide you a sense of the security concerns with respect to each domain depending on your risk/vulnerability protection objectives.

Security domains

A security domain comprises users, applications, servers or networks that share common trust requirements and expectations within a system. Typically they have the same authentication and authorization (AuthN/Z) requirements and users.

Although you may desire to break these domains down further (we later discuss where this may be appropriate), we generally refer to four distinct security domains which form the bare minimum that is required to deploy any OpenStack cloud securely. These security domains are:

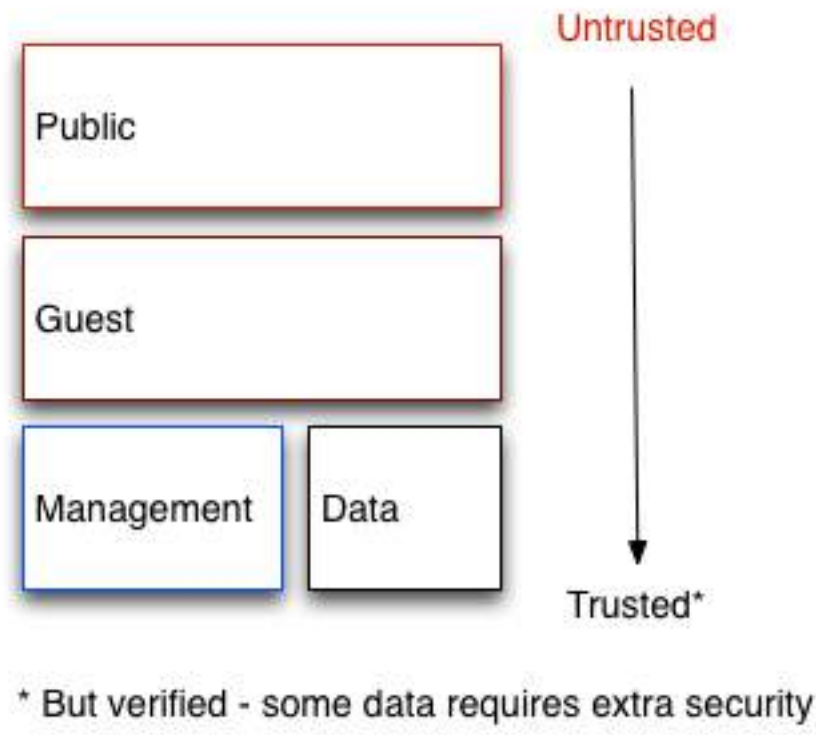
1. Public

2. Guest

3. Management

4. Data

We selected these security domains because they can be mapped independently or combined to represent the majority of the possible areas of trust within a given OpenStack deployment. For example, some deployment topologies may consist of a combination of guest and data domains onto one physical network while other topologies have these domains separated. In each case, the cloud operator should be aware of the appropriate security concerns. Security domains should be mapped out against your specific OpenStack deployment topology. The domains and their trust requirements depend upon whether the cloud instance is public, private, or hybrid.



Public

The public security domain is an entirely untrusted area of the cloud infrastructure. It can refer to the Internet as a whole or simply to networks over which you have no authority. Any data that transits this domain with confi-

dentiality or integrity requirements should be protected using compensating controls.

This domain should always be considered *untrusted*.

Guest

Typically used for compute instance-to-instance traffic, the guest security domain handles compute data generated by instances on the cloud but not services that support the operation of the cloud, such as API calls.

Public and private cloud providers that do not have stringent controls on instance use or allow unrestricted internet access to VMs should consider this domain to be *untrusted*. Private cloud providers may want to consider this network as internal and *trusted*, only if the proper controls are implemented to assert that the instances and all associated tenants are to be trusted.

Management

The management security domain is where services interact. Sometimes referred to as the "control plane", the networks in this domain transport confidential data such as configuration parameters, user names, and passwords. Command and Control traffic typically resides in this domain, which necessitates strong integrity requirements. Access to this domain should be highly restricted and monitored. At the same time, this domain should still employ all of the security best practices described in this guide.

In most deployments this domain is considered *trusted*. However, when considering an OpenStack deployment, there are many systems that bridge this domain with others, potentially reducing the level of trust you can place on this domain. See the section called “Bridging security domains” [15] for more information.

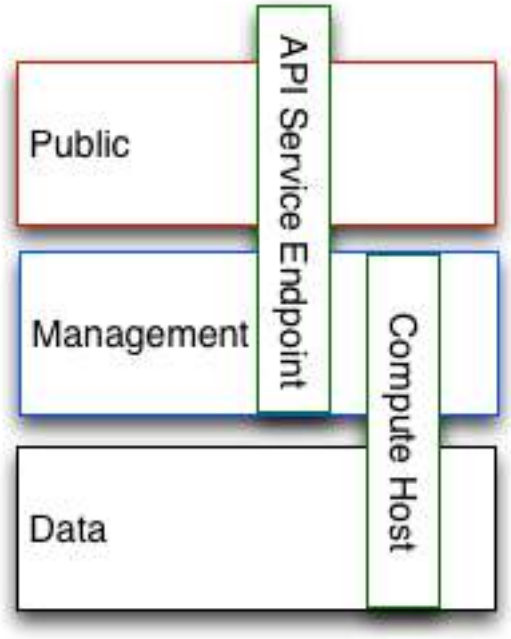
Data

The data security domain is concerned primarily with information pertaining to the storage services within OpenStack. Most of the data transmitted across this network requires high levels of integrity and confidentiality. In some cases, depending on the type of deployment there may also be strong availability requirements.

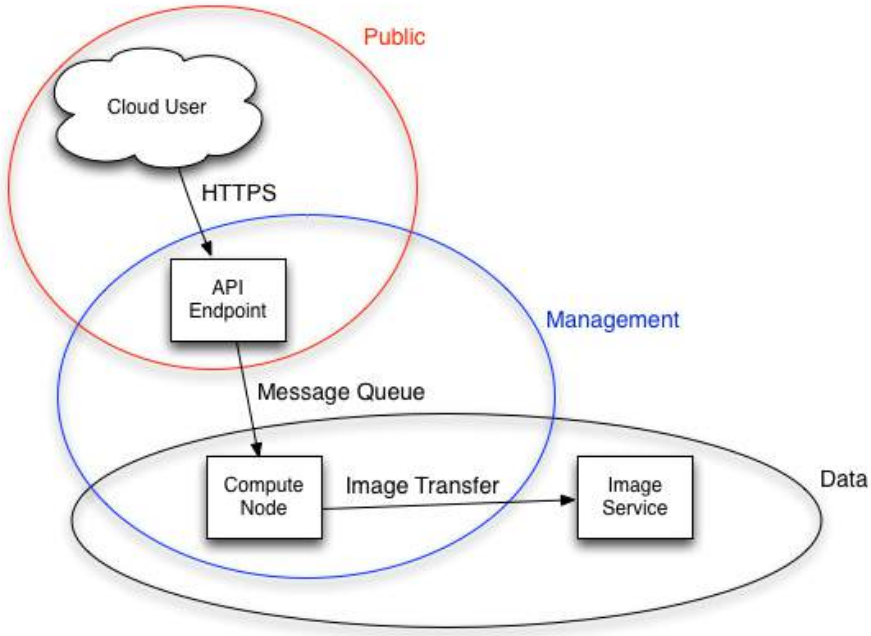
The trust level of this network is heavily dependent on deployment decisions and as such we do not assign this any default level of trust.

Bridging security domains

A *bridge* is a component that exists inside more than one security domain. Any component that bridges security domains with different trust levels or authentication requirements must be carefully configured. These bridges are often the weak points in network architecture. A bridge should always be configured to meet the security requirements of the highest trust level of any of the domains it is bridging. In many cases the security controls for bridges should be a primary concern due to the likelihood of attack.



The diagram above shows a compute node bridging the data and management domains, as such the compute node should be configured to meet the security requirements of the management domain. Similarly, the API Endpoint in this diagram is bridging the untrusted public domain and the management domain, which should be configured to protect against attacks from the public domain propagating through to the management domain.



In some cases deployers may want to consider securing a bridge to a higher standard than any of the domains in which it resides. Given the above example of an API endpoint, an adversary could potentially target the API endpoint from the public domain, leveraging it in the hopes of compromising or gaining access to the management domain.

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

Threat classification, actors and attack vectors

Most types of cloud deployment, public or private, are exposed to some form of attack. In this chapter we categorize attackers and summarize potential types of attacks in each security domain.

Threat actors

A threat actor is an abstract way to refer to a class of adversary that you may attempt to defend against. The more capable the actor, the more expensive the security controls that are required for successful attack mitigation and prevention. Security is a tradeoff between cost, usability and de-

fense. In some cases it will not be possible to secure a cloud deployment against all of the threat actors we describe here. Those deploying an OpenStack cloud will have to decide where the balance lies for their deployment / usage.

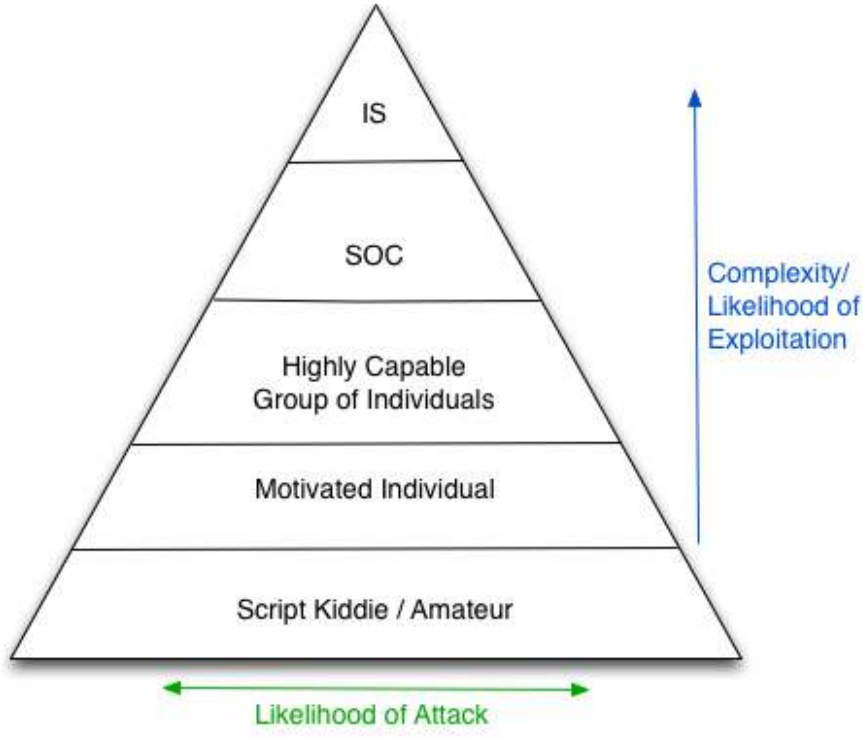
- **Intelligence services** — Considered by this guide as the most capable adversary. Intelligence Services and other state actors can bring tremendous resources to bear on a target. They have capabilities beyond that of any other actor. It is very difficult to defend against these actors without incredibly stringent controls in place, both human and technical.

- **Serious organized crime** — Highly capable and financially driven groups of attackers. Able to fund in-house exploit development and target research. In recent years the rise of organizations such as the Russian Business Network, a massive cyber-criminal enterprise has demonstrated how cyber attacks have become a commodity. Industrial espionage falls within the serious organized crime group.

- **Highly capable groups** — This refers to 'Hacktivist' type organizations who are not typically commercially funded but can pose a serious threat to service providers and cloud operators.

- **Motivated individuals** — Acting alone, these attackers come in many guises, such as rogue or malicious employees, disaffected customers, or small-scale industrial espionage.

- **Script kiddies** — Automated vulnerability scanning/exploitation. Non-targeted attacks. Often only a nuisance, compromise by one of these actors presents a major risk to an organization's reputation.



Public and private cloud considerations

Private clouds are typically deployed by enterprises or institutions inside their networks and behind their firewalls. Enterprises will have strict policies on what data is allowed to exit their network and may even have different clouds for specific purposes. Users of a private cloud are typically employees of the organization that owns the cloud and are able to be held accountable for their actions. Employees often attend training sessions before accessing the cloud and will likely take part in regular scheduled security awareness training. Public clouds by contrast cannot make any assertions about their users, cloud use-cases or user motivations. This immediately pushes the guest security domain into a completely *untrusted* state for public cloud providers.

A notable difference in the attack surface of public clouds is that they must provide internet access to their services. Instance connectivity, access to files over the internet and the ability to interact with the cloud controlling fabric such as the API endpoints and dashboard are must-haves for the public cloud.

Privacy concerns for public and private cloud users are typically diametrically opposed. The data generated and stored in private clouds is normally owned by the operator of the cloud, who is able to deploy technologies such as data loss prevention (DLP) protection, file inspection, deep packet inspection and prescriptive firewalling. In contrast, privacy is one of the primary barriers for the adoption of public cloud infrastructures, as many of the previously mentioned controls do not exist.

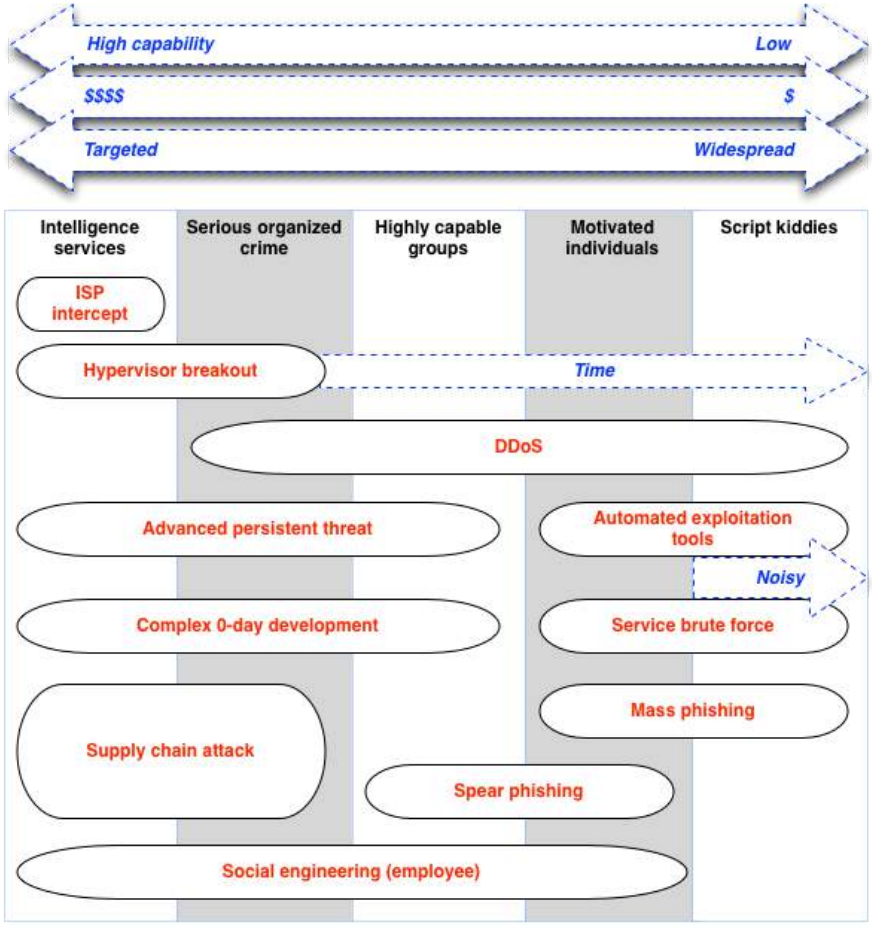
Outbound attacks and reputational risk

Careful consideration should be given to potential outbound abuse from a cloud deployment. Whether public or private, clouds tend to have lots of resource available. An attacker who has established a point of presence within the cloud, either through hacking or entitled access, such as rogue employee, can bring these resources to bear against the internet at large. Clouds with compute services make for ideal DDoS and brute force engines. The issue is more pressing for public clouds as their users are largely unaccountable, and can quickly spin up numerous disposable instances for outbound attacks. Major damage can be inflicted upon a company's reputation if it becomes known for hosting malicious software or launching attacks on other networks. Methods of prevention include egress security groups, outbound traffic inspection, customer education and awareness, and fraud and abuse mitigation strategies.

Attack types

The diagram shows the types of attacks that may be expected from the actors described in the previous section. Note that there will always be exceptions to this diagram but in general, this describes the sorts of attack that could be typical for each actor.

Figure 1.1. Attack types



The prescriptive defense for each form of attack is beyond the scope of this document. The above diagram can assist you in making an informed decision about which types of threats, and threat actors, should be protected against. For commercial public cloud deployments this might include prevention against serious crime. For those deploying private clouds for government use, more stringent protective mechanisms should be in place, including carefully protected facilities and supply chains. In contrast those standing up basic development or test environments will likely require less restrictive controls (middle of the spectrum).

Introduction to case studies

This guide refers to two running case studies, which are introduced here and referred to at the end of each chapter.

Case study: Alice, the private cloud builder

Alice deploys a private cloud for use by a government department in the US. The cloud must comply with relevant standards, such as FedRAMP. The security paperwork requirements for this cloud are very high. It must have no direct access to the internet: its API endpoints, compute instances, and other resources must be exposed to only systems within the department's network, which is entirely air-gapped from all other networks. The cloud can access other network services on the organization's intranet such as the authentication and logging services.

Case study: Bob, the public cloud provider

Bob is a lead architect for a company that deploys a large greenfield public cloud. This cloud provides IaaS for the masses and enables any consumer with a valid credit card access to utility computing and storage, but the primary focus is enterprise customers. Data privacy concerns are a big priority for Bob as they are seen as a major barrier to large-scale adoption of the cloud by organizations.

2. System documentation

The system documentation for an OpenStack cloud deployment should follow the templates and best practices for the Enterprise Information Technology System in your organization. Organizations often have compliance requirements which may require an overall System Security Plan to inventory and document the architecture of a given system. There are common challenges across the industry related to documenting the dynamic cloud infrastructure and keeping the information up-to-date.

System documentation requirements

System roles and types

The two broadly defined types of nodes that generally make up an OpenStack installation are:

- Infrastructure nodes. The nodes that run the cloud related services such as the OpenStack Identity service, the message queuing service, storage, networking, and other services required to support the operation of the cloud.
- Compute, storage, or other resource nodes. Provide storage capacity or virtual machines for your cloud.

System inventory

Documentation should provide a general description of the OpenStack environment and cover all systems used (production, development, test, etc.). Documenting system components, networks, services, and software often provides the bird's-eye view needed to thoroughly cover and consider security concerns, attack vectors and possible security domain bridging points. A system inventory may need to capture ephemeral resources such as virtual machines or virtual disk volumes that would otherwise be persistent resources in a traditional IT system.

Hardware inventory

Clouds without stringent compliance requirements for written documentation might benefit from having a Configuration Management Database

(*CMDB*). CMDBs are normally used for hardware asset tracking and overall life-cycle management. By leveraging a CMDB, an organization can quickly identify cloud infrastructure hardware such as compute nodes, storage nodes, or network devices. A CMDB can assist in identifying assets that exist on the network which may have vulnerabilities due to inadequate maintenance, inadequate protection or being displaced and forgotten. An OpenStack provisioning system can provide some basic CMDB functions if the underlying hardware supports the necessary auto-discovery features.

Software inventory

As with hardware, all software components within the OpenStack deployment should be documented. Examples include:

- System databases, such as MySQL or mongoDB
- OpenStack software components, such as Identity or Compute
- Supporting components, such as load-balancers, reverse proxies, DNS or DHCP services

An authoritative list of software components may be critical when assessing the impact of a compromise or vulnerability in a library, application or class of software.

Network topology

A network topology should be provided with highlights specifically calling out the data flows and bridging points between the security domains. Network ingress and egress points should be identified along with any OpenStack logical system boundaries. Multiple diagrams may be needed to provide complete visual coverage of the system. A network topology document should include virtual networks created on behalf of tenants by the system along with virtual machine instances and gateways created by OpenStack.

Services, protocols and ports

Knowing information about organizational assets is typically a best practice, therefore it is beneficial to create a table which contains information regarding the service, protocols and ports being utilized in the OpenStack deployment. The table can be created from information derived from a CMDB or can be constructed manually. The table can be customized to include an overview of all services running within the cloud infrastructure.

The level of detail contained in this type of table can be beneficial as the information can immediately inform, guide, and assist with validating security requirements. Standard security components such as firewall configuration, service port conflicts, security remediation areas, and compliance become easier to maintain when concise information is available. An example of this type of table is provided below:

Service	Protocols	Ports	Purpose	Used By	Security Domain(s)
beam.smp	AMQP	tcp/5672	AMQP message service	RabbitMQ	MGMT
tgtd	iSCSI	tcp/3260	iSCSI initiator service	iSCSI	PRIVATE (data network)
sshd	ssh	tcp/22	allows secure login to nodes and guest VMs	Various	MGMT, GUEST and PUBLIC as configured
mysqld	mysql	tcp/3306	MySQL database service	Various	MGMT
apache2	http	tcp/443	Horizon dashboard service	Tenants	PUBLIC
dnsmasq	dns	tcp/53	DNS services	Guest VMs	GUEST

Referencing a table of services, protocols and ports can help in understanding the relationship between OpenStack components. It is highly recommended that OpenStack deployments have information similar to this on record.

Case studies

Earlier in the section called "Introduction to case studies" [21] we introduced the Alice and Bob case studies where Alice is deploying a government cloud and Bob is deploying a public cloud each with different security requirements. Here we discuss how Alice and Bob would address their system documentation requirements. The documentation suggested above includes hardware and software records, network diagrams, and system configuration details.

Alice's private cloud

Alice needs detailed documentation to satisfy FedRAMP requirements. She sets up a configuration management database (CMDB) to store information regarding all of the hardware, firmware, and software versions used throughout the cloud. She also creates a network diagram detailing the cloud architecture, paying careful attention to the security domains and the services that span multiple security domains.

Alice also needs to record each network service running in the cloud, what interfaces and ports it binds to, the security domains for each service, and why the service is needed. Alice decides to build automated tools to log into each system in the cloud over secure shell (SSH) using the Python Fabric library. The tools collect and store the information in the CMDB, which simplifies the audit process.

Bob's public cloud

In this case, Bob will approach these steps the same as Alice.

3. Management

A cloud deployment is a living system. Machines age and fail, software becomes outdated, vulnerabilities are discovered. When errors or omissions are made in configuration, or when software fixes must be applied, these changes must be made in a secure, but convenient, fashion. These changes are typically solved through configuration management.

Likewise, it is important to protect the cloud deployment from being configured or manipulated by malicious entities. With many systems in a cloud employing compute and networking virtualization, there are distinct challenges applicable to OpenStack which must be addressed through integrity lifecycle management.

Finally, administrators must perform command and control over the cloud for various operational functions. It is important these command and control facilities are understood and secured.

Continuous systems management

A cloud will always have bugs. Some of these will be security problems. For this reason, it is critically important to be prepared to apply security updates and general software updates. This involves smart use of configuration management tools, which are discussed below. This also involves knowing when an upgrade is necessary.

Vulnerability management

For announcements regarding security relevant changes, subscribe to the OpenStack Announce mailing list. The security notifications are also posted through the downstream packages, for example, through Linux distributions that you may be subscribed to as part of the package updates.

The OpenStack components are only a small fraction of the software in a cloud. It is important to keep up to date with all of these other components, too. While certain data sources will be deployment specific, it is important that a cloud administrator subscribe to the necessary mailing lists in order to receive notification of any security updates applicable to the

organization's environment. Often this is as simple as tracking an upstream Linux distribution.

Note

OpenStack releases security information through two channels.

- OpenStack Security Advisories (OSSA) are created by the OpenStack Vulnerability Management Team (VMT). They pertain to security holes in core OpenStack services. More information on the VMT can be found here: https://wiki.openstack.org/wiki/Vulnerability_Management
- OpenStack Security Notes (OSSN) are created by the OpenStack Security Group (OSSG) to support the work of the VMT. OSSN address issues in supporting software and common deployment configurations. They are referenced throughout this guide. Security Notes are archived at https://launchpad.net/ossn/

Triage

After you are notified of a security update, the next step is to determine how critical this update is to a given cloud deployment. In this case, it is useful to have a pre-defined policy. Existing vulnerability rating systems such as the common vulnerability scoring system (CVSS) v2 do not properly account for cloud deployments.

In this example we introduce a scoring matrix that places vulnerabilities in three categories: Privilege Escalation, Denial of Service and Information Disclosure. Understanding the type of vulnerability and where it occurs in your infrastructure will enable you to make reasoned response decisions.

Privilege Escalation describes the ability of a user to act with the privileges of some other user in a system, bypassing appropriate authorization checks. A guest user performing an operation that allows them to conduct unauthorized operations with the privileges of an administrator is an example of this type of vulnerability.

Denial of Service refers to an exploited vulnerability that may cause service or system disruption. This includes both distributed attacks to overwhelm network resources, and single-user attacks that are typically caused through resource allocation bugs or input induced system failure flaws.

Information Disclosure vulnerabilities reveal information about your system or operations. These vulnerabilities range from debugging informa-

tion disclosure, to exposure of critical security data, such as authentication credentials and passwords.

	Attacker position / Privilege level			
	External	**Cloud user**	**Cloud admin**	**Control plane**
Privilege elevation (3 levels)	Critical	n/a	n/a	n/a
Privilege elevation (2 levels)	Critical	Critical	n/a	n/a
Privilege elevation (1 level)	Critical	Critical	Critical	n/a
Denial of service	High	Medium	Low	Low
Information disclosure	Critical / high	Critical / high	Medium / low	Low

This table illustrates a generic approach to measuring the impact of a vulnerability based on where it occurs in your deployment and the effect. For example, a single level privilege escalation on a Compute API node potentially allows a standard user of the API to escalate to have the same privileges as the root user on the node.

We suggest that cloud administrators use this table as a model to help define which actions to take for the various security levels. For example, a critical-level security update might require the cloud to be upgraded quickly whereas a low-level update might take longer to be completed.

Testing the updates

You should test any update before you deploy it in a production environment. Typically this requires having a separate test cloud setup that first receives the update. This cloud should be as close to the production cloud as possible, in terms of software and hardware. Updates should be tested thoroughly in terms of performance impact, stability, application impact, and more. Especially important is to verify that the problem theoretically addressed by the update, such as a specific vulnerability, is actually fixed.

Deploying the updates

Once the updates are fully tested, they can be deployed to the production environment. This deployment should be fully automated using the configuration management tools described below.

Configuration management

A production quality cloud should always use tools to automate configuration and deployment. This eliminates human error, and allows the cloud to scale much more rapidly. Automation also helps with continuous integration and testing.

When building an OpenStack cloud it is strongly recommended to approach your design and implementation with a configuration management tool or framework in mind. Configuration management allows you to avoid the many pitfalls inherent in building, managing, and maintaining an infrastructure as complex as OpenStack. By producing the manifests, cookbooks, or templates required for a configuration management utility, you are able to satisfy a number of documentation and regulatory reporting requirements. Further, configuration management can also function as part of your business continuity plan (BCP) and data recovery (DR) plans wherein you can rebuild a node or service back to a known state in a DR event or given a compromise.

Additionally, when combined with a version control system such as Git or SVN, you can track changes to your environment over time and re-mediate unauthorized changes that may occur. For example, a `nova.conf` file or other configuration file falls out of compliance with your standard, your configuration management tool can revert or replace the file and bring your configuration back into a known state. Finally a configuration management tool can also be used to deploy updates; simplifying the security patch process. These tools have a broad range of capabilities that are useful in this space. The key point for securing your cloud is to choose a tool for configuration management and use it.

There are many configuration management solutions; at the time of this writing there are two in the marketplace that are robust in their support of OpenStack environments: *Chef* and *Puppet*. A non-exhaustive listing of tools in this space is provided below:

- Chef
- Puppet
- Salt Stack
- Ansible

Policy changes

Whenever a policy or configuration management is changed, it is good practice to log the activity, and backup a copy of the new set. Often, such policies and configurations are stored in a version controlled repository such as Git.

Secure backup and recovery

It is important to include Backup procedures and policies in the overall System Security Plan. For a good overview of OpenStack's Backup and Recovery capabilities and procedures, please refer to the OpenStack Operations Guide.

Security considerations

- Ensure only authenticated users and backup clients have access to the backup server.
- Use data encryption options for storage and transmission of backups.
- Use a dedicated and hardened backup servers. The logs for the backup server must be monitored daily and accessible by only few individuals.
- Test data recovery options regularly. One of the things that can be restored from secured backups is the images. In case of a compromise, the best practice would be to terminate running instances immediately and then relaunch the instances from the images in the secured backup repository.

References

- *OpenStack Operations Guide* on backup and recovery
- http://www.sans.org/reading_room/whitepapers/backup/security-considerations-enterprise-level-backups_515
- OpenStack Security Primer, an entry in the music piracy blog by a former member of the original NASA project team that created nova

Security auditing tools

Security auditing tools can complement the configuration management tools. Security auditing tools automate the process of verifying that a large

number of security controls are satisfied for a given system configuration. These tools help to bridge the gap from security configuration guidance documentation (for example, the STIG and NSA Guides) to a specific system installation. For example, SCAP can compare a running system to a pre-defined profile. SCAP outputs a report detailing which controls in the profile were satisfied, which ones failed, and which ones were not checked.

Combining configuration management and security auditing tools creates a powerful combination. The auditing tools will highlight deployment concerns. And the configuration management tools simplify the process of changing each system to address the audit concerns. Used together in this fashion, these tools help to maintain a cloud that satisfies security requirements ranging from basic hardening to compliance validation.

Configuration management and security auditing tools will introduce another layer of complexity into the cloud. This complexity brings additional security concerns with it. We view this as an acceptable risk trade-off, given their security benefits. Securing the operational use of these tools is beyond the scope of this guide.

Integrity life-cycle

We define integrity life cycle as a deliberate process that provides assurance that we are always running the expected software with the expected configurations throughout the cloud. This process begins with secure bootstrapping and is maintained through configuration management and security monitoring. This chapter provides recommendations on how to approach the integrity life-cycle process.

Secure bootstrapping

Nodes in the cloud—including compute, storage, network, service, and hybrid nodes—should have an automated provisioning process. This ensures that nodes are provisioned consistently and correctly. This also facilitates security patching, upgrading, bug fixing, and other critical changes. Since this process installs new software that runs at the highest privilege levels in the cloud, it is important to verify that the correct software is installed. This includes the earliest stages of the boot process.

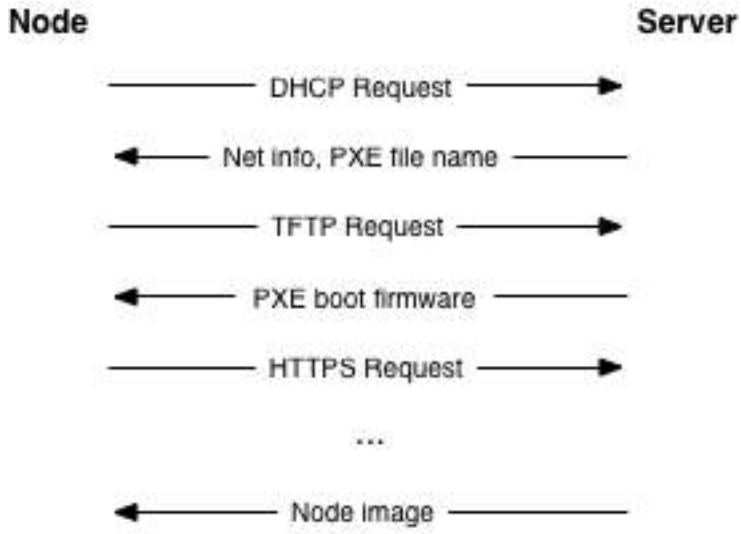
There are a variety of technologies that enable verification of these early boot stages. These typically require hardware support such as the trusted platform module (TPM), Intel Trusted Execution Technology (TXT), dynamic root of trust measurement (DRTM), and Unified Extensible Firmware In-

terface (UEFI) secure boot. In this book, we will refer to all of these collectively as *secure boot technologies*. We recommend using secure boot, while acknowledging that many of the pieces necessary to deploy this require advanced technical skills in order to customize the tools for each environment. Utilizing secure boot will require deeper integration and customization than many of the other recommendations in this guide. TPM technology, while common in most business class laptops and desktops for several years, and is now becoming available in servers together with supporting BIOS. Proper planning is essential to a successful secure boot deployment.

A complete tutorial on secure boot deployment is beyond the scope of this book. Instead, here we provide a framework for how to integrate secure boot technologies with the typical node provisioning process. For additional details, cloud architects should refer to the related specifications and software configuration manuals.

Node provisioning

Nodes should use Preboot eXecution Environment (PXE) for provisioning. This significantly reduces the effort required for redeploying nodes. The typical process involves the node receiving various boot stages—that is progressively more complex software to execute— from a server.



We recommend using a separate, isolated network within the management security domain for provisioning. This network will handle all PXE traffic, along with the subsequent boot stage downloads depicted above.

Note that the node boot process begins with two insecure operations: DHCP and TFTP. Then the boot process uses TLS to download the remaining information required to deploy the node. This may be an operating system installer, a basic install managed by Chef or Puppet, or even a complete file system image that is written directly to disk.

While utilizing TLS during the PXE boot process is somewhat more challenging, common PXE firmware projects, such as iPXE, provide this support. Typically this involves building the PXE firmware with knowledge of the allowed TLS certificate chain(s) so that it can properly validate the server certificate. This raises the bar for an attacker by limiting the number of insecure, plain text network operations.

Verified boot

In general, there are two different strategies for verifying the boot process. Traditional *secure boot* will validate the code run at each step in the process, and stop the boot if code is incorrect. *Boot attestation* will record which code is run at each step, and provide this information to another machine as proof that the boot process completed as expected. In both cases, the first step is to measure each piece of code before it is run. In this context, a measurement is effectively a SHA-1 hash of the code, taken before it is executed. The hash is stored in a platform configuration register (PCR) in the TPM.

Note: SHA-1 is used here because this is what the TPM chips support.

Each TPM has at least 24 PCRs. The TCG Generic Server Specification, v1.0, March 2005, defines the PCR assignments for boot-time integrity measurements. The table below shows a typical PCR configuration. The context indicates if the values are determined based on the node hardware (firmware) or the software provisioned onto the node. Some values are influenced by firmware versions, disk sizes, and other low-level information. Therefore, it is important to have good practices in place around configuration management to ensure that each system deployed is configured exactly as desired.

Register	What is measured	Context
PCR-00	Core Root of Trust Measurement (CRTM), BIOS code, Host platform extensions	Hardware
PCR-01	Host platform configuration	Hardware
PCR-02	Option ROM code	Hardware

PCR-03	Option ROM configuration and data	Hardware
PCR-04	Initial Program Loader (IPL) code. For example, master boot record.	Software
PCR-05	IPL code configuration and data	Software
PCR-06	State transition and wake events	Software
PCR-07	Host platform manufacturer control	Software
PCR-08	Platform specific, often kernel, kernel extensions, and drivers	Software
PCR-09	Platform specific, often Initramfs	Software
PCR-10 to PCR-23	Platform specific	Software

At the time of this writing, very few clouds are using secure boot technologies in a production environment. As a result, these technologies are still somewhat immature. We recommend planning carefully in terms of hardware selection. For example, ensure that you have a TPM and Intel TXT support. Then verify how the node hardware vendor populates the PCR values. For example, which values will be available for validation. Typically the PCR values listed under the software context in the table above are the ones that a cloud architect has direct control over. But even these may change as the software in the cloud is upgraded. Configuration management should be linked into the PCR policy engine to ensure that the validation is always up to date.

Each manufacturer must provide the BIOS and firmware code for their servers. Different servers, hypervisors, and operating systems will choose to populate different PCRs. In most real world deployments, it will be impossible to validate every PCR against a known good quantity ("golden measurement"). Experience has shown that, even within a single vendor's product line, the measurement process for a given PCR may not be consistent. We recommend establishing a baseline for each server and monitoring the PCR values for unexpected changes. Third-party software may be available to assist in the TPM provisioning and monitoring process, depending upon your chosen hypervisor solution.

The initial program loader (IPL) code will most likely be the PXE firmware, assuming the node deployment strategy outlined above. Therefore, the secure boot or boot attestation process can measure all of the early stage boot code, such as BIOS, firmware, the PXE firmware, and the kernel image. Ensuring that each node has the correct versions of these pieces in-

stalled provides a solid foundation on which to build the rest of the node software stack.

Depending on the strategy selected, in the event of a failure the node will either fail to boot or it can report the failure back to another entity in the cloud. For secure boot, the node will fail to boot and a provisioning service within the management security domain must recognize this and log the event. For boot attestation, the node will already be running when the failure is detected. In this case the node should be immediately quarantined by disabling its network access. Then the event should be analyzed for the root cause. In either case, policy should dictate how to proceed after a failure. A cloud may automatically attempt to re-provision a node a certain number of times. Or it may immediately notify a cloud administrator to investigate the problem. The right policy here will be deployment and failure mode specific.

Node hardening

At this point we know that the node has booted with the correct kernel and underlying components. There are many paths for hardening a given operating system deployment. The specifics on these steps are outside of the scope of this book. We recommend following the guidance from a hardening guide specific to your operating system. For example, the security technical implementation guides (STIG) and the NSA guides are useful starting places.

The nature of the nodes makes additional hardening possible. We recommend the following additional steps for production nodes:

- Use a read-only file system where possible. Ensure that writeable file systems do not permit execution. This can be handled through the mount options provided in `/etc/fstab`.
- Use a mandatory access control policy to contain the instances, the node services, and any other critical processes and data on the node. See the discussions on sVirt / SELinux and AppArmor below.
- Remove any unnecessary software packages. This should result in a very stripped down installation because a compute node has a relatively small number of dependencies.

Finally, the node kernel should have a mechanism to validate that the rest of the node starts in a known good state. This provides the necessary link from the boot validation process to validating the entire system. The steps for doing this will be deployment specific. As an example, a kernel mod-

ule could verify a hash over the blocks comprising the file system before mounting it using dm-verity.

Runtime verification

Once the node is running, we need to ensure that it remains in a good state over time. Broadly speaking, this includes both configuration management and security monitoring. The goals for each of these areas are different. By checking both, we achieve higher assurance that the system is operating as desired. We discuss configuration management in the management section, and security monitoring below.

Intrusion detection system

Host-based intrusion detection tools are also useful for automated validation of the cloud internals. There are a wide variety of host-based intrusion detection tools available. Some are open source projects that are freely available, while others are commercial. Typically these tools analyze data from a variety of sources and produce security alerts based on rule sets and/or training. Typical capabilities include log analysis, file integrity checking, policy monitoring, and rootkit detection. More advanced – often custom – tools can validate that in-memory process images match the on-disk executable and validate the execution state of a running process.

One critical policy decision for a cloud architect is what to do with the output from a security monitoring tool. There are effectively two options. The first is to alert a human to investigate and/or take corrective action. This could be done by including the security alert in a log or events feed for cloud administrators. The second option is to have the cloud take some form of remedial action automatically, in addition to logging the event. Remedial actions could include anything from re-installing a node to performing a minor service configuration. However, automated remedial action can be challenging due to the possibility of false positives.

False positives occur when the security monitoring tool produces a security alert for a benign event. Due to the nature of security monitoring tools, false positives will most certainly occur from time to time. Typically a cloud administrator can tune security monitoring tools to reduce the false positives, but this may also reduce the overall detection rate at the same time. These classic trade-offs must be understood and accounted for when setting up a security monitoring system in the cloud.

The selection and configuration of a host-based intrusion detection tool is highly deployment specific. We recommend starting by exploring the fol-

lowing open source projects which implement a variety of host-based intrusion detection and file monitoring features.

- OSSEC
- Samhain
- Tripwire
- AIDE

Network intrusion detection tools complement the host-based tools. OpenStack doesn't have a specific network IDS built-in, but OpenStack Networking provides a plug-in mechanism to enable different technologies through the Networking API. This plug-in architecture will allow tenants to develop API extensions to insert and configure their own advanced networking services like a firewall, an intrusion detection system, or a VPN between the VMs.

Similar to host-based tools, the selection and configuration of a network-based intrusion detection tool is deployment specific. Snort is the leading open source networking intrusion detection tool, and a good starting place to learn more.

There are a few important security considerations for network and host-based intrusion detection systems.

- It is important to consider the placement of the Network IDS on the cloud (for example, adding it to the network boundary and/or around sensitive networks). The placement depends on your network environment but make sure to monitor the impact the IDS may have on your services depending on where you choose to add it. Encrypted traffic, such as TLS, cannot generally be inspected for content by a Network IDS. However, the Network IDS may still provide some benefit in identifying anomalous unencrypted traffic on the network.
- In some deployments it may be required to add host-based IDS on sensitive components on security domain bridges. A host-based IDS may detect anomalous activity by compromised or unauthorized processes on the component. The IDS should transmit alert and log information on the Management network.

Server hardening

Servers in the cloud, including undercloud and overcloud infrastructure, should implement hardening best practices. As OS and server hardening is

common, applicable best practices including but not limited to logging, user account restrictions, and regular updates will not be covered here, but should be applied to all infrastructure.

File integrity management (FIM)

File integrity management (FIM) is the method of ensuring that files such as sensitive system or application configuration files are not corrupted or changed to allow unauthorized access or malicious behavior. This can be done through a utility such as Samhain that will create a checksum hash of the specified resource and then validate that hash at regular intervals, or through a tool such as DMVerity that can take a hash of block devices and will validate those hashes as they are accessed by the system before they are presented to the user.

These should be put in place to monitor and report on changes to system, hypervisor, and application configuration files such as `/etc/pam.d/system-auth` and `/etc/keystone.conf`, as well as kernel modules (such as virtio). Best practice is to use the **lsmod** command to show what is regularly being loaded on a system to help determine what should or should not be included in FIM checks.

Management interfaces

It is necessary for administrators to perform command and control over the cloud for various operational functions. It is important these command and control facilities are understood and secured.

OpenStack provides several management interfaces for operators and tenants:

- OpenStack dashboard (horizon)
- OpenStack API
- Secure shell (SSH)
- OpenStack management utilities such as `nova-manage` and `glance-manage`
- Out-of-band management interfaces, such as IPMI

Dashboard

The OpenStack dashboard (horizon) provides administrators and tenants with a web-based graphical interface to provision and access cloud-

based resources. The dashboard communicates with the back-end services through calls to the OpenStack API.

Capabilities

- As a cloud administrator, the dashboard provides an overall view of the size and state of your cloud. You can create users and tenants/projects, assign users to tenant/projects and set limits on the resources available for them.
- The dashboard provides tenant-users a self-service portal to provision their own resources within the limits set by administrators.
- The dashboard provides GUI support for routers and load-balancers. For example, the dashboard now implements all of the main Networking features.
- It is an extensible *Django* web application that allows easy plug-in of third-party products and services, such as billing, monitoring, and additional management tools.
- The dashboard can also be branded for service providers and other commercial vendors.

Security considerations

- The dashboard requires cookies and JavaScript to be enabled in the web browser.
- The web server that hosts the dashboard should be configured for TLS to ensure data is encrypted.
- Both the horizon web service and the OpenStack API it uses to communicate with the back end are susceptible to web attack vectors such as denial of service and must be monitored.
- It is now possible (though there are numerous deployment/security implications) to upload an image file directly from a user's hard disk to OpenStack Image Service through the dashboard. For multi-gigabyte images it is still strongly recommended that the upload be done using the **glance** CLI.
- Create and manage security groups through dashboard. The security groups allows L3-L4 packet filtering for security policies to protect virtual machines.

References

Icehouse Release Notes

OpenStack API

The OpenStack API is a RESTful web service endpoint to access, provision and automate cloud-based resources. Operators and users typically access the API through command-line utilities (for example, **nova** or **glance**), language-specific libraries, or third-party tools.

Capabilities

- To the cloud administrator, the API provides an overall view of the size and state of the cloud deployment and allows the creation of users, tenants/projects, assigning users to tenants/projects, and specifying resource quotas on a per tenant/project basis.
- The API provides a tenant interface for provisioning, managing, and accessing their resources.

Security considerations

- The API service should be configured for TLS to ensure data is encrypted.
- As a web service, OpenStack API is susceptible to familiar web site attack vectors such as denial of service attacks.

Secure shell (SSH)

It has become industry practice to use secure shell (SSH) access for the management of Linux and Unix systems. SSH uses secure cryptographic primitives for communication. With the scope and importance of SSH in typical OpenStack deployments, it is important to understand best practices for deploying SSH.

Host key fingerprints

Often overlooked is the need for key management for SSH hosts. As most or all hosts in an OpenStack deployment will provide an SSH service, it is important to have confidence in connections to these hosts. It cannot be understated that failing to provide a reasonably secure and accessible method to verify SSH host key fingerprints is ripe for abuse and exploitation.

All SSH daemons have private host keys and, upon connection, offer a host key fingerprint. This host key fingerprint is the hash of an unsigned public key. It is important these host key fingerprints are known in advance of making SSH connections to those hosts. Verification of host key fingerprints is instrumental in detecting man-in-the-middle attacks.

Typically, when an SSH daemon is installed, host keys will be generated. It is necessary that the hosts have sufficient entropy during host key generation. Insufficient entropy during host key generation can result in the possibility to eavesdrop on SSH sessions.

Once the SSH host key is generated, the host key fingerprint should be stored in a secure and queryable location. One particularly convenient solution is DNS using SSHFP resource records as defined in RFC-4255. For this to be secure, it is necessary that DNSSEC be deployed.

Management utilities

The OpenStack Management Utilities are open-source Python command-line clients that make API calls. There is a client for each OpenStack service (for example, `nova`, `glance`). In addition to the standard CLI client, most of the services have a management command-line utility which makes direct calls to the database. These dedicated management utilities are slowly being deprecated.

Security considerations

- The dedicated management utilities (*-manage) in some cases use the direct database connection.
- Ensure that the .rc file which has your credential information is secured.

References

OpenStack End User Guide section command-line clients overview

OpenStack End User Guide section Download and source the OpenStack RC file

Out-of-band management interface

OpenStack management relies on out-of-band management interfaces such as the IPMI protocol to access into nodes running OpenStack components. IPMI is a very popular specification to remotely manage, diagnose,

and reboot servers whether the operating system is running or the system has crashed.

Security considerations

- Use strong passwords and safeguard them, or use client-side TLS authentication.
- Ensure that the network interfaces are on their own private(management or a separate) network. Segregate management domains with firewalls or other network gear.
- If you use a web interface to interact with the *BMC*/IPMI, always use the TLS interface, such as HTTPS or port 443. This TLS interface should **NOT** use self-signed certificates, as is often default, but should have trusted certificates using the correctly defined fully qualified domain names (FQDNs).
- Monitor the traffic on the management network. The anomalies might be easier to track than on the busier compute nodes.

Out of band management interfaces also often include graphical machine console access. It is often possible, although not necessarily default, that these interfaces are encrypted. Consult with your system software documentation for encrypting these interfaces.

References

Hacking servers that are turned off

Case studies

Previously we discussed typical OpenStack management interfaces and associated backplane issues. We will now approach these issues by returning to the Alice and Bob case studies (See the section called "Introduction to case studies" [21]) where Alice is deploying a government cloud and Bob is deploying a public cloud each with different security requirements. In this section, we will look into how both Alice and Bob will address:

- Cloud administration
- Self service
- Data replication and recovery

- SLA and security monitoring

Alice's private cloud

When building her private cloud, while air-gapped, Alice still needs to consider her service management interfaces. Before deploying her private cloud, Alice has completed her system documentation. Specifically she has identified which OpenStack services will exist in each security domain. From there Alice has further restricted access to management interfaces by deploying a combination of IDS, TLS encryption, and physical network isolation. Additionally, Alice requires high availability and redundant services. Thus, Alice sets up redundant infrastructure for various OpenStack API services.

Alice also needs to provide assurances that the physical servers and hypervisors have been built from a known secure state into a well-defined configuration. To enable this, Alice uses a combination of a Configuration Management platform to configure each machine according to the standards and regulations she must comply with. It will also enable Alice to report periodically on the state of her cloud and perform remediation to a known state should anything be out of the ordinary. Additionally, Alice provides hardware assurances by using a PXE system to build her nodes from a known set of base images. During the boot process, Alice provides further assurances by enabling Intel TXT and related trusted boot technologies provided by the hardware.

Bob's public cloud

As a public cloud provider, Bob is concerned with both the continuous availability of management interfaces and the security of transactions to the management interfaces. To that end Bob implements multiple redundant OpenStack API endpoints for the services his cloud will run. Additionally on the public network Bob uses TLS to encrypt all transactions between his customers and his cloud interfaces. To isolate his cloud operations Bob has physically isolated his management, instance migration, and storage networks.

To ease scaling and reduce management overhead Bob implements a configuration management system. For customer data assurances, Bob offers a backup as a service product as requirements will vary between customers. Finally, Bob does not provide a "baremetal" or the ability to schedule an entire node, so to reduce management overhead and increase operational efficiency Bob does not implement any node boot time security.

4. Secure communication

Inter-device communication is an issue still plaguing security researchers. Between large project errors such as Heartbleed or more advanced attacks such as BEAST and CRIME, secure methods of communication over a network are becoming more important. It should be remembered, however that encryption should be applied as one part of a larger security strategy. The compromise of an endpoint means that an attacker no longer needs to break the encryption used, but is able to view and manipulate messages as they are processed by the system.

This chapter will review several features around configuring TLS to secure both internal and external resources, and will call out specific categories of systems that should be given specific attention.

Introduction to TLS and SSL

There are a number of situations where there is a security requirement to assure the confidentiality or integrity of network traffic in an OpenStack deployment. This is generally achieved using cryptographic measures, such as the Transport Layer Security (TLS) protocol.

In a typical deployment all traffic transmitted over public networks is secured, but security best practice dictates that internal traffic must also be secured. It is insufficient to rely on security domain separation for protection. If an attacker gains access to the hypervisor or host resources, compromises an API endpoint, or any other service, they must not be able to easily inject or capture messages, commands, or otherwise affect the management capabilities of the cloud.

All domains should be secured with TLS, including the management domain services and intra-service communications. TLS provides the mechanisms to ensure authentication, non-repudiation, confidentiality, and integrity of user communications to the OpenStack services and between the OpenStack services themselves.

Due to the published vulnerabilities in the Secure Sockets Layer (SSL) protocols, we strongly recommend that TLS is used in preference to SSL, and

that SSL is disabled in all cases, unless compatibility with obsolete browsers or libraries is required.

Public Key Infrastructure (PKI) is the framework for securing communication in a network. It consists of a set of systems and processes to ensure traffic can be sent securely while validating the identities of the parties. The core components of PKI are:

End entity	User, process, or system that is the subject of a certificate.
Certification Authority (*CA*)	Defines certificate policies, management, and issuance of certificates.
Registration Authority (RA)	An optional system to which a CA delegates certain management functions.
Repository	Where the end entity certificates and certificate revocation lists are stored and looked up - sometimes referred to as the *certificate bundle*.
Relying party	The endpoint that is trusting that the CA is valid.

PKI builds the framework on which to provide encryption algorithms, cipher modes, and protocols for securing data and authentication. We strongly recommend securing all services with Public Key Infrastructure (PKI), including the use of TLS for API endpoints. It is impossible for the encryption or signing of transports or messages alone to solve all these problems. Hosts themselves must be secure and implement policy, namespaces, and other controls to protect their private credentials and keys. However, the challenges of key management and protection do not reduce the necessity of these controls, or lessen their importance.

Certification authorities

Many organizations have an established Public Key Infrastructure with their own certification authority (CA), certificate policies, and management for which they should use to issue certificates for internal OpenStack users or services. Organizations in which the public security domain is Internet facing will additionally need certificates signed by a widely recognized public CA. For cryptographic communications over the management network, it is recommended one not use a public CA. Instead, we expect and recommend most deployments deploy their own internal CA.

It is recommended that the OpenStack cloud architect consider using separate PKI deployments for internal systems and customer facing services. This allows the cloud deployer to maintain control of their PKI infrastructure and among other things makes requesting, signing and deploying certificates for internal systems easier. Advanced configurations may use separate PKI deployments for different security domains. This allows deployers to maintain cryptographic separation of environments, ensuring that certificates issued to one are not recognized by another.

Certificates used to support TLS on internet facing cloud endpoints (or customer interfaces where the customer is not expected to have installed anything other than standard operating system provided certificate bundles) should be provisioned using Certificate Authorities that are installed in the operating system certificate bundle. Typical well known vendors include Verisign and Thawte but many others exist.

There are many management, policy, and technical challenges around creating and signing certificates. This is an area where cloud architects or operators may wish to seek the advice of industry leaders and vendors in addition to the guidance recommended here.

TLS libraries

Various components, services, and applications within the OpenStack ecosystem or dependencies of OpenStack are implemented and can be configured to use TLS libraries. The TLS and HTTP services within OpenStack are typically implemented using OpenSSL which has a module that has been validated for FIPS 140-2. However, keep in mind that each application or service can still introduce weaknesses in how they use the OpenSSL libraries.

Cryptographic algorithms, cipher modes, and protocols

We recommend only using TLSv1.2. TLSv1.0 and TLSv1.1 may be used for broad client compatibility but we recommend using caution and only enabling these protocols if you have a strong requirement to do so. Other TLS versions, explicitly older versions, should not be used. These older versions include SSLv2 which is deprecated, and SSLv3 which suffers from the attack known as POODLE. When using TLS v1.2 and in control of both the clients and the server, the cipher suite should be limited to `ECDHE-ECDSA-AES256-GCM-SHA384`. Where you don't control both ends and are

using TLS v1+, the more general `HIGH:!aNULL:!eNULL:!DES:!3DES` is a reasonable cipher selection.

However, as this book does not intend to be a thorough reference on cryptography we do not wish to be prescriptive about what specific algorithms or cipher modes you should enable or disable in your OpenStack services. However, there are some authoritative references we would like to recommend for further information:

- National Security Agency, Suite B Cryptography
- OWASP Guide to Cryptography
- OWASP Transport Layer Protection Cheat Sheet
- SoK: SSL and HTTPS: Revisiting past challenges and evaluating certificate trust model enhancements
- The Most Dangerous Code in the World: Validating SSL Certificates in Non-Browser Software
- OpenSSL and FIPS 140-2

Summary

Given the complexity of the OpenStack components and the number of deployment possibilities, you must take care to ensure that each component gets the appropriate configuration of TLS certificates, keys, and CAs. Subsequent sections discuss the following services:

- Compute API endpoints
- Identity API endpoints
- Networking API endpoints
- Storage API endpoints
- Messaging server
- Database server
- Dashboard

TLS proxies and HTTP services

OpenStack endpoints are HTTP services providing APIs to both end-users on public networks and to other OpenStack services on the management

network. It is highly recommended that all of these requests, both internal and external, operate over TLS. To achieve this goal, API services must be deployed behind a TLS proxy that can establish and terminate TLS sessions. The following table offers a non-exhaustive list of open source software that can be used for this purpose:

- Pound
- Stud
- nginx
- Apache httpd

In cases where software termination offers insufficient performance, hardware accelerators may be worth exploring as an alternative option. It is important to be mindful of the size of requests that will be processed by any chosen TLS proxy.

Examples

Below we provide sample recommended configuration settings for enabling TLS in some of the more popular web servers/TLS terminators.

Before we delve into the configurations, we briefly discuss the ciphers' configuration element and its format. A more exhaustive treatment on available ciphers and the OpenSSL cipher list format can be found at: ciphers.

```
ciphers = "HIGH:!RC4:!MD5:!aNULL:!eNULL:!EXP:!LOW:!MEDIUM"
```

or

```
ciphers = "kEECDH:kEDH:kRSA:HIGH:!RC4:!MD5:!aNULL:!eNULL:!EXP:!
LOW:!MEDIUM"
```

Cipher string options are separated by ":", while "!" provides negation of the immediately following element. Element order indicates preference unless overridden by qualifiers such as HIGH. Let us take a closer look at the elements in the above sample strings.

kEECDH:kEDH Ephemeral Elliptic Curve Diffie-Hellman (abbreviated as EECDH and ECDHE).

Ephemeral Diffie-Hellman (abbreviated either as EDH or DHE) uses prime field groups.

	Both approaches provide Perfect Forward Secrecy (PFS). See the section called "Perfect forward secrecy" [54] for additional discussion on properly configuring PFS. Ephemeral Elliptic Curves require the server to be configured with a named curve, and provide better security than prime field groups and at lower computational cost. However, prime field groups are more widely implemented, and thus typically both are included in list.
kRSA	Cipher suites using the RSA exchange, authentication or either respectively.
HIGH	Selects highest possible security cipher in the negotiation phase. These typically have keys of length 128 bits or longer.
!RC4	No RC4. RC4 has flaws in the context of TLS V3. See On the Security of RC4 in TLS and WPA.
!MD5	No MD5. MD5 is not collision resistant, and thus not acceptable for Message Authentication Codes (MAC) or signatures.
!aNULL:!eNULL	Disallows clear text.
!EXP	Disallows export encryption algorithms, which by design tend to be weak, typically using 40 and 56 bit keys. US Export restrictions on cryptography systems have been lifted and no longer need to be supported.
!LOW:!MEDIUM	Disallows low (56 or 64 bit long keys) and medium (128 bit long keys) ciphers because of their vulnerability to brute force attacks (example 2-DES). This rule still permits Triple Data Encryption Standard (Triple DES) also known as Triple Data Encryption Algorithm (TDEA) and the Advanced Encryption Standard (AES), each of which has keys greater than equal to 128 bits and thus more secure.
Protocols	Protocols are enabled/disabled through SSL_CTX_set_options. We recommend disabling SSLv2/v3 and enabling TLS.

Pound

This Pound example enables AES-NI acceleration, which helps to improve performance on systems with processors that support this feature.

```
## see pound(8) for details
daemon      1
######################################################################
## global options:
User        "swift"
Group       "swift"
#RootJail   "/chroot/pound"
## Logging: (goes to syslog by default)
##  0   no logging
##  1   normal
##  2   extended
##  3   Apache-style (common log format)
LogLevel    0
## turn on dynamic scaling (off by default)
# Dyn Scale 1
## check backend every X secs:
Alive       30
## client timeout
#Client     10
## allow 10 second proxy connect time
ConnTO      10
## use hardware-acceleration card supported by openssl(1):
SSLEngine   "aesni"
# poundctl control socket
Control "/var/run/pound/poundctl.socket"
######################################################################
## listen, redirect and ... to:
## redirect all swift requests on port 443 to local swift proxy
ListenHTTPS
    Address 0.0.0.0
    Port    443
    Cert    "/etc/pound/cert.pem"
    ## Certs to accept from clients
    ##  CAlist      "CA_file"
    ## Certs to use for client verification
    ##  VerifyList  "Verify_file"
    ## Request client cert - don't verify
    ##  Ciphers     "AES256-SHA"
    ## allow PUT and DELETE also (by default only GET, POST and
 HEAD)?:
    NoHTTPS11   0
    ## allow PUT and DELETE also (by default only GET, POST and
 HEAD)?:
    xHTTP       1
    Service
        BackEnd
```

```
            Address 127.0.0.1
            Port    80
        End
    End
End
```

Stud

The *ciphers* line can be tweaked based on your needs, however this is a reasonable starting place.

```
# SSL x509 certificate file.
pem-file = "
# SSL protocol.
tls = on
ssl = off
# List of allowed SSL ciphers.
# OpenSSL's high-strength ciphers which require authentication
# NOTE: forbids clear text, use of RC4 or MD5 or LOW and MEDIUM
 strength ciphers
ciphers = "HIGH:!RC4:!MD5:!aNULL:!eNULL:!EXP:!LOW:!MEDIUM"
# Enforce server cipher list order
prefer-server-ciphers = on
# Number of worker processes
workers = 4
# Listen backlog size
backlog = 1000
# TCP socket keepalive interval in seconds
keepalive = 3600
# Chroot directory
chroot = ""
# Set uid after binding a socket
user = "www-data"
# Set gid after binding a socket
group = "www-data"
# Quiet execution, report only error messages
quiet = off
# Use syslog for logging
syslog = on
# Syslog facility to use
syslog-facility = "daemon"
# Run as daemon
daemon = off
# Report client address using SENDPROXY protocol for haproxy
# Disabling this until we upgrade to HAProxy 1.5
write-proxy = off
```

nginx

This nginx example requires TLS v1.1 or v1.2 for maximum security. The `ssl_ciphers` line can be tweaked based on your needs, however this is a reasonable starting place.

```
server {
    listen : ssl;
    ssl_certificate ;
    ssl_certificate_key ;
    ssl_protocols TLSv1.1 TLSv1.2;
    ssl_ciphers HIGH:!RC4:!MD5:!aNULL:!eNULL:!EXP:!LOW:!MEDIUM
    ssl_session_tickets off;

    server_name _;
    keepalive_timeout 5;

    location / {

    }
}
```

Apache

```
<VirtualHost <ip address>:80>
  ServerName <site FQDN>
  RedirectPermanent / https://<site FQDN>/
</VirtualHost>
<VirtualHost <ip address>:443>
  ServerName <site FQDN>
  SSLEngine On
  SSLProtocol +TLSv1 +TLSv1.1 +TLSv1.2,
  SSLCipherSuite HIGH:!RC4:!MD5:!aNULL:!eNULL:!EXP:!LOW:!MEDIUM
  SSLCertificateFile    /path/<site FQDN>.crt
  SSLCACertificateFile  /path/<site FQDN>.crt
  SSLCertificateKeyFile /path/<site FQDN>.key
  WSGIScriptAlias / <WSGI script location>
  WSGIDaemonProcess horizon user=<user> group=<group> processes=
3 threads=10
  Alias /static <static files location>
  <Directory <WSGI dir>>
    # For http server 2.2 and earlier:
    Order allow,deny
    Allow from all

    # Or, in Apache http server 2.4 and later:
    # Require all granted
  </Directory>
</VirtualHost>
```

Compute API SSL endpoint in Apache, which you must pair with a short WSGI script.

```
<VirtualHost <ip address>:8447>
  ServerName <site FQDN>
  SSLEngine On
  SSLProtocol +TLSv1 +TLSv1.1 +TLSv1.2
  SSLCipherSuite HIGH:!RC4:!MD5:!aNULL:!eNULL:!EXP:!LOW:!MEDIUM
  SSLCertificateFile    /path/<site FQDN>.crt
  SSLCACertificateFile  /path/<site FQDN>.crt
  SSLCertificateKeyFile /path/<site FQDN>.key
  SSLSessionTickets Off
  WSGIScriptAlias / <WSGI script location>
  WSGIDaemonProcess osapi user=<user> group=<group> processes=3
 threads=10
  <Directory <WSGI dir>>
    # For http server 2.2 and earlier:
    Order allow,deny
    Allow from all

    # Or, in Apache http server 2.4 and later:
    # Require all granted
  </Directory>
</VirtualHost>
```

HTTP strict transport security

We recommend that all production deployments use HTTP strict transport security (HSTS). This header prevents browsers from making insecure connections after they have made a single secure one. If you have deployed your HTTP services on a public or an untrusted domain, HSTS is especially important. To enable HSTS, configure your web server to send a header like this with all requests:

```
Strict-Transport-Security: max-age=31536000; includeSubDomains
```

Start with a short timeout of 1 day during testing, and raise it to one year after testing has shown that you have not introduced problems for users. Note that once this header is set to a large timeout, it is (by design) very difficult to disable.

Perfect forward secrecy

Configuring TLS servers for perfect forward secrecy requires careful planning around key size, session IDs, and session tickets. In addition, for multi-server deployments, shared state is also an important consideration. The example configurations for Apache and Nginx above disable the session

tickets options to help mitigate some of these concerns. Real-world deployments may desire to enable this feature for improved performance. This can be done securely, but would require special consideration around key management. Such configurations are beyond the scope of this guide. We suggest reading How to botch TLS forward secrecy by ImperialViolet as a starting place for understanding the problem space.

Secure reference architectures

We recommend using SSL/TLS on both public networks and management networks in the section called "TLS proxies and HTTP services" [48]. However, if actually deploying SSL/TLS everywhere is too difficult, we recommend evaluating your OpenStack SSL/TLS needs and following one of the architectures discussed here.

The first thing one should do when evaluating their OpenStack SSL/TLS needs is to identify the threats. You can divide these threats into external and internal attacker categories, but the lines tend to get blurred since certain components of OpenStack operate on both the public and management networks.

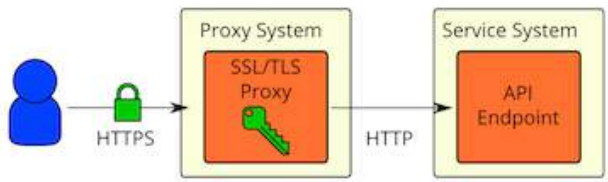
For publicly facing services, the threats are pretty straightforward. Users will be authenticating against horizon and keystone with their username and password. Users will also be accessing the API endpoints for other services using their keystone tokens. If this network traffic is unencrypted, passwords and tokens can be intercepted by an attacker using a man-in-the-middle attack. The attacker can then use these valid credentials to perform malicious operations. All real deployments should be using SSL/TLS to protect publicly facing services.

For services that are deployed on management networks, the threats aren't so clear due to the bridging of security domains with network security. There is always the chance that an administrator with access to the management network decides to do something malicious. SSL/TLS isn't going to help in this situation if the attacker is allowed to access the private key. Not everyone on the management network would be allowed to access the private key of course, so there is still value in using SSL/TLS to protect yourself from internal attackers. Even if everyone that is allowed to access your management network is 100% trusted, there is still a threat that an unauthorized user gains access to your internal network by exploiting a misconfiguration or software vulnerability. One must keep in mind that you have users running their own code on instances in the OpenStack Compute nodes, which are deployed on the management network. If a

vulnerability allows them to break out of the hypervisor, they will have access to your management network. Using SSL/TLS on the management network can minimize the damage that an attacker can cause.

SSL/TLS proxy in front

It is generally accepted that it is best to encrypt sensitive data as early as possible and decrypt it as late as possible. Despite this best practice, it seems that it's common to use a SSL/TLS proxy in front of the OpenStack services and use clear communication afterwards as shown below:

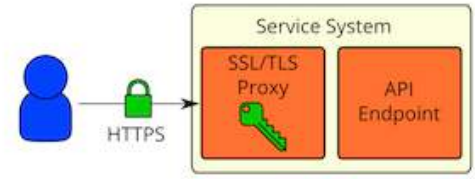


Some of the concerns with the use of SSL/TLS proxies as pictured above:

- Native SSL/TLS in OpenStack services does not perform/scale as well as SSL proxies (particularly for Python implementations like Eventlet).
- Native SSL/TLS in OpenStack services not as well scrutinized/ audited as more proven solutions.
- Native SSL/TLS configuration is difficult (not well documented, tested, or consistent across services).
- Privilege separation (OpenStack service processes should not have direct access to private keys used for SSL/TLS).
- Traffic inspection needs for load balancing.

All of the above are valid concerns, but none of them prevent SSL/TLS from being used on the management network. Let's consider the next deployment model.

SSL/TLS on same physical hosts as API endpoints

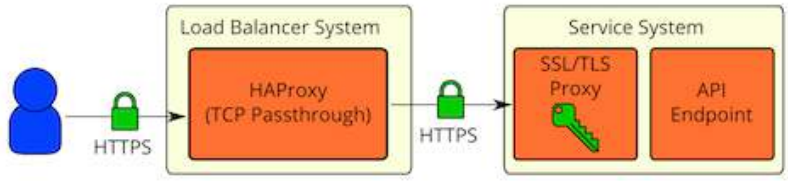


This is very similar to the "SSL/TLS in front model" but the SSL/TLS proxy is on the same physical system as the API endpoint. The API endpoint would be configured to only listen on the local network interface. All remote communication with the API endpoint would go through the SSL/TLS proxy. With this deployment model, we address a number of the bullet points in "SSL/TLS in front model" . A proven SSL implementation that performs well would be used. The same SSL proxy software would be used for all services, so SSL configuration for the API endpoints would be consistent. The OpenStack service processes would not have direct access to the private keys used for SSL/TLS, as you would run the SSL proxies as a different user and restrict access using permissions (and additionally mandatory access controls using something like SELinux). We would ideally have the API endpoints listen on a Unix socket such that we could restrict access to it using permissions and mandatory access controls as well. Unfortunately, this does not seem to work currently in Eventlet from our testing. It is a good future development goal.

SSL/TLS over load balancer

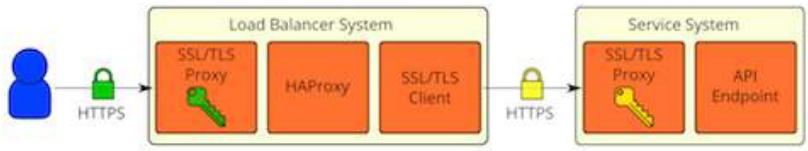
What about high availability or load balanced deployments that need to inspect traffic? The previous deployment model (SSL/TLS on same physical hosts as API endpoints) would not allow for deep packet inspection since the traffic is encrypted. If the traffic only needs to be inspected for basic routing purposes, it might not be necessary for the load balancer to have access to the unencrypted traffic. HAProxy has the ability to extract the SSL/TLS session ID during the handshake, which can then be used to achieve session affinity (configuration details here). HAProxy can also use the TLS Server Name Indication (SNI) extension to determine where traffic should be routed to (configuration details here). These features likely cover some of the most common load balancer needs. HAProxy would be able

to just pass the HTTPS traffic straight through to the API endpoint systems in this case:



Cryptographic seperation of external and internal environments

What if you want cryptographic separation of your external and internal environments? A public cloud provider would likely want their public facing services (or proxies) to use certificates that are issued by a CA that chains up to a trusted Root CA that is distributed in popular web browser software for SSL/TLS. For the internal services, one might want to instead use their own PKI to issue certificates for SSL/TLS. This cryptographic separation can be accomplished by terminating SSL at the network boundary, then re-encrypting using the internally issued certificates. The traffic will be unencrypted for a brief period on the public facing SSL/TLS proxy, but it will never be transmitted over the network in the clear. The same re-encryption approach that is used to achieve cryptographic separation can also be used if deep packet inspection is really needed on a load balancer. Here is what this deployment model would look like:



As with most things, there are trade-offs. The main trade-off is going to be between security and performance. Encryption has a cost, but so does being hacked. The security and performance requirements are going to be

different for every deployment, so how SSL/TLS is used will ultimately be an individual decision.

Case studies

Earlier in the section called "Introduction to case studies" [21] we introduced the Alice and Bob case study where Alice is deploying a government cloud and Bob is deploying a public cloud each with different security requirements. Here we discuss how Alice and Bob would address deployment of PKI certification authorities (CA) and certificate management.

Alice's private cloud

Alice as a cloud architect within a government agency knows that her agency operates its own certification authority. Alice contacts the PKI office in her agency that manages her PKI and certificate issuance. Alice obtains certificates issued by this CA and configures the services within both the public and management security domains to use these certificates. Since Alice's OpenStack deployment exists entirely on a disconnected from the Internet network, she makes sure to remove all default CA bundles that contain external public CA providers to ensure the OpenStack services only accept client certificates issued by her agency's CA.

Bob's public cloud

Bob is architecting a public cloud and needs to ensure that the publicly facing OpenStack services are using certificates issued by a major public CA. Bob acquires certificates for his public OpenStack services and configures the services to use PKI and TLS and includes the public CAs in his trust bundle for the services. Additionally, Bob also wants to further isolate the internal communications amongst the services within the management security domain. Bob contacts the team within his organization that is responsible for managing his organization's PKI and issuance of certificates using their own internal CA. Bob obtains certificates issued by this internal CA and configures the services that communicate within the management security domain to use these certificates and configures the services to only accept client certificates issued by his internal CA.

5. API endpoints

The process of engaging an OpenStack cloud is started through the querying of 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.

API endpoint configuration recommendations

Internal API communications

OpenStack provides both public facing 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.

Services select their respective API endpoints based on the OpenStack service catalog. These services might not obey the listed public or internal API end point values. This can lead to internal management traffic being routed to external API endpoints.

Configure internal URLs in the Identity service catalog

The Identity service catalog should be aware of your internal URLs. While this feature is not utilized by default, it may be leveraged through configuration. Additionally, it should be forward-compatible with expectant changes once this behavior becomes the default.

To register an internal URL for an endpoint:

```
$ keystone endpoint-create \
 --region RegionOne \
 --service-id=1ff4ece13c3e48d8a6461faebd9cd38f \
 --publicurl='https://public-ip:8776/v1/%(tenant_id)s' \
 --internalurl='https://management-ip:8776/v1/%(tenant_id)s' \
 --adminurl='https://management-ip:8776/v1/%(tenant_id)s'
```

Configure applications for internal URLs

You can force some services to use specific API endpoints. Therefore, it is recommended that each OpenStack service communicating to the API of another service must be explicitly configured to access the proper internal API endpoint.

Each project may 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.

Configuration example #1: nova

```
[DEFAULT]
cinder_catalog_info='volume:cinder:internalURL'
glance_protocol='https'
neutron_url='https://neutron-host:9696'
neutron_admin_auth_url='https://neutron-host:9696'
s3_host='s3-host'
s3_use_ssl=True
```

Configuration example #2: cinder

```
glance_host='https://glance-server'
```

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. We recommend implementers make careful consideration of the potential exposure introduced by the addition of non-standard software components to their HTTP request pipeline.

For more information about Paste Deploy, see http://pythonpaste.org/deploy/.

API endpoint process isolation and policy

You should isolate API endpoint processes, especially those that reside within the public security domain should be isolated as much as possible.

Where deployments allow, API endpoints should be deployed on separate hosts for increased isolation.

Namespaces

Many operating systems now provide compartmentalization support. Linux supports namespaces to assign processes into independent domains. Other parts of this guide cover system compartmentalization in more detail.

Network policy

Because API endpoints typically bridge multiple security domains, you must pay particular attention to the compartmentalization of the API processes. See the section called "Bridging security domains" [15] for additional information in this area.

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 or AppArmor), and optionally global network policy.

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 not only by Discretionary Access Controls, but through Mandatory Access Controls. The goal of these enhanced access controls is to aid in the containment and escalation of API endpoint security breaches. With mandatory access controls, such breaches severely limit access to resources and provide earlier alerting on such events.

Case studies

Earlier in the section called "Introduction to case studies" [21] we introduced the Alice and Bob case studies where Alice is deploying a private government cloud and Bob is deploying a public cloud each with different security requirements. Here we discuss how Alice and Bob would address endpoint configuration to secure their private and public clouds. Alice's

cloud is not publicly accessible, but she is still concerned about securing the endpoints against improper use. Bob's cloud, being public, must take measures to reduce the risk of attacks by external adversaries.

Alice's private cloud

Alice's organization requires that the security architecture protect the access to the public and private endpoints, so she elects to use the Apache TLS proxy on both public and internal services. Alice's organization has implemented its own certificate authority. Alice contacts the PKI office in her agency that manages her PKI and certificate issuance. Alice obtains certificates issued by this CA and configures the services within both the public and management security domains to use these certificates. Since Alice's OpenStack deployment exists entirely on a network disconnected from the Internet, she makes sure to remove all default CA bundles that contain external public CA providers to ensure the OpenStack services only accept client certificates issued by her agency's CA. Alice has registered all of the services in the Identity service's catalog, using the internal URLs for access by internal services. She has installed host-based intrusion detection on all of the API endpoints.

Bob's public cloud

Bob must also protect the access to the public and private endpoints, so he elects to use the Apache TLS proxy on both public and internal services. On the public services, he has configured the certificate key files with certificates signed by a well-known Certificate Authority. He has used his organization's self-signed CA to sign certificates in the internal services on the Management network. Bob has registered his services in the Identity service's catalog, using the internal URLs for access by internal services. Bob's public cloud runs services on SELinux, which he has configured with a mandatory access control policy to reduce the impact of any publicly accessible services that may be compromised. He has also configured the endpoints with a host-based IDS.

6. Identity

Identity service (keystone) provides identity, token, catalog, and policy services for use specifically by services in the OpenStack family. Identity service is organized as a group of internal services exposed on one or many endpoints. Many of these services are used in a combined fashion by the frontend, for example an authenticate call will validate user/project credentials with the identity service and, upon success, create and return a token with the token service. Further information can be found by reading the Keystone Developer Documentation.

Authentication

The OpenStack Identity service (keystone) supports multiple methods of authentication, including user name & password, LDAP, and external authentication methods. Upon successful authentication, The Identity service provides the user with an authorization token used for subsequent service requests.

Transport Layer Security (TLS) provides authentication between services and persons using X.509 certificates. Although the default mode for TLS is server-side only authentication, certificates may also be used for client authentication.

Invalid login attempts

The Identity service does not provide a method to limit access to accounts after repeated unsuccessful login attempts. A pattern of repetitive failed login attempts is generally an indicator of brute-force attacks (refer to Figure 1.1, “Attack types” [20]). This type of attack is more prevalent in public cloud deployments.

Prevention is possible by using an external authentication system that blocks out an account after some configured number of failed login at-

tempts. The account then may only be unlocked with further side-channel intervention.

If prevention is not an option, detection can be used to mitigate damage. Detection involves frequent review of access control logs to identify unauthorized attempts to access accounts. Possible remediation would include reviewing the strength of the user password, or blocking the network source of the attack through firewall rules. Firewall rules on the keystone server that restrict the number of connections could be used to reduce the attack effectiveness, and thus dissuade the attacker.

In addition, it is useful to examine account activity for unusual login times and suspicious actions, and take corrective actions such as disabling the account. Oftentimes this approach is taken by credit card providers for fraud detection and alert.

Multi-factor authentication

Employ multi-factor authentication for network access to privileged user accounts. The Identity service supports external authentication services through the Apache web server that can provide this functionality. Servers may also enforce client-side authentication using certificates.

This recommendation provides insulation from brute force, social engineering, and both spear and mass phishing attacks that may compromise administrator passwords.

Authentication methods

Internally implemented authentication methods

The Identity service can store user credentials in an SQL Database, or may use an LDAP-compliant directory server. The Identity database may be separate from databases used by other OpenStack services to reduce the risk of a compromise of the stored credentials.

When you use a user name and password to authenticate, Identity does not enforce policies on password strength, expiration, or failed authentication attempts as recommended by NIST Special Publication 800-118 (draft). Organizations that desire to enforce stronger password policies should consider using Identity extensions or external authentication services.

LDAP simplifies integration of Identity authentication into an organization's existing directory service and user account management processes.

Authentication and authorization policy in OpenStack may be delegated to another service. A typical use case is an organization that seeks to deploy a private cloud and already has a database of employees and users in an LDAP system. Using this as the authentication authority, requests to the Identity service are delegated to the LDAP system, which will then authorize or deny based on its policies. Upon successful authentication, the Identity service then generates a token that is used for access to authorized services.

Note that if the LDAP system has attributes defined for the user such as admin, finance, HR etc, these must be mapped into roles and groups within Identity for use by the various OpenStack services. The `/etc/keystone/keystone.conf` file maps LDAP attributes to Identity attributes.

The Identity service **MUST NOT** be allowed to write to LDAP services used for authentication outside of the OpenStack deployment as this would allow a sufficiently privileged keystone user to make changes to the LDAP directory. This would allow privilege escalation within the wider organization or facilitate unauthorized access to other information and resources. In such a deployment, user provisioning would be out of the realm of the OpenStack deployment.

Note

There is an OpenStack Security Note (OSSN) regarding keystone.conf permissions.

There is an OpenStack Security Note (OSSN) regarding potential DoS attacks.

External authentication methods

Organizations may desire to implement external authentication for compatibility with existing authentication services or to enforce stronger authentication policy requirements. Although passwords are the most common form of authentication, they can be compromised through numerous methods, including keystroke logging and password compromise. External authentication services can provide alternative forms of authentication that minimize the risk from weak passwords.

These include:

- Password policy enforcement: Requires user passwords to conform to minimum standards for length, diversity of characters, expiration, or failed login attempts.

- Multi-factor authentication: The authentication service requires the user to provide information based on something they have, such as a one-time password token or X.509 certificate, and something they know, such as a password.

- Kerberos

Authorization

The Identity service supports the notion of groups and roles. Users belong to groups while a group has a list of roles. OpenStack services reference the roles of the user attempting to access the service. The OpenStack policy enforcer middleware takes into consideration the policy rule associated with each resource then the user's group/roles and association to determine if access is allowed to the requested resource.

The policy enforcement middleware enables fine-grained access control to OpenStack resources. Only admin users can provision new users and have access to various management functionality. The cloud users would only be able to spin up instances, attach volumes, and perform other operational tasks.

Establish formal access control policies

Prior to configuring roles, groups, and users, document your required access control policies for the OpenStack installation. The policies should be consistent with any regulatory or legal requirements for the organization. Future modifications to the access control configuration should be done consistently with the formal policies. The policies should include the conditions and processes for creating, deleting, disabling, and enabling accounts, and for assigning privileges to the accounts. Periodically review the policies and ensure that the configuration is in compliance with approved policies.

Service authorization

Cloud administrators must define a user with the role of admin for each service, as described in the *OpenStack Cloud Administrator Guide*. This ser-

vice account provides the service with the authorization to authenticate users.

The Compute and Object Storage services can be configured to use the Identity service to store authentication information. Other options to store authentication information include the use of the "tempAuth" file, however this should not be deployed in a production environment as the password is displayed in plain text.

The Identity service supports client authentication for TLS which may be enabled. TLS client authentication provides an additional authentication factor, in addition to the user name and password, that provides greater reliability on user identification. It reduces the risk of unauthorized access when user names and passwords may be compromised. However, there is additional administrative overhead and cost to issue certificates to users that may not be feasible in every deployment.

Note

We recommend that you use client authentication with TLS for the authentication of services to the Identity service.

The cloud administrator should protect sensitive configuration files from unauthorized modification. This can be achieved with mandatory access control frameworks such as SELinux, including `/etc/keystone/keystone.conf` and X.509 certificates.

Client authentication with TLS requires certificates be issued to services. These certificates can be signed by an external or internal certificate authority. OpenStack services check the validity of certificate signatures against trusted CAs by default and connections will fail if the signature is not valid or the CA is not trusted. Cloud deployers may use self-signed certificates. In this case, the validity check must be disabled or the certificate should be marked as trusted. To disable validation of self-signed certificates, set `insecure=False` in the `[filter:authtoken]` section in the `/etc/nova/api.paste.ini` file. This setting also disables certificates for other components.

Administrative users

We recommend that admin users authenticate using Identity service and an external authentication service that supports 2-factor authentication, such as a certificate. This reduces the risk from passwords that may be compromised. This recommendation is in compliance with NIST 800-53

IA-2(1) guidance in the use of multi-factor authentication for network access to privileged accounts.

End users

The Identity service can directly provide end-user authentication, or can be configured to use external authentication methods to conform to an organization's security policies and requirements.

Policies

Each OpenStack service has a policy file in JSON format, called `policy.json`. The policy file specifies rules, and the rule that governs each resource. A resource could be API access, the ability to attach to a volume, or to fire up instances.

The policies can be updated by the cloud administrator to further control access to the various resources. The middleware could also be further customized. Note that your users must be assigned to groups/roles that you refer to in your policies.

Below is a snippet of the Block Storage service `policy.json` file.

```
{
    "context_is_admin": "role:admin",
    "admin_or_owner":  "is_admin:True or project_id:
%(project_id)s",
    "default": "rule:admin_or_owner",

    "admin_api": "is_admin:True",

    "volume:create": "",
    "volume:get_all": "",
    "volume:get_volume_metadata": "",
    "volume:get_volume_admin_metadata": "rule:admin_api",
    "volume:delete_volume_admin_metadata": "rule:admin_api",
    "volume:update_volume_admin_metadata": "rule:admin_api",
    "volume:get_snapshot": "",
    "volume:get_all_snapshots": "",
    "volume:extend": "",
    "volume:update_readonly_flag": "",
    "volume:retype": "",

    "volume_extension:types_manage": "rule:admin_api",
    "volume_extension:types_extra_specs": "rule:admin_api",
    "volume_extension:volume_type_encryption": "rule:admin_api",
    "volume_extension:volume_encryption_metadata":
 "rule:admin_or_owner",
```

```
    "volume_extension:extended_snapshot_attributes": "",
    "volume_extension:volume_image_metadata": "",

    "volume_extension:quotas:show": "",
    "volume_extension:quotas:update": "rule:admin_api",
    "volume_extension:quota_classes": "",

    "volume_extension:volume_admin_actions:reset_status":
"rule:admin_api",
    "volume_extension:snapshot_admin_actions:reset_status":
"rule:admin_api",
    "volume_extension:backup_admin_actions:reset_status":
"rule:admin_api",
    "volume_extension:volume_admin_actions:force_delete":
"rule:admin_api",
    "volume_extension:volume_admin_actions:force_detach":
"rule:admin_api",
    "volume_extension:snapshot_admin_actions:force_delete":
"rule:admin_api",
    "volume_extension:volume_admin_actions:migrate_volume":
"rule:admin_api",

"volume_extension:volume_admin_actions:migrate_volume_completion":
"rule:admin_api",

    "volume_extension:volume_host_attribute": "rule:admin_api",
    "volume_extension:volume_tenant_attribute":
"rule:admin_or_owner",
    "volume_extension:volume_mig_status_attribute":
"rule:admin_api",
    "volume_extension:hosts": "rule:admin_api",
    "volume_extension:services": "rule:admin_api",

    "volume_extension:volume_manage": "rule:admin_api",
    "volume_extension:volume_unmanage": "rule:admin_api",

    "volume:services": "rule:admin_api",

    "volume:create_transfer": "",
    "volume:accept_transfer": "",
    "volume:delete_transfer": "",
    "volume:get_all_transfers": "",

    "volume_extension:replication:promote": "rule:admin_api",
    "volume_extension:replication:reenable": "rule:admin_api",

    "backup:create" : "",
    "backup:delete": "",
    "backup:get": "",
    "backup:get_all": "",
    "backup:restore": "",
```

```
    "backup:backup-import": "rule:admin_api",
    "backup:backup-export": "rule:admin_api",


 "snapshot_extension:snapshot_actions:update_snapshot_status":
 "",

    "consistencygroup:create" : "group:nobody",
    "consistencygroup:delete": "group:nobody",
    "consistencygroup:get": "group:nobody",
    "consistencygroup:get_all": "group:nobody",

    "consistencygroup:create_cgsnapshot" : "",
    "consistencygroup:delete_cgsnapshot": "",
    "consistencygroup:get_cgsnapshot": "",
    "consistencygroup:get_all_cgsnapshots": "",

    "scheduler_extension:scheduler_stats:get_pools" :
 "rule:admin_api"
}
```

Note the **default** rule specifies that the user must be either an admin or the owner of the volume. It essentially says only the owner of a volume or the admin may create/delete/update volumes. Certain other operations such as managing volume types are accessible only to admin users.

Tokens

Once a user is authenticated a token is generated for authorization and access to an OpenStack environment. A token can have a variable life span; however since the release of OpenStack Icehouse, the default value for expiry has been reduced to one hour. The recommended expiry value should be set to a lower value that allows enough time for internal services to complete tasks. In the event that the token expires before tasks complete, the cloud may become unresponsive or stop providing services. An example of expended time during use would be the time needed by the Compute service to transfer a disk image onto the hypervisor for local caching.

The following example shows a PKI token. Note that token id values are typically 3500 bytes. In this example, the value has been truncated.

```
{
    "token": {
        "expires": "2013-06-26T16:52:50Z",
        "id": "MIIKXAY...",
        "issued_at": "2013-06-25T16:52:50.622502",
        "tenant": {
            "description": null,
            "enabled": true,
            "id": "912426c8f4c04fb0a07d2547b0704185",
            "name": "demo"
        }
    }
}
```

The token is often passed within the structure of a larger context of an Identity service response. These responses also provide a catalog of the various OpenStack services. Each service is listed with its name, access endpoints for internal, admin, and public access.

The Identity service supports token revocation. This manifests as an API to revoke a token, to list revoked tokens and individual OpenStack services that cache tokens to query for the revoked tokens and remove them from their cache and append the same to their list of cached revoked tokens.

Future

Domains are high-level containers for projects, users and groups. As such, they can be used to centrally manage all keystone-based identity components. With the introduction of account domains, server, storage and other resources can now be logically grouped into multiple projects (previously called tenants) which can themselves be grouped under a master account-like container. In addition, multiple users can be managed within an account domain and assigned roles that vary for each project.

The Identity V3 API supports multiple domains. Users of different domains may be represented in different authentication back ends and even have different attributes that must be mapped to a single set of roles and privileges, that are used in the policy definitions to access the various service resources.

Where a rule may specify access to only admin users and users belonging to the tenant, the mapping may be trivial. In other scenarios the cloud administrator may need to approve the mapping routines per tenant.

Federated Identity

Federated Identity is a mechanism to establish trusts between Identity Providers and Service Providers (SP), in this case, between Identity Providers and the services provided by an OpenStack Cloud.

Federated Identity provides a way to securely use existing credentials to access cloud resources such as servers, volumes, and databases, across multiple endpoints provided in multiple authorized clouds using a single set of credentials, without having to provision additional identities or log in multiple times. The credential is maintained by the user's Identity Provider.

Some important definitions:

***Service Provider* (SP)**	A system entity that provides services to principals or other system entities, in this case, OpenStack Identity is the Service Provider.
***Identity Provider* (IdP)**	A directory service, such as LDAP, RADIUS and Active Directory, which allows users to login with a user name and password, is a typical source of authentication tokens (e.g. passwords) at an identity provider.
SAML assertion	Contains information about a user as provided by an IdP. It is an indication that a user has been authenticated.
Mapping	Adds a set of rules to map Federation protocol attributes to Identity API objects. An Identity Provider has exactly one mapping specified per protocol.
Protocol	Contains information that dictates which Mapping rules to use for an incoming request made by an IdP. An IdP may support multiple protocols. There are three major protocols for federated identity: OpenID, SAML, and OAuth.
Unscoped token	Allows a user to authenticate with the Identity service to exchange the un-

	scoped token for a scoped token, by providing a project ID or a domain ID.
Scoped token	Allows a user to use all OpenStack services apart from the Identity service.

Why use Federated Identity?

- Provisioning new identities often incurs some security risk. It is difficult to secure credential storage and to deploy it with proper policies. A common identity store is useful as it can be set up properly once and used in multiple places. With Federated Identity, there is no longer a need to provision user entries in Identity service, since the user entries already exist in the IdP's databases.

  This does introduce new challenges around protecting that identity. However, this is a worthwhile tradeoff given the greater control, and fewer credential databases that come with a centralized common identity store.

- It is a burden on the clients to deal with multiple tokens across multiple cloud service providers. Federated Identity provides single sign on to the user, who can use the credentials provided and maintained by the user's IdP to access many different services on the Internet.

- Users spend too much time logging in or going through 'Forget Password' workflows. Federated identity allows for single sign on, which is easier and faster for users and requires fewer password resets. The IdPs manage user identities and passwords so OpenStack does not have to.

- Too much time is spent administering identities in various service providers.

- The best test of interoperability in the cloud is the ability to enable a user with one set of credentials in an IdP to access multiple cloud services. Organizations, each using its own IdP can easily allow their users to collaborate and quickly share the same cloud services.

- Removes a blocker to cloud brokering and multi-cloud workload management. There is no need to build additional authentication mechanisms ito authenticate users, since the IdPs take care of authenticating their own users using whichever technologies they deem to be appropriate. In most organizations, multiple authentication technologies are already in use.

Configuring Identity service for Federation

Federated users are not mirrored in the Identity service back end (for example, using the SQL driver). The external IdP is responsible for authenticating users, and communicates the result of the authentication to Identity service using SAML assertions. Identity service maps the SAML assertions to keystone user groups and assignments created in Identity service.

Enabling Federation

To enable Federation, perform the following steps:

1. Run the Identity service under Apache, instead of using **keystone-all**.

   a. Enable TLS support. Install `mod_nss` according to your distribution, then apply the following patch and restart HTTPD:

```
--- /etc/httpd/conf.d/nss.conf.orig 2012-03-29 12:59:06.
319470425 -0400
+++ /etc/httpd/conf.d/nss.conf      2012-03-29 12:19:38.
862721465 -0400
@@ -17,7 +17,7 @@
# Note: Configurations that use IPv6 but not IPv4-mapped
 addresses need two
#       Listen directives: "Listen [::]:8443" and
 "Listen 0.0.0.0:443"
#
-Listen 8443
+Listen 443

##
##  SSL Global Context
@@ -81,7 +81,7 @@
## SSL Virtual Host Context
##

-<virtualhost _default_:8443="">
+<virtualhost _default_:443="">

#   General setup for the virtual host
 #DocumentRoot "/etc/httpd/htdocs"
</virtualhost></virtualhost>
```

   b. If you have a firewall in place, configure it to allow TLS traffic. For example:

```
-A INPUT -m state --state NEW -m tcp -p tcp --dport 443
 -j ACCEPT
```

Note this needs to be added before your reject all rule which might be:

```
-A INPUT -j REJECT --reject-with icmp-host-prohibited
```

c. Copy the `httpd/wsgi-keystone.conf` file to the appropriate location for your Apache server, for example, `/etc/httpd/conf.d/wsgi-keystone.conf` file.

d. Create the directory `/var/www/cgi-bin/keystone/`. Then link the files `main` and `admin` to the `keystone.py` file in this directory.

For a distribution appropriate place, it should probably be copied to `/usr/share/openstack/keystone/httpd/keystone.py`.

Note

This path is Ubuntu-specific. For other distributions, replace with appropriate path.

e. If you are running with SELinux enabled ensure that the file has the appropriate SELinux context to access the linked file. For example, if you have the file in `/var/www/cgi-bin` location, you can do this by running:

```
# restorecon /var/www/cgi-bin
```

Adding it in a different location requires you set up your SELinux policy accordingly.

f. Make sure you use either the SQL or the `memcached` driver for tokens, otherwise the tokens will not be shared between the processes of the Apache HTTPD server.

For SQL, in `/etc/keystone/keystone.conf` , set:

```
[token]
driver = keystone.token.backends.sql.Token
```

For `memcached`, in `/etc/keystone/keystone.conf`, set:

```
[token]
driver = keystone.token.backends.memcache.Token
```

In both cases, all servers that are storing tokens need a shared back end. This means either that both point to the same database server, or both point to a common memcached instance.

g. Install Shibboleth:

```
# apt-get install libapache2-mod-shib2
```

Note

The `apt-get` command is Ubuntu specific. For other distributions, replace with appropriate command.

h. Configure the Identity service virtual host and adjust the config to properly handle SAML2 workflow.

Add `WSGIScriptAlias` directive to your vhost configuration:

```
WSGIScriptAliasMatch ^(/v3/OS-FEDERATION/
identity_providers/.*?/protocols/.*?/auth)$ /var/www/
keystone/main/$1
```

i. Add two `<Location>` directives to the `wsgi-keystone.conf` file:

```
<Location /Shibboleth.sso>
SetHandler shib
</Location>

<LocationMatch /v3/OS-FEDERATION/identity_providers/.*?/
protocols/saml2/auth>
ShibRequestSetting requireSession 1
AuthType shibboleth
ShibRequireAll On
ShibRequireSession On
ShibExportAssertion Off
Require valid-user
</LocationMatch>
```

Note

The option `saml2` may be different in your deployment, but do not use a wildcard value. Otherwise every Federated protocol will be handled by Shibboleth.

The `ShibRequireSession` rule is invalid in Apache 2.4 or newer and should be dropped in that specific setup.

j. Enable the Identity service virtual host:

```
# a2ensite wsgi-keystone.conf
```

k. Enable the `ssl` and `shib2` modules:

```
# a2enmod ssl
# a2enmod shib2
```

l. Restart Apache:

```
# service apache2 restart
```

Note

The `service apache2 restart` command is Ubuntu-specific. For other distributions, replace with appropriate command.

2. Configure Apache to use a Federation capable authentication method.

   a. Once you have your Identity service virtual host ready, configure Shibboleth and upload your metadata to the Identity Provider.

   If new certificates are required, they can be easily created by executing:

   ```
   $ shib-keygen -y NUMBER_OF_YEARS
   ```

   The newly created file will be stored under `/etc/shibboleth/sp-key.pem`

   b. Upload your Service Provider's metadata file to your Identity Provider.

   c. Configure your Service Provider by editing `/etc/shibboleth/shibboleth2.xml`.

   For more information, see *Shibboleth Service Provider Configuration*.

d. Identity service enforces `external` authentication when environment variable REMOTE_USER is present so make sure Shibboleth does not set the REMOTE_USER environment variable. To do so, scan through the `/etc/shibboleth/shibboleth2.xml` configuration file and remove the REMOTE_USER directives.

e. Examine your attributes map in the `/etc/shibboleth/attributes-map.xml` file and adjust your requirements if needed. For more information see *Shibboleth Attributes*.

f. Restart the Shibboleth daemon:

```
# service shibd restart
# service apache2 restart
```

3. Enable `OS-FEDERATION` extension:

a. Add the Federation extension driver to the `[federation]` section in the `keystone.conf` file. For example:

```
[federation]
driver = keystone.contrib.federation.backends.sql.
Federation
```

b. Add the saml2 authentication method to the `[auth]` section in `keystone.conf` file:

```
[auth]
methods = external,password,token,saml2
saml2 = keystone.auth.plugins.saml2.Saml2
```

Note

The `external` method should be dropped to avoid any interference with some Apache and Shibboleth SP setups, where a REMOTE_USER environment variable is always set, even as an empty value.

c. Add the `federation_extension` middleware to the `api_v3` pipeline in the `keystone-paste.ini` file. For example:

```
[pipeline:api_v3]
pipeline = access_log sizelimit url_normalize token_auth
 admin_token_auth
xml_body json_body ec2_extension s3_extension
 federation_extension
service_v3
```

d. Create the Federation extension tables if using the provided SQL back end. For example:

```
$ keystone-manage db_sync --extension federation
```

Ideally, to test that the Identity Provider and the Identity service are communicating, navigate to the protected URL and attempt to sign in. If you get a response back from keystone, even if it is a wrong response, indicates the communication.

Configuring Federation

Now that the Identity Provider and Identity service are communicating, you can start to configure the `OS-FEDERATION` extension.

1. Create Identity groups and assign roles.

   No new users will be added to the Identity back end, but the Identity service requires group-based role assignments to authorize federated users. The Federation mapping function will map the user into local Identity service groups objects, and hence to local role assignments.

   Thus, it is required to create the necessary Identity service groups that correspond to the Identity Provider's groups; additionally, these groups should be assigned roles on one or more projects or domains. For example, groups here refers to the Identity service groups that should be created so that when mapping from the SAML attribute `Employees`, you can map it to a Identity service group `devs`.

   The Identity service administrator can create as many groups as there are SAML attributes, whatever the mapping calls for.

2. Add Identity Providers, Mappings and Protocols.

   To utilize Federation, create the following in the Identity service: Identity Provider, Mapping, Protocol.

Performing Federation authentication

1. Authenticate externally and generate an unscoped token in Identity service.

   To start Federated authentication a user must access the dedicated URL with Identity Provider's and Protocol's identifiers stored with-

in a protected URL. The URL has a format of: `/v3/OS-FEDERATION/identity_providers/{identity_provider}/protocols/{protocol}/auth`.

This instance follows a standard SAML2 authentication procedure, that is, the user will be redirected to the Identity Provider's authentication webpage and be prompted for credentials. After successfully authenticating the user will be redirected to the Service Provider's endpoint. If using a web browser, a token will be returned in XML format. As an alternative to using a web browser, you can use Enhanced Client or Proxy (ECP), which is available in the `keystoneclient` in the Identity service API.

In the returned unscoped token, a list of Identity service groups the user belongs to will be included.

For example, the following URL would be considered protected by `mod_shib` and Apache, as such a request made to the URL would be redirected to the Identity Provider, to start the SAML authentication procedure.

```
# curl -X GET \
-D - http://localhost:5000/v3/OS-FEDERATION/
identity_providers/{identity_provider}/protocols/{protocol}/
auth
```

Note

It is assumed that the `keystone` service is running on port `5000`.

2. Determine accessible resources.

   By using the previously returned token, the user can issue requests to the list projects and domains that are accessible.

   - List projects a federated user can access: `GET /OS-FEDERATION/projects`
   - List domains a federated user can access: `GET /OS-FEDERATION/domains`

   For example,

```
# curl -X GET \
-H "X-Auth-Token: <unscoped token>" http://localhost:5000/
v3/OS-FEDERATION/projects
```

or

```
# curl -X GET \
-H "X-Auth-Token: <unscoped token>" http://localhost:5000/
v3/OS-FEDERATION/domains
```

3. Get a scoped token.

   A federated user may request a scoped token, by using the unscoped token. A project or domain may be specified by either ID or name. An ID is sufficient to uniquely identify a project or domain. For example,

```
# curl -X POST \
-H "Content-Type: application/json" \
-d '{"auth":{"identity":{"methods":["saml2"],"saml2":
{"id":"<unscoped_token_id>"}},"scope":{"project":{"domain":
 {"name": "Default"},"name":"service"}}}}' \
-D - http://localhost:5000/v3/auth/tokens
```

Setting Identity service as Identity Provider

Configuration options

Before attempting to federate multiple Identity service deployments, you must setup certain configuration options in the `keystone.conf` file.

Within the `keystone.conf` assign values to the `[saml]` related fields, for example:

```
[saml]
certfile=/etc/keystone/ssl/certs/ca.pem
keyfile=/etc/keystone/ssl/private/cakey.pem
idp_entity_id=https://keystone.example.com/v3/OS-FEDERATION/
saml2/idp
idp_sso_endpoint=https://keystone.example.com/v3/OS-FEDERATION/
saml2/sso
idp_metadata_path=/etc/keystone/saml2_idp_metadata.xml
```

It is recommended that the following `Organization` configuration options be setup.

```
idp_organization_name=example_company
idp_organization_display_name=Example Corp.
idp_organization_url=example.com
```

It is also recommended the following `Contact` options are set.

```
idp_contact_company=example_company
idp_contact_name=John
idp_contact_surname=Smith
idp_contact_email=jsmith@example.com
idp_contact_telephone=555-55-5555
idp_contact_type=technical
```

Generate metadata

In order to create a trust between the Identity Provider and the Service Provider, metadata must be exchanged. To create metadata for your Identity service, run the **keystone-manage** command and pipe the output to a file. For example:

```
$ keystone-manage saml_idp_metadata > /etc/keystone/
saml2_idp_metadata.xml
```

Note

The file location should match the value of the configuration option `idp_metadata_path` that was assigned in the list of `[saml]` updates.

Create a region for the Service Provider

Create a new region for the service provider, for example, create a new region with an `ID` of *BETA*, and URL of *`https://beta.com/Shibboleth.sso/SAML2/POST`*. This URL will be used when creating a SAML assertion for *BETA*, and signed by the current keystone Identity Provider.

```
$ curl -s -X PUT \
  -H "X-Auth-Token: $OS_TOKEN" \
  -H "Content-Type: application/json" \
  -d '{"region": {"url": "http://beta.com/Shibboleth.sso/SAML2/
POST"}}' \
  http://localhost:5000/v3/regions/BETA | python -mjson.tool
```

Testing it all out

Lastly, if a scoped token and a Service Provider region are presented to keystone, the result will be a full SAML Assertion, signed by the IdP keystone, specifically intended for the Service Provider keystone.

```
$ curl -s -X POST \
  -H "Content-Type: application/json" \
```

```
 -d '{"auth": {"scope": {"region": {"id": "BETA"}}, "identity":
{"token": {"id": "d793d935b9c343f783955cf39ee7dc3c"},
"methods": ["token"]}}}' \
 http://localhost:5000/v3/auth/OS-FEDERATION/saml2
```

At this point the SAML Assertion can be sent to the Service Provider keystone, and a valid OpenStack token, issued by a Service Provider keystone, will be returned.

Future

Currently, the CLI supports the Enhanced Client or Proxy (ECP), (the non-browser) support for `keystoneclient` from an API perspective. So, if you are using the `keystoneclient`, you can create a client instance and use the SAML authorization plugin. There is no support for dashboard available presently. With the upcoming OpenStack releases, Federated Identity should be supported with both CLI and the dashboard.

Checklist

Check-Identity-01: Is user and group ownership of Identity configuration files set to keystone?

Configuration files contain critical parameters and information required for smooth functioning of the component. If an unprivileged user, either intentionally or accidently modifies or deletes any of the parameters or the file itself then it would cause severe availability issue causing denial of service to the other end users. Thus user and group ownership of such critical configuration files must be set to that component owner.

Run the following commands:

```
$ stat -L -c "%U %G" /etc/keystone/keystone.conf | egrep
 "keystone keystone"
$ stat -L -c "%U %G" /etc/keystone/keystone-paste.ini | egrep
 "keystone keystone"
$ stat -L -c "%U %G" /etc/keystone/policy.json | egrep "keystone
 keystone"
$ stat -L -c "%U %G" /etc/keystone/logging.conf | egrep
 "keystone keystone"
$ stat -L -c "%U %G" /etc/keystone/ssl/certs/signing_cert.pem |
 egrep "keystone keystone"
$ stat -L -c "%U %G" /etc/keystone/ssl/private/signing_key.pem |
 egrep "keystone keystone"
$ stat -L -c "%U %G" /etc/keystone/ssl/certs/ca.pem | egrep
 "keystone keystone"
```

Pass: If user and group ownership of all these config files is set to keystone. The above commands show output of keystone keystone.

Fail: If the above commands does not return any output as the user or group ownership might have set to any user other than keystone.

Recommended in: the section called "Internally implemented authentication methods" [66]

Check-Identity-02: Are strict permissions set for Identity configuration files?

Similar to previous check, it is recommended to set strict access permissions for such configuration files.

Run the following commands:

```
$ stat -L -c "%a" /etc/keystone/keystone.conf
$ stat -L -c "%a" /etc/keystone/keystone-paste.ini
$ stat -L -c "%a" /etc/keystone/policy.json
$ stat -L -c "%a" /etc/keystone/logging.conf
$ stat -L -c "%a" /etc/keystone/ssl/certs/signing_cert.pem
$ stat -L -c "%a" /etc/keystone/ssl/private/signing_key.pem
$ stat -L -c "%a" /etc/keystone/ssl/certs/ca.pem
```

Pass: If permissions are set to 640 or stricter.

Fail: If permissions are not set to atleast 640.

Recommended in: the section called "Internally implemented authentication methods" [66]

Check-Identity-03: is SSL enabled for Identity?

OpenStack components communicate with each other using various protocols and the communication might involve sensitive or confidential data. An attacker may try to eavesdrop on the channel in order to get access to sensitive information. Thus all the components must communicate with each other using a secured communication protocol like HTTPS.

Pass: If value of parameter `enable` under `[ssl]` section in `/etc/keystone/keystone.conf` is set to `True`.

Fail: If value of parameter `enable` under `[ssl]` section is not set to `True`.

Recommended in: Chapter 4, "Secure communication" [45]

Check-Identity-04: Does Identity use strong hashing algorithms for PKI tokens?

MD5 is a weak and depreciated hashing algorithm. It can be cracked using bruteforce attack. Identity tokens are sensitive and need to be protected with a stronger hashing algorithm to prevent unauthorized disclosure and subsequent access.

Pass: If value of parameter `hash_algorithm` under `[token]` section in `/etc/keystone/keystone.conf` is set to SHA256.

Fail: If value of parameter `hash_algorithm` under `[token]`section is set to MD5.

Recommended in: the section called "Tokens" [72]

Check-Identity-05: Is value of parameter `max_request_body_size` set to default (114688)?

The parameter `max_request_body_size` defines the maximum body size per request in bytes. If the maximum size is not defined, the attacker could craft an arbitary request of large size causing the service to crash and finally resulting in Denial Of Service attack. Assigning the maximum value ensures that any malicious oversized request gets blocked ensuring continued availability of the component.

Pass: If value of parameter `max_request_body_size` in `/etc/keystone/keystone.conf` is set to default (114688) or some reasonable value based on your environment.

Fail: If value of parameter `max_request_body_size` is not set.

Check-Identity-06: is admin token disabled in `/etc/keystone/keystone.conf`?

Admin token is generally used to bootstrap Identity. This token is the most valuable Identity asset, which could be used to gain cloud admin privileges.

Pass: If `admin_token` under `[DEFAULT]` section in `/etc/keystone/keystone.conf` is disabled. And, AdminTokenAuthMiddleware under `[filter:admin_token_auth]` is deleted from `/etc/keystone/keystone-paste.ini`

Fail: If `admin_token` under [DEFAULT] section is set and AdminTokenAuthMiddleware exists in `keystone-paste.ini`.

7. Dashboard

Horizon is the OpenStack dashboard that provides users a self-service portal to provision their own resources within the limits set by administrators. These include provisioning users, defining instance flavors, uploading VM images, managing networks, setting up security groups, starting instances, and accessing the instances through a console.

The dashboard is based on the Django web framework, therefore secure deployment practices for Django apply directly to horizon. This guide provides a popular set of Django security recommendations. Further information can be found by reading the Django documentation.

The dashboard ships with reasonable default security settings, and has good deployment and configuration documentation.

Basic web server configuration

The dashboard should be deployed as a Web Services Gateway Interface (WSGI) application behind an HTTPS proxy such as Apache or nginx. If Apache is not already in use, we recommend nginx since it is lightweight and easier to configure correctly.

When using nginx, we recommend gunicorn as the WSGI host with an appropriate number of synchronous workers. When using Apache, we recommend `mod_wsgi` to host the dashboard.

HTTPS

Deploy the dashboard behind a secure *HTTPS* server by using a valid, trusted certificate from a recognized certificate authority (CA). Private organization-issued certificates are only appropriate when the root of trust is preinstalled in all user browsers.

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

HTTP Strict Transport Security (HSTS)

It is highly recommended to use HTTP Strict Transport Security (HSTS).

Note

If you are using an HTTPS proxy in front of your web server, rather than using an HTTP server with HTTPS functionality, modify the SECURE_PROXY_SSL_HEADER variable. Refer to the Django documentation for information about modifying the SECURE_PROXY_SSL_HEADER variable.

See the chapter on PKI/SSL Everywhere for more specific recommendations and server configurations for HTTPS configurations, including the configuration of HSTS.

Front end caching

Since the dashboard is rendering dynamic content passed directly from OpenStack API requests, we do not recommend front end caching layers such as varnish. In Django, static media is directly served from Apache or nginx and already benefits from web host caching.

Domain names

Many organizations typically deploy web applications at subdomains of an overarching organization domain. It is natural for users to expect a domain of the form `openstack.example.org`. In this context, there are often many other applications deployed in the same second-level namespace, often serving user-controlled content. This name structure is convenient and simplifies name server maintenance.

We strongly recommend deploying horizon to a *second-level domain*, such as `https://example.com`, and advise against deploying horizon on a *shared subdomain* of any level, for example `https://openstack.example.org` or `https://horizon.openstack.example.org`. We also advise against deploying to bare internal domains like `https://horizon/`. These recommendations are based on the limitations of browser same-origin-policy.

The recommendations in this guide cannot effectively protect users against known attacks if the dashboard is deployed on a domain which also hosts user-generated content, such as scripts, images, or uploads of any kind, even if the user-generated content is on a different subdomain. This approach is used by most major web presences, such as googleusercontent.com, fbcdn.com, github.io, and twimg.com, to ensure that user generated content stays separate from cookies and security tokens.

Additionally, if you decline to follow this recommendation above about second-level domains, it is vital that you avoid the cookie backed session store and employ HTTP Strict Transport Security (HSTS). When deployed on a subdomain, the dashboard's security is only as strong as the weakest application deployed on the same second-level domain.

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.

Django media settings are documented in the Django documentation.

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, we recommend disabling resource compression entirely. Online compression dependencies (less, Node.js) should not be installed on production machines.

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 may 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.

Session back end

Horizon's default session back end (`django.contrib.sessions.backends.signed_cookies`) stores user data in *signed* but *unencrypted* cookies stored in the browser. This approach allows the most simple session back-end scaling since each dashboard instance is stateless, but it comes at the cost of *storing sensitive access tokens in the client browser* and transmitting them with every request. This back end ensures that session data has not been tampered with, but the data itself is not encrypted other than the encryption provided by HTTPS.

If your architecture allows it, we recommend using `django.contrib.sessions.backends.cache` as your session back end with memcache as the cache. Memcache must not be exposed publicly, and should communicate over a secured private channel. If you choose to use the signed cookies back end, refer to the Django documentation understand the security trade-offs.

For further details, see the Django documentation.

Allowed hosts

Configure the ALLOWED_HOSTS setting with the domain or domains where the dashboard is available. Failure to configure this setting (especially if not following the recommendation above regarding second level domains) opens the dashboard to a number of serious attacks. Wild card domains should be avoided.

For further details, see the Django documentation.

Cross Site Request Forgery (CSRF)

Django has dedicated middleware for cross-site request forgery (CSRF). For further details, see the Django documentation.

Dashboard is designed to discourage developers from introducing cross-site scripting vulnerabilities with custom dashboards. However, it is important to audit custom dashboards, especially ones that are JavaScript-heavy for inappropriate use of the `@csrf_exempt` decorator. Dashboards which do not follow these recommended security settings should be carefully evaluated before restrictions are relaxed.

Cookies

Session Cookies should be set to HTTPONLY:

```
SESSION_COOKIE_HTTPONLY = True
```

Never configure CSRF or session cookies to have a wild card domain with a leading dot. Horizon's session and CSRF cookie should be secured when deployed with HTTPS:

```
CSRF_COOKIE_SECURE = True
SESSION_COOKIE_SECURE = True
```

Cross Site Scripting (XSS)

Unlike many similar systems, the OpenStack dashboard allows the entire Unicode character set in most fields. This means developers have less latitude to make escaping mistakes that open attack vectors for cross-site scripting (XSS).

Dashboard provides tools for developers to avoid creating XSS vulnerabilities, but they only work if developers use them correctly. Audit any custom dashboards, paying particular attention to use of the `mark_safe` function, use of `is_safe` with custom template tags, the `safe` template tag, anywhere auto escape is turned off, and any JavaScript which might evaluate improperly escaped data.

Cross Origin Resource Sharing (CORS)

Configure your web server to send a restrictive CORS header with each response, allowing only the dashboard domain and protocol:

```
Access-Control-Allow-Origin: https://example.com/
```

Never allow the wild card origin.

Horizon image upload

We recommend that implementers disable HORIZON_IMAGES_ALLOW_UPLOAD unless they have implemented a plan to prevent resource exhaustion and denial of service.

Upgrading

Django security releases are generally well tested and aggressively backwards compatible. In almost all cases, new major releases of Django are also fully backwards compatible with previous releases. Dashboard implementers are strongly encouraged to run the latest stable release of Django with up-to-date security releases.

Debug

Make sure DEBUG is set to `False` in production. In Django, DEBUG displays stack traces and sensitive web server state information on any exception.

8. Compute

The Compute service (nova) is one of the more complex OpenStack services. It runs in many locations throughout the cloud and interacts with a variety of internal services. For this reason, most of our recommendations regarding best practices for Compute service configuration are distributed throughout this book. We provide specific details in the sections on Management, API Endpoints, Messaging, and Database.

How to select virtual consoles

One decision a cloud architect will need to make regarding Compute service configuration is whether to use *VNC* or *SPICE*. Below we provide some details on the differences between these options.

Virtual Network Computer (VNC)

OpenStack can be configured to provide remote desktop console access to instances for tenants and/or administrators using the Virtual Network Computer (VNC) protocol.

Capabilities

- The OpenStack dashboard (horizon) can provide a VNC console for instances directly on the web page using the HTML5 noVNC client. This requires the `nova-novncproxy` service to bridge from the public network to the management network.
- The **nova** command-line utility can return a URL for the VNC console for access by the `nova` Java VNC client. This requires the `nova-xvpvncproxy` service to bridge from the public network to the management network.

Security considerations

- The `nova-novncproxy` and `nova-xvpvncproxy` services by default open public-facing ports that are token authenticated.
- By default, the remote desktop traffic is not encrypted. TLS can be enabled to encrypt the VNC traffic. Please refer to Introduction to TLS and SSL for appropriate recommendations.

Bibliography

blog.malchuk.ru, OpenStack VNC Security. 2013. Secure Connections to VNC ports

OpenStack Mailing List, [OpenStack] nova-novnc SSL configuration - Havana. 2014. OpenStack nova-novnc SSL Configuration

Redhat.com/solutions, Using SSL Encryption with OpenStack nova-novacproxy. 2014. OpenStack nova-novncproxy SSL encryption

Simple Protocol for Independent Computing Environments (SPICE)

As an alternative to VNC, OpenStack provides remote desktop access to guest virtual machines using the Simple Protocol for Independent Computing Environments (SPICE) protocol.

Capabilities

- SPICE is supported by the OpenStack dashboard (horizon) directly on the instance web page. This requires the `nova-spicehtml5proxy` service.
- The nova command-line utility can return a URL for SPICE console for access by a SPICE-html client.

Limitations

- Although SPICE has many advantages over VNC, the spice-html5 browser integration currently doesn't really allow admins to take advantage of any of the benefits. To take advantage of SPICE features like multi-monitor, USB pass through, etc. admins are recommended to use a standalone SPICE client within the Management Network.

Security considerations

- The `nova-spicehtml5proxy` service by default opens public-facing ports that are token authenticated.
- The functionality and integration are still evolving. We will access the features in the next release and make recommendations.

- As is the case for VNC, at this time we recommend using SPICE from the management network in addition to limiting use to few individuals.

Bibliography

OpenStack Configuration Reference - Havana. SPICE Console. SPICE Console

bugzilla.redhat.com, Bug 913607 - RFE: Support Tunnelling SPICE over websockets. 2013. Red Hat bug 913607

9. Object Storage

OpenStack Object Storage (swift) is a service that provides software that stores and retrieves data over HTTP. Objects (blobs of data) are stored in an organizational hierarchy that offers anonymous read-only access, ACL defined access, or even temporary access. Object Store supports multiple token-based authentication mechanisms implemented via middleware.

Applications store and retrieve data in Object Store via an industry-standard HTTP RESTful API. Back-end components of Object Storage follow the same RESTful model however some of the APIs for managing durability, for example, are kept private to the cluster. For more details on the API see the OpenStack Storage documentation.

For this document the components will be grouped into the following primary groups:

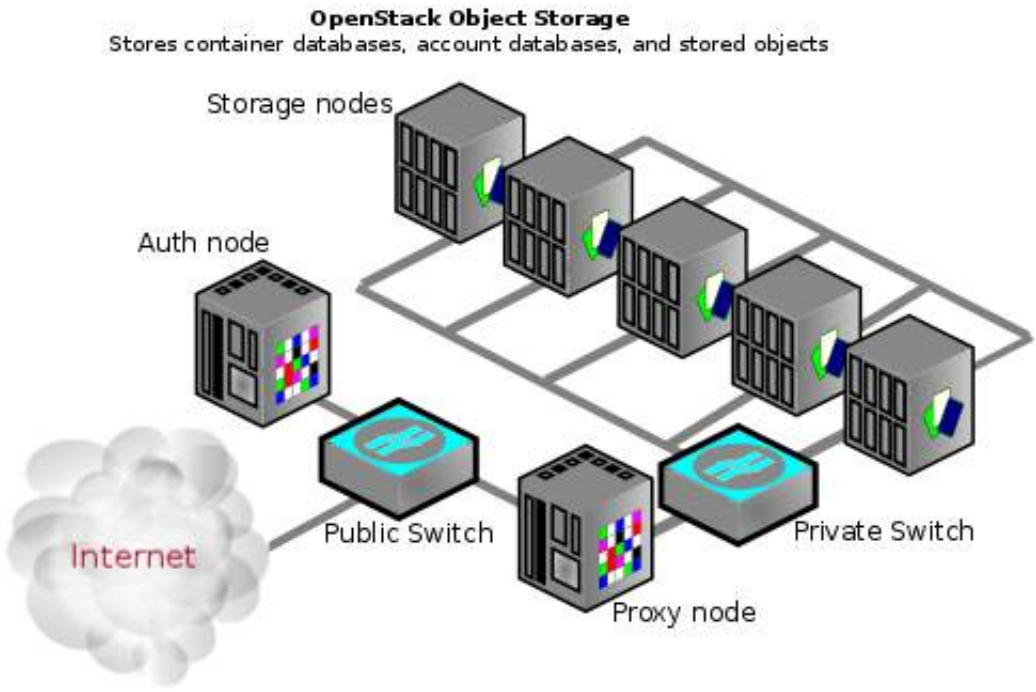
1. Proxy services

2. Auth services

3. Storage services

   - Account service
   - Container service
   - Object service

Figure 9.1. An example diagram from the OpenStack Object Storage Administration Guide (2013)



Note

An Object Storage installation does not have to necessarily be on the Internet and could also be a private cloud with the "Public Switch" being part of the organization's internal network infrastructure.

First thing to secure: the network

Securing the Object Storage service begins with securing the networking component. The rsync protocol is used between storage service nodes to replicate data for high availability. In addition, the proxy service communicates with the storage service when relaying data back and forth between the client end-point and the cloud environment.

Caution

Object Storage does not employ encryption or authentication with inter-node communications. This is why you see a "Private Switch" or private network ([V]LAN) in the architecture diagrams. This data domain should be separate from other OpenStack data networks as well. For further discussion on security domains please see the section called “Security boundaries and threats” [12].

Tip

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

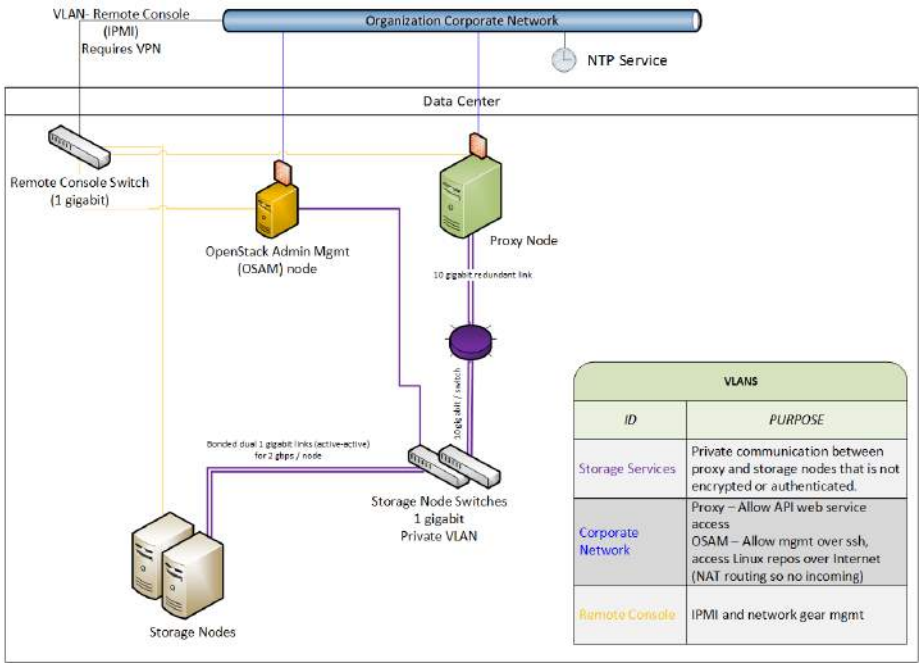
This necessitates that the proxy nodes have dual interfaces (physical or virtual):

1. One as a "public" interface for consumers to reach

2. Another as a "private" interface with access to the storage nodes

The following figure demonstrates one possible network architecture.

Figure 9.2. Object Storage network architecture with a management node (OSAM)



Securing services: general

Run services as non-root user

It is recommended that you configure each Object Storage service to run under a non-root (UID 0) service account. One recommendation is the user name "swift" with the primary group "swift." Object Storage services include, for example, 'proxy-server', 'container-server', 'account-server'. Detailed steps for setup and configuration can be found in the Add Object Storage chapter of the *Installation Guide* in the OpenStack Documentation index. (The link defaults to the Ubuntu version.)

File permissions

The `/etc/swift` directory contains information about the ring topology and environment configuration. The following permissions are recommended:

```
# chown -R root:swift /etc/swift/*
# find /etc/swift/ -type f -exec chmod 640 {} \;
```

```
# find /etc/swift/ -type d -exec chmod 750 {} \;
```

This restricts only root to be able to modify configuration files while allowing the services to read them through their group membership in the 'swift' group.

Securing storage services

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

Service name	Port	Type
Account service	6002	TCP
Container service	6001	TCP
Object service	6000	TCP
Rsync[a]	873	TCP

[a] If ssync is used instead of rsync, the Object service port is used for maintaining durability.

Authentication does not take place at the storage nodes. If someone was able to connect to a storage node on one of these ports they could access or modify data without authentication. In order to secure against this issue you should follow the recommendations given previously about using a private storage network.

Object Storage "account" terminology

An Object Storage "account" is not a user account or credential. The following explains the relations:

OpenStack Object Storage account	Collection of containers; not user accounts or authentication. Which users are associated with the account and how they may access it depends on the authentication system used. See the section called “Object Storage authentication” [105].
OpenStack Object Storage containers	Collection of objects. Metadata on the container is available for ACLs. The meaning of ACLs is dependent on the authentication system used.
OpenStack Object Storage objects	The actual data objects. ACLs at the object level are also possible with metadata and are dependent on the authentication system used.

Tip

Another way of thinking about the above would be: A single shelf (account) holds zero or more buckets (containers) which

each hold zero or more objects. A garage (Object Storage cluster) may have multiple shelves (accounts) with each shelf belonging to zero or more users.

At each level you may have ACLs that dictate who has what type of access. ACLs are interpreted based on what authentication system is in use. The two most common types of authentication providers used are Identity service (keystone) and TempAuth. Custom authentication providers are also possible. Please see the section called "Object Storage authentication" [105] for more information.

Securing proxy services

A proxy node should have at least two interfaces (physical or virtual): one public and one private. Firewalls or service binding might 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.

HTTP listening port

You should configure your web service as a non-root (no UID 0) user such as "swift" mentioned before. The use of a port greater than 1024 is required to make this easy and avoid running any part of the web container as root. Doing so is not a burden as end-point clients are not typically going to type in the URL manually into a web browser to browse around in the object storage. Additionally, for clients using the HTTP REST API and performing authentication they will normally automatically grab the full REST API URL they are to use as provided by the authentication response. OpenStack's REST API allows for a client to authenticate to one URL and then be told to use a completely different URL for the actual service. Example: Client authenticates to `https://identity.cloud.example.org:55443/v1/auth` and gets 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`.

The method for configuring your web server to start and run as a non-root user varies by web server and OS.

Load balancer

If the option of using Apache is not feasible or for performance you wish to offload your TLS work you may employ a dedicated network device load balancer. This is also the 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 were to occur the attacker could gain access to end-point client or cloud administrator credentials and access the cloud data.

The authentication service you use, such as Identity service (keystone) or TempAuth, will determine how you configure a different URL in the responses to end-point clients so they use your load balancer instead of an individual proxy node.

Object Storage authentication

Object Storage uses a WSGI model to provide for a middleware capability that not only provides general extensibility but is also used for authentication of end-point clients. The authentication provider defines what roles and user types exist. Some use traditional user name and password credentials while others may leverage API key tokens or even client-side x.509 certificates. Custom providers can be integrated in 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.

TempAuth

TempAuth is the default authentication for Object Storage. In contrast to Identity it stores the user accounts, credentials, and metadata in object storage itself. More information can be found in the section The Auth System of the Object Storage (swift) documentation.

Keystone

Keystone is the commonly used Identity provider in OpenStack. It may also be used for authentication in Object Storage. Coverage of securing keystone is already provided in Chapter 6, “Identity” [65].

Other notable items

In `/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.

10. Case studies: Identity management

Earlier in the section called "Introduction to case studies" [21] we introduced the Alice and Bob case studies where Alice is deploying a private government cloud and Bob is deploying a public cloud each with different security requirements. Here we discuss how Alice and Bob would address configuration of OpenStack core services. These include the Identity service, dashboard, and Compute services. Alice will be concerned with integration into the existing government directory services, while Bob will need to provide access to the public.

Alice's private cloud

Alice's enterprise has a well-established directory service with two-factor authentication for all users. She configures the Identity service to support an external authentication service supporting authentication with government-issued access cards. She also uses an external LDAP server to provide role information for the roles that are integrated with the access control policy. Due to FedRAMP compliance requirements, Alice implements two-factor authentication on the management network for all administrator access.

Alice also deploys the dashboard to manage many aspects of the cloud. She deploys the dashboard with HSTS to ensure that only HTTPS is used. The dashboard resides within an internal subdomain of the private network domain name system.

Alice decides to use SPICE instead of VNC for the virtual console. She wants to take advantage of the emerging capabilities in SPICE.

Bob's public cloud

Because Bob must support authentication for the general public, he decides to use authentication based on a user name and password. He has concerns about brute force attacks attempting to crack user passwords, so he also uses an external authentication extension that throttles the num-

ber of failed login attempts. Bob's management network is separate from the other networks within his cloud, but can be reached from his corporate network through ssh. As recommended earlier, Bob requires administrators to use two-factor authentication on the Management network to reduce the risk of compromised administrator passwords.

Bob also deploys the dashboard to manage many aspects of the cloud. He deploys the dashboard with HSTS to ensure that only HTTPS is used. He has ensured that the dashboard is deployed on a second-level domain due to the limitations of the same-origin policy. He also disables `HORIZON_IMAGES_ALLOW_UPLOAD` to prevent resource exhaustion.

Bob decides to use VNC for his virtual console for its maturity and security features.

11. Networking

OpenStack Networking enables the end-user or tenant to define, utilize, and consume networking resources. OpenStack Networking provides a tenant-facing API for defining network connectivity and IP addressing for instances in the cloud in addition to orchestrating the network configuration. With the transition to an API-centric networking service, cloud architects and administrators should take into consideration best practices to secure physical and virtual network infrastructure and services.

OpenStack Networking was designed with a plug-in architecture that provides extensibility of the API through open source community or third-party services. As you evaluate your architectural design requirements, it is important to determine what features are available in OpenStack Networking core services, any additional services that are provided by third-party products, and what supplemental services are required to be implemented in the physical infrastructure.

This section is a high-level overview of what processes and best practices should be considered when implementing OpenStack Networking. We will talk about the current state of services that are available, what future services will be implemented, and the current limitations in this project.

Networking architecture

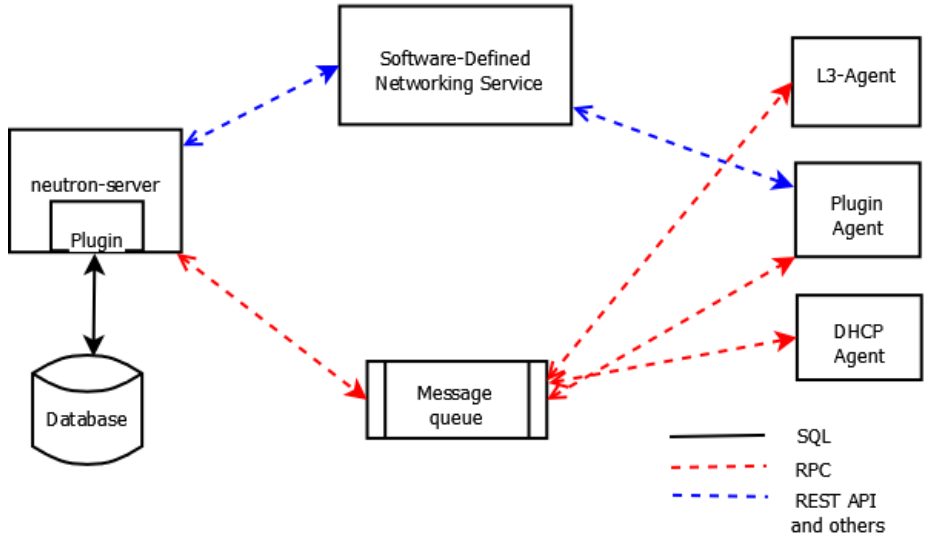
OpenStack Networking is a standalone service that often deploys several processes across a number of nodes. These processes interact with each other and other OpenStack services. The main process of the OpenStack Networking service is `neutron-server`, a Python daemon that exposes the OpenStack Networking API and passes tenant requests to a suite of plug-ins for additional processing.

The OpenStack Networking components are:

neutron server (`neutron-server` and `neutron-*-plugin`) — This service runs on the network node to service the Networking API and its

	extensions. It also enforces the network model and IP addressing of each port. The neutron-server and plugin agents require access to a database for persistent storage and access to a message queue for inter-communication.
plugin agent (`neutron-*-agent`)	Runs on each compute node to manage local virtual switch (vswitch) configuration. The plug-in that you use determine which agents run. This service requires message queue access. This service requires message queue access and depends on the plugin used.
DHCP agent (`neutron-dhcp-agent`)	Provides DHCP services to tenant networks. This agent is the same across all plug-ins and is responsible for maintaining DHCP configuration. The `neutron-dhcp-agent` requires message queue access.
L3 agent (`neutron-l3-agent`)	Provides L3/NAT forwarding for external network access of VMs on tenant networks. Requires message queue access. *Optional depending on plug-in.*
network provider services (SDN server/services)	Provides additional networking services to tenant networks. These SDN services may interact with `neutron-server`, `neutron-plugin`, and plugin-agents through communication channels such as REST APIs.

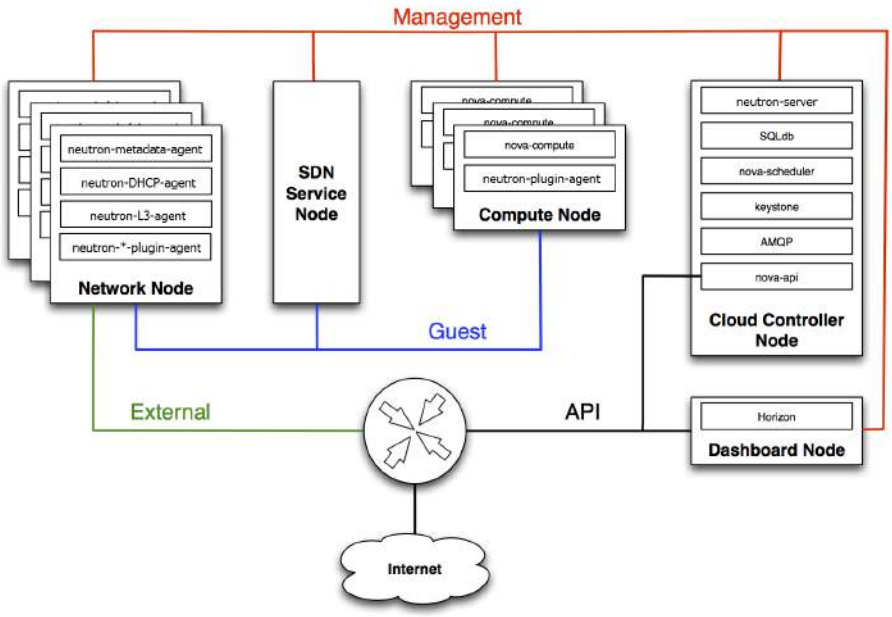
The following figure shows an architectural and networking flow diagram of the OpenStack Networking components:



OpenStack Networking service placement on physical servers

This guide focuses on a standard architecture that includes a *cloud controller* host, a *network* host, and a set of *compute* hypervisors for running VMs.

Network connectivity of physical servers



A standard OpenStack Networking setup has up to four distinct physical data center networks:

Management network — Used for internal communication between OpenStack Components. The IP addresses on this network should be reachable only within the data center and is considered the Management Security Domain.

Guest network — Used for VM data communication within the cloud deployment. The IP addressing requirements of this network depend on the OpenStack Networking plug-in in use and the network configuration choices of the virtual networks made by the tenant. This network is considered the Guest Security Domain.

External network — Used to provide VMs with Internet access in some deployment scenarios. The IP addresses on this network should be reachable by anyone on the Internet. This network is considered to be in the Public Security Domain.

API network	Exposes all OpenStack APIs, including the OpenStack Networking API, to tenants. The IP addresses on this network should be reachable by anyone on the Internet. This may be the same network as the external network, as it is possible to create a subnet for the external network that uses IP allocation ranges to use only less than the full range of IP addresses in an IP block. This network is considered the Public Security Domain.

For additional information see the Networking chapter in the *OpenStack Cloud Administrator Guide*.

Networking services

In the initial architectural phases of designing your OpenStack Network infrastructure it is important to ensure appropriate expertise is available to assist with the design of the physical networking infrastructure, to identify proper security controls and auditing mechanisms.

OpenStack Networking adds a layer of virtualized network services which gives tenants the capability to architect their own virtual networks. Currently, these virtualized services are not as mature as their traditional networking counterparts. Consider the current state of these virtualized services before adopting them as it dictates what controls you may have to implement at the virtualized and traditional network boundaries.

L2 isolation using VLANs and tunneling

OpenStack Networking can employ two different mechanisms for traffic segregation on a per tenant/network combination: VLANs (IEEE 802.1Q tagging) or L2 tunnels using GRE encapsulation. The scope and scale of your OpenStack deployment determines which method you should utilize for traffic segregation or isolation.

VLANs

VLANs are realized as packets on a specific physical network containing IEEE 802.1Q headers with a specific VLAN ID (VID) field value. VLAN networks sharing the same physical network are isolated from each other at L2, and can even have overlapping IP address spaces. Each distinct physical network supporting VLAN networks is treated as a separate VLAN trunk, with a distinct space of VID values. Valid VID values are 1 through 4094.

VLAN configuration complexity depends on your OpenStack design requirements. In order to allow OpenStack Networking to efficiently use VLANs, you must allocate a VLAN range (one for each tenant) and turn each compute node physical switch port into a VLAN trunk port.

Note

NOTE: If you intend for your network to support more than 4094 tenants VLAN is probably not the correct option for you as multiple 'hacks' are required to extend the VLAN tags to more than 4094 tenants.

L2 tunneling

Network tunneling encapsulates each tenant/network combination with a unique "tunnel-id" that is used to identify the network traffic belonging to that combination. The tenant's L2 network connectivity is independent of physical locality or underlying network design. By encapsulating traffic inside IP packets, that traffic can cross Layer-3 boundaries, removing the need for preconfigured VLANs and VLAN trunking. Tunneling adds a layer of obfuscation to network data traffic, reducing the visibility of individual tenant traffic from a monitoring point of view.

OpenStack Networking currently supports both GRE and VXLAN encapsulation.

The choice of technology to provide L2 isolation is dependent upon the scope and size of tenant networks that will be created in your deployment. If your environment has limited VLAN ID availability or will have a large number of L2 networks, it is our recommendation that you utilize tunneling.

Network services

The choice of tenant network isolation affects how the network security and control boundary is implemented for tenant services. The following additional network services are either available or currently under development to enhance the security posture of the OpenStack network architecture.

Access control lists

OpenStack Compute supports tenant network traffic access controls directly when deployed with the legacy nova-network service, or may defer access control to the OpenStack Networking service.

Note, legacy nova-network security groups are applied to all virtual interface ports on an instance using iptables.

Security groups allow administrators and tenants the ability to specify the type of traffic, and direction (ingress/egress) that is allowed to pass through a virtual interface port. Security groups rules are stateful L2-L4 traffic filters.

When using the Networking service, we recommend that you enable security groups in this service and disable it in the Compute service.

L3 routing and NAT

OpenStack Networking routers can connect multiple L2 networks, and can also provide a *gateway* that connects one or more private L2 networks to a shared *external* network, such as a public network for access to the Internet.

The L3 router provides basic Network Address Translation (NAT) capabilities on *gateway* ports that uplink the router to external networks. This router SNATs (Static NAT) all traffic by default, and supports floating IPs, which creates a static one-to-one mapping from a public IP on the external network to a private IP on one of the other subnets attached to the router.

It is our recommendation to leverage per tenant L3 routing and Floating IPs for more granular connectivity of tenant VMs.

Quality of Service (QoS)

The ability to set QoS on the virtual interface ports of tenant instances is a current deficiency for OpenStack Networking. The application of QoS for traffic shaping and rate-limiting at the physical network edge device is insufficient due to the dynamic nature of workloads in an OpenStack deployment and can not be leveraged in the traditional way. QoS-as-a-Service (QoSaaS) is currently in development for the OpenStack Networking Icehouse release as an experimental feature. QoSaaS is planning to provide the following services:

- Traffic shaping through DSCP markings
- Rate-limiting on a per port/network/tenant basis.
- Port mirroring (through open source or third-party plug-ins)
- Flow analysis (through open source or third-party plug-ins)

Tenant traffic port mirroring or Network Flow monitoring is currently not an exposed feature in OpenStack Networking. There are third-party plug-in extensions that do provide Port Mirroring on a per port/network/tenant basis. If Open vSwitch is used on the networking hypervisor, it is possible to enable sFlow and port mirroring, however it will require some operational effort to implement.

Load balancing

Another feature in OpenStack Networking is Load-Balancer-as-a-service (LBaaS). The LBaaS reference implementation is based on HA-Proxy. There are third-party plug-ins in development for extensions in OpenStack Networking to provide extensive L4-L7 functionality for virtual interface ports.

Firewalls

FW-as-a-Service (FWaaS) is considered an experimental feature for the Kilo release of OpenStack Networking. FWaaS addresses the need to manage and leverage the rich set of security features provided by typical firewall products which are typically far more comprehensive than what is currently provided by security groups. Both Freescale and Intel developed third-party plug-ins as extensions in OpenStack Networking to support this component in the Kilo release. Documentation for administration of FWaaS is located at

It is critical during the design of an OpenStack Networking infrastructure to understand the current features and limitations of network services that are available. Understanding where the boundaries of your virtual and physical networks will help you add the required security controls in your environment.

Network services extensions

A list of known plug-ins provided by the open source community or by SDN companies that work with OpenStack Networking is available at OpenStack Neutron Plugins and Drivers wiki page.

Networking services limitations

OpenStack Networking has the following known limitations:

Overlapping IP addresses If nodes that run either `neutron-l3-agent` or `neutron-dhcp-agent`

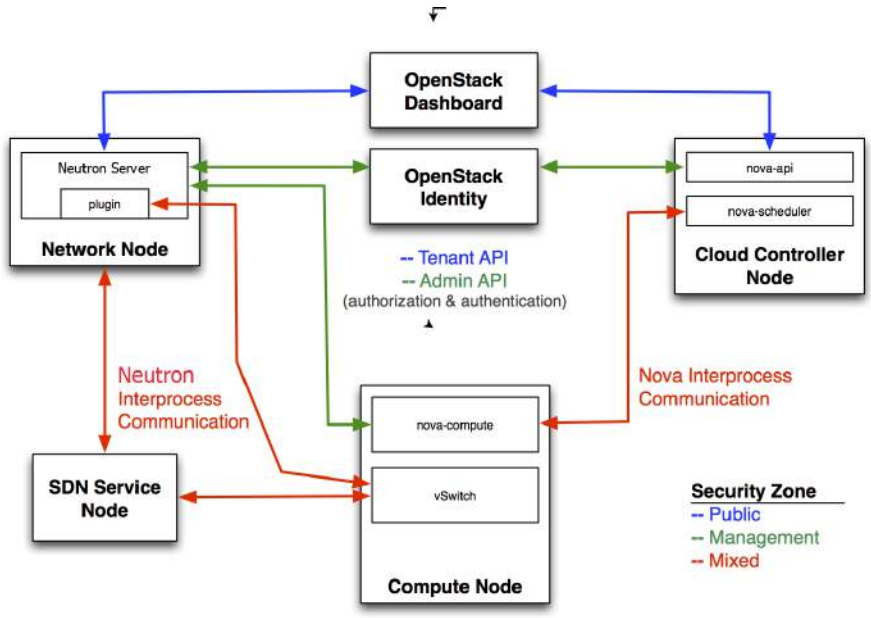
	use overlapping IP addresses, those nodes must use Linux network namespaces. By default, the DHCP and L3 agents use Linux network namespaces. However, if the host does not support these namespaces, run the DHCP and L3 agents on different hosts. If network namespace support is not present, a further limitation of the L3 agent is that only a single logical router is supported.
Multi-host DHCP-agent	OpenStack Networking supports multiple L3 and DHCP agents with load balancing. However, tight coupling of the location of the virtual machine is not supported.
No IPv6 support for L3 agents	The neutron-l3-agent, used by many plug-ins to implement L3 forwarding, supports only IPv4 forwarding.

Securing OpenStack Networking services

To secure OpenStack Networking, you must understand how the workflow process for tenant instance creation needs to be mapped to security domains.

There are four main services that interact with OpenStack Networking. In a typical OpenStack deployment these services map to the following security domains:

- OpenStack dashboard: Public and management
- OpenStack Identity: Management
- OpenStack compute node: Management and guest
- OpenStack network node: Management, guest, and possibly public depending upon neutron-plugin in use.
- SDN services node: Management, guest and possibly public depending upon product used.



To isolate sensitive data communication between the OpenStack Networking services and other OpenStack core services, configure these communication channels to only allow communication over an isolated management network.

OpenStack Networking service configuration

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 neutron.conf file as shown:

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

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

Restrict DB and RPC communication of the OpenStack Networking services

Various components of the OpenStack Networking services use either the messaging queue or database connections to communicate with other components in OpenStack Networking.

It is recommended that you follow the guidelines provided in the section called "Database authentication and access control" [144] for all components which require direct DB connections.

It is recommended that you follow the guidelines provided in the section called "Queue authentication and access control" [125] for all components which require RPC communication.

Networking services security best practices

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

Tenant network services workflow

OpenStack Networking provides users self services of network resources and configurations. 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.

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 tenant networking methods and objects. It is important that cloud architects and operators evaluate their design and use cases in providing users and tenants the ability to create, update, and destroy available network resources as it has a tangible effect on tenant network availability, network security, and overall OpenStack security. For a more detailed explanation of OpenStack Networking policy definition, please refer to the Authentication and authorization section in the *OpenStack Cloud Administrator Guide*.

Note

It is important to review the default networking resource policy and modify the policy appropriately for your security posture.

If your deployment of OpenStack provides multiple external access points into different security domains it is important that you limit the tenant's ability to attach multiple vNICs to multiple external access points—this would bridge these security domains and could lead to unforeseen security compromise. It is possible mitigate this risk by utilizing the host aggregates functionality provided by OpenStack Compute or through splitting the tenant VMs into multiple tenant projects with different virtual network configurations.

Security groups

The OpenStack Networking service provides security group functionality using a mechanism that is more flexible and powerful than the security group capabilities built into OpenStack Compute. Thus, `nova.conf` should always disable built-in security groups and proxy all security group calls to the OpenStack Networking API when using OpenStack Networking. Failure to do so results in conflicting security policies being simultaneously applied by both services. To proxy security groups to OpenStack Networking, use the following configuration values:

- `firewall_driver` must be set to `nova.virt.firewall.NoopFirewallDriver` so that `nova-compute` does not perform iptables-based filtering itself.

- `security_group_api` must be set to `neutron` so that all security group requests are proxied to the OpenStack Networking service.

A security group is a container for security group rules. Security groups and their rules allow administrators and tenants the ability to specify the type of traffic and direction (ingress/egress) that is allowed to pass through a virtual interface port. When a virtual interface port is created in OpenStack Networking it is associated with a security group. If a security group is not specified, the port will be associated with a 'default' security group. By default this group will drop all ingress traffic and allow all egress. Rules can be added to this group in order to change the behaviour.

When using the security group API through OpenStack Compute, security groups are applied to all virtual interface ports on an instance. The reason

for this is that OpenStack Compute security group APIs are instance based and not virtual interface port based as OpenStack Networking.

Quotas

Quotas provide the ability to limit the number of network resources available to tenants. You can enforce default quotas for all tenants. The `/etc/neutron/neutron.conf` includes these options for quota:

```
[QUOTAS]
# resource name(s) that are supported in quota features
quota_items = network,subnet,port

# default number of resource allowed per tenant, minus for
 unlimited
#default_quota = -1

# number of networks allowed per tenant, and minus means
 unlimited
quota_network = 10

# number of subnets allowed per tenant, and minus means
 unlimited
quota_subnet = 10

# number of ports allowed per tenant, and minus means unlimited
quota_port = 50

# number of security groups allowed per tenant, and minus means
 unlimited
quota_security_group = 10

# number of security group rules allowed per tenant, and minus
 means unlimited
quota_security_group_rule = 100

# default driver to use for quota checks
quota_driver = neutron.quota.ConfDriver
```

OpenStack Networking also supports per-tenant quotas limit through a quota extension API. To enable per-tenant quotas, you must set the `quota_driver` option in `neutron.conf`.

```
quota_driver = neutron.db.quota_db.DbQuotaDriver
```

Case studies

Earlier in the section called “Introduction to case studies” [21] we introduced the Alice and Bob case studies where Alice is deploying a private

government cloud and Bob is deploying a public cloud each with different security requirements. Here we discuss how Alice and Bob would address providing networking services to the user.

Alice's private cloud

A key objective of Alice's cloud is to integrate with the existing auth services and security resources. The key design parameters for this private cloud are a limited scope of tenants, networks and workload type. This environment can be designed to limit what available network resources are available to the tenant and what are the various default quotas and security policies are available. The network policy engine can be modified to restrict creation and changes to network resources. In this environment, Alice might want to leverage nova-network in the application of security group polices on a per instance basis vs. neutron's application of security group polices on a per port basis. L2 isolation in this environment would leverage VLAN tagging. The use of VLAN tags will allow great visibility of tenant traffic by leveraging existing features and tools of the physical infrastructure.

Bob's public cloud

A major business driver for Bob is to provide an advanced networking services to his customers. Bob's customers would like to deploy multi-tiered application stacks. This multi-tiered application are either existing enterprise application or newly deployed applications. Since Bob's public cloud is a multi-tenancy enterprise service, the choice to use for L2 isolation in this environment is to use overlay networking. Another aspect of Bob's cloud is the self-service aspect where the customer can provision available networking services as needed. These networking services encompass L2 networks, L3 Routing, Network *ACL* and NAT. It is important that per-tenant quota's be implemented in this environment.

An added benefit with utilizing OpenStack Networking is when new advanced networking services become available, these new features can be easily provided to the end customers.

12. Message queuing

Message queuing services facilitate inter-process communication in OpenStack. OpenStack supports these message queuing service back ends:

- RabbitMQ
- Qpid
- ZeroMQ or 0MQ

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. ZeroMQ provides direct peer-to-peer communication through TCP sockets.

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.

With ZeroMQ messaging, IPC sockets are used on individual machines. Because these sockets are vulnerable to attack, ensure that the cloud operator has secured them.

Messaging security

This section discusses security hardening approaches for the three most common message queuing solutions used in OpenStack: RabbitMQ, Qpid, and ZeroMQ.

Messaging transport security

AMQP based solutions (Qpid and RabbitMQ) support transport-level security using TLS. ZeroMQ messaging does not natively support TLS, but transport-level security is possible using labelled IPsec or CIPSO network labels.

We highly recommend 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. Below is guidance 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.

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 an on non-SSL port. The `ssl_listeners` option should be restricted to only listen on the management network for the services.

For more information on RabbitMQ SSL configuration see:

- RabbitMQ Configuration
- RabbitMQ SSL

Qpid server SSL configuration

The Apache Foundation has a messaging security guide for Qpid. See:

- Apache Qpid SSL

Queue authentication and access control

RabbitMQ and Qpid offer authentication and access control mechanisms for controlling access to queues. ZeroMQ offers no such mechanisms.

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 user names 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 user name 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.

We recommend 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 user names 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.

Authentication configuration example: RabbitMQ

On the RabbitMQ server, delete the default `guest` user:

```
# rabbitmqctl delete_user quest
```

On the RabbitMQ server, for each OpenStack service or node that communicates with the message queue set up user accounts and privileges:

```
# rabbitmqctl add_user compute01 RABBIT_PASS
# rabbitmqctl set_permissions compute01 ".*" ".*" ".*"
```

Replace *RABBIT_PASS* with a suitable password.

For additional configuration information see:

- RabbitMQ Access Control
- RabbitMQ Authentication
- RabbitMQ Plugins
- RabbitMQ SASL External Auth

OpenStack service configuration: RabbitMQ

```
[DEFAULT]
rpc_backend=nova.openstack.common.rpc.impl_kombu
rabbit_use_ssl=True
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
```

Authentication configuration example: Qpid

For configuration information see:

- Apache Qpid Authentication
- Apache Qpid Authorization

OpenStack service configuration: Qpid

```

[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
```

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>
```

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.

Namespaces

Network namespaces are highly recommended for all services running on OpenStack Compute Hypervisors. This will help prevent against the bridging of network traffic between VM guests and the management network.

When using ZeroMQ messaging, each host must run at least one ZeroMQ message receiver to receive messages from the network and forward messages to local processes through IPC. It is possible and advisable to run an independent message receiver per project within an IPC namespace, along with other services within the same project.

Network policy

Queue servers should only accept connections from the management network. This applies to all implementations. This should be implemented through configuration of services and optionally enforced through global network policy.

When using ZeroMQ messaging, each project should run a separate ZeroMQ receiver process on a port dedicated to services belonging to that project. This is equivalent to the AMQP concept of control exchanges.

Mandatory access controls

Use both mandatory access controls (MACs) and discretionary access controls (DACs) to restrict the configuration for processes to only those processes. This restriction prevents these processes from being isolated from other processes that run on the same machine(s).

Case studies

Earlier in the section called "Introduction to case studies" [21] we introduced the Alice and Bob case studies where Alice is deploying a private government cloud and Bob is deploying a public cloud each with different security requirements. Here we discuss how Alice and Bob would address security concerns around the messaging service.

The message queue is a critical piece of infrastructure that supports a number of OpenStack services but is most strongly associated with the Compute service. Due to the nature of the message queue service, Alice and Bob have similar security concerns. One of the larger concerns that remains is that many systems have access to this queue and there is no way for a consumer of the queue messages to verify which host or service placed the messages on the queue. An attacker who is able to successfully place messages on the queue is able to create and delete VM instances, attach the block storage of any tenant and a myriad of other malicious actions. There are a number of solutions anticipated in the near future, with several proposals for message signing and encryption making their way through the OpenStack development process.

Alice's private cloud

In this case, Alice's controls are the same as Bob's controls, which are described below.

Bob's public cloud

Bob assumes the infrastructure or networks underpinning the Compute service could become compromised, therefore he recognizes the importance of hardening the system by restricting access to the message queue. In order to accomplish this task Bob deploys his RabbitMQ servers with TLS and X.509 client auth for access control. Hardening activities assists in limiting the capabilities of a malicious user that has compromised the system by disallowing queue access, provided that this user does not have valid credentials to override the controls.

Additionally, Bob adds strong network ACL rulesets to enforce which endpoints can communicate with the message servers. This second control provides some additional assurance should the other protections fail.

13. Data processing

The Data processing service for OpenStack (sahara) provides a platform for the provisioning and management of instance clusters using processing frameworks such as Hadoop and Spark. Through the OpenStack dashboard or REST API, users will be able to upload and execute framework applications which may access data in object storage or external providers. The data processing controller uses the Orchestration service to create clusters of instances which may exist as long-running groups that can grow and shrink as requested, or as transient groups created for a single workload.

Introduction to Data processing

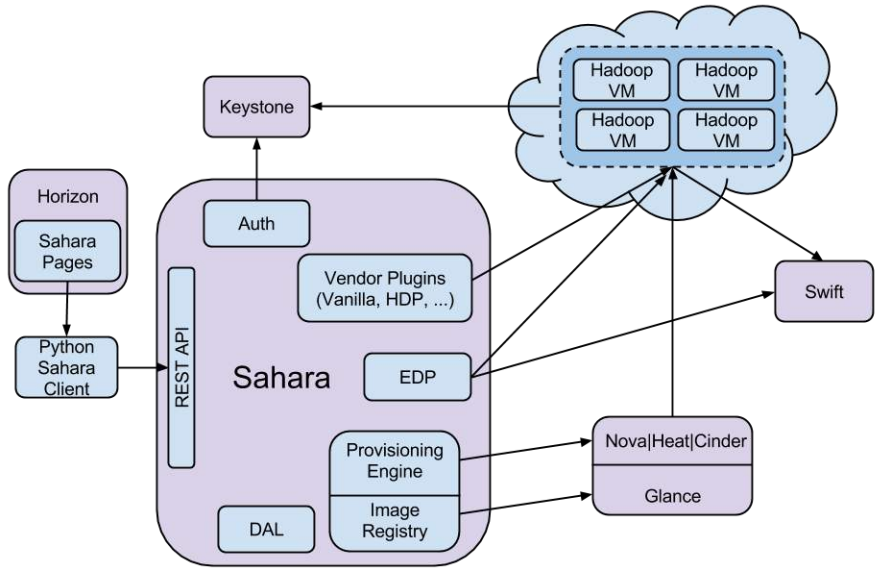
The Data processing service controller will be responsible for creating, maintaining, and destroying any instances created for its clusters. The controller will use the Networking service to establish network paths between itself and the cluster instances. It will also manage the deployment and lifecycle of user applications that are to be run on the clusters. The instances within a cluster contain the core of a framework's processing engine and the Data processing service provides several options for creating and managing the connections to these instances.

Data processing resources (clusters, jobs, and data sources) are segregated by projects defined within the Identity service. These resources are shared within a project and it is important to understand the access needs of those who are using the service. Activities within projects (for example launching clusters, uploading jobs, etc.) can be restricted further through the use of role-based access controls.

In this chapter we discuss how to assess the needs of data processing users with respect to their applications, the data that they use, and their expected capabilities within a project. We will also demonstrate a number of hardening techniques for the service controller and its clusters, and provide examples of various controller configurations and user management approaches to ensure an adequate level of security and privacy.

Architecture

The following diagram presents a conceptual view of how the Data processing service fits into the greater OpenStack ecosystem.



The Data processing service makes heavy use of the Compute, Orchestration, Image, and Block Storage services during the provisioning of clusters. It will also use one or more networks, created by the Networking service, provided during cluster creation for administrative access to the instances. While users are running framework applications the controller and the clusters will be accessing the Object Storage service. Given these service usages, we recommend following the instructions outlined in Chapter 2, "System documentation" [23] for cataloging all the components of an installation.

Technologies involved

The Data Processing service is responsible for the deployment and management of several applications. For a complete understanding of the security options provided we recommend that operators have a general familiarity with these applications. The list of highlighted technologies is broken into two sections: first, high priority applications that have a greater impact on security, and second, supporting applications with a lower impact.

Higher impact

- Hadoop
- Hadoop secure mode docs
- HDFS
- Spark
- Spark Security
- Storm
- Zookeeper

Lower impact

- Oozie
- Hive
- Pig

These technologies comprise the core of the frameworks that are deployed with the Data processing service. In addition to these technologies, the service also includes bundled frameworks provided by third party vendors. These bundled frameworks are built using the same core pieces described above plus configurations and applications that the vendors include. For more information on the third party framework bundles please see the following links:

- Cloudera CDH
- Hortonworks Data Platform
- MapR

User access to resources

The resources (clusters, jobs, and data sources) of the Data processing service are shared within the scope of a project. Although a single controller installation may manage several sets of resources, these resources will each be scoped to a single project. Given this constraint we recommend that user membership in projects is monitored closely to maintain proper segregation of resources.

As the security requirements of organizations deploying this service will vary based on their specific needs, we recommend that operators focus on data privacy, cluster management, and end-user applications as a starting point for evaluating the needs of their users. These decisions will help guide the process of configuring user access to the service. For an expanded discussion on data privacy see Chapter 15, "Tenant data privacy" [151].

The default assumption for a data processing installation is that users will have access to all functionality within their projects. In the event that more granular control is required the Data processing service provides a policy file (as described in the section called "Policies" [70]). These configurations will be highly dependent on the needs of the installing organization, and as such there is no general advice on their usage: see the section called "Role-based access control policies" [134] for details.

Deployment

The Data processing service is deployed, like many other OpenStack services, as an application running on a host connected to the stack. As of the Kilo release, it has the ability to be deployed in a distributed manner with several redundant controllers. Like other services, it also requires a database to store information about its resources. See Chapter 14, "Databases" [141]. It is important to note that the Data processing service will need to manage several Identity service trusts, communicate directly with the Orchestration and Networking services, and potentially create users in a proxy domain. For these reasons the controller will need access to the control plane and as such we recommend installing it alongside other service controllers.

Data processing interacts directly with several openstack services:

- Compute
- Identity
- Networking
- Object Storage
- Orchestration
- Block Storage (optional)

We recommend documenting all the data flows and bridging points between these services and the data processing controller. See Chapter 2, "System documentation" [23].

The Object Storage service is used by the Data processing service to store job binaries and data sources. Users wishing to have access to the full Data processing service functionality will need an object store in the projects they are using.

The Networking service plays an important role in the provisioning of clusters. Prior to provisioning, the user is expected to provide one or more networks for the cluster instances. The action of associating networks is similar to the process of assigning networks when launching instances through the dashboard. These networks are used by the controller for administrative access to the instances and frameworks of its clusters.

Also of note is the Identity service. Users of the Data processing service will need appropriate roles in their projects to allow the provisioning of instances for their clusters. Installations that use the proxy domain configuration require special consideration. See the section called "Proxy domains" [135]. Specifically, the Data processing service will need the ability to create users within the proxy domain.

Controller network access to clusters

One of the primary tasks of the data processing controller is to communicate with the instances it spawns. These instances are provisioned and then configured depending on the framework being used. The communication between the controller and the instances uses secure shell (SSH) and HTTP protocols.

When provisioning clusters each instance will be given an IP address in the networks provided by the user. The first network is often referred to as the data processing management network and instances can use the fixed IP address assigned by the Networking service for this network. The controller can also be configured to use floating IP addresses for the instances in addition to their fixed address. When communicating with the instances the controller will prefer the floating address if enabled.

For situations where the fixed and floating IP addresses do not provide the functionality required the controller can provide access through two alternate methods: custom network topologies and indirect access. The custom network topologies feature allows the controller to access the instances through a supplied shell command in the configuration file. Indirect access is used to specify instances that can be used as proxy gateways by the user

during cluster provisioning. These options are discussed with examples of usage in the section called "Configuration and hardening" [134].

Configuration and hardening

There are several configuration options and deployment strategies that can improve security in the Data processing service. The service controller is configured through a main configuration file and one or more policy files. Installations that are using the data-locality features will also have two additional files to specify the physical location of Compute and Object Storage nodes.

TLS

The Data processing service controller, like many other OpenStack controllers, can be configured to require TLS connections.

Pre-Kilo releases will require a TLS proxy as the controller does not allow direct TLS connections. Configuring TLS proxies is covered in the section called "TLS proxies and HTTP services" [48], and we recommend following the advice there to create this type of installation.

From the Kilo release onward the data processing controller allows direct TLS connections. Enabling this behavior requires some small adjustments to the controller configuration file. For any post-Juno installation we recommend enabling the direct TLS connections in the controller configuration.

Example. Configuring TLS access to the controller

```
[ssl]
ca_file = cafile.pem
cert_file = certfile.crt
key_file = keyfile.key
```

Role-based access control policies

The Data processing service uses a policy file, as described in the section called "Policies" [70], to configure role-based access control. Using the policy file an operator can restrict a group's access to specific data processing functionality.

The reasons for doing this will change depending on the organizational requirements of the installation. In general, these fine grained controls are used in situations where an operator needs to restrict the creation, dele-

tion, and retrieval of the Data processing service resources. Operators who need to restrict access within a project should be fully aware that there will need to be alternative means for users to gain access to the core functionality of the service (for example, provisioning clusters).

Example. Allow all methods to all users (default policy)

```
{
  "default": ""
}
```

Example. Disallow image registry manipulations to non-admin users

```
{
  "default": "",

  "images:register": "role:admin",
  "images:unregister": "role:admin",
  "images:add_tags": "role:admin",
  "images:remove_tags": "role:admin"
}
```

Security groups

The Data processing service allows for the association of security groups with instances provisioned for its clusters. With no additional configuration the service will use the default security group for any project that provisions clusters. A different security group may be used if requested, or an automated option exists which instructs the service to create a security group based on ports specified by the framework being accessed.

For production environments we recommend controlling the security groups manually and creating a set of group rules that are appropriate for the installation. In this manner the operator can ensure that the default security group will contain all the appropriate rules. For an expanded discussion of security groups please see the section called “Security groups” [120].

Proxy domains

When using the Object Storage service in conjunction with data processing it is necessary to add credentials for the store access. With proxy domains the Data processing service can instead use a delegated trust from the Identity service to allow store access via a temporary user created in the domain. For this delegation mechanism to work the Data processing

service must be configured to use proxy domains and the operator must configure an identity domain for the proxy users.

The data processing controller retains temporary storage of the username and password provided for object store access. When using proxy domains the controller will generate this pair for the proxy user, and the access of this user will be limited to that of the identity trust. We recommend using proxy domains in any installation where the controller or its database have routes to or from public networks.

Example. Configuring for a proxy domain named "dp_proxy"

```
[DEFAULT]
use_domain_for_proxy_users = true
proxy_user_domain_name = dp_proxy
proxy_user_role_names = Member
```

Custom network topologies

The data processing controller can be configured to use proxy commands for accessing its cluster instances. In this manner custom network topologies can be created for installations which will not use the networks provided directly by the Networking service. We recommend using this option for installations which require limiting access between the controller and the instances.

Example. Access instances through a specified relay machine

```
[DEFAULT]
proxy_command='ssh relay-machine-{tenant_id} nc {host} {port}'
```

Example. Access instances through a custom network namespace

```
[DEFAULT]
proxy_command='ip netns exec ns_for_{network_id} nc {host}
 {port}'
```

Indirect access

For installations in which the controller will have limited access to all the instances of a cluster, due to limits on floating IP addresses or security rules, indirect access may be configured. This allows some instances to be designated as proxy gateways to the other instances of the cluster.

This configuration can only be enabled while defining the node group templates that will make up the data processing clusters. It is provided as a run time option to be enabled during the cluster provisioning process.

Rootwrap

When creating custom topologies for network access it can be necessary to allow non-root users the ability to run the proxy commands. For these situations the oslo rootwrap package is used to provide a facility for non-root users to run privileged commands. This configuration requires the user associated with the data processing controller application to be in the sudoers list and for the option to be enabled in the configuration file. Optionally, an alternative rootwrap command can be provided.

Example. Enabling rootwrap usage and showing the default command

```
[DEFAULT]
use_rootwrap=True
rootwrap_command='sudo sahara-rootwrap /etc/sahara/rootwrap.
conf'
```

For more information on the rootwrap project, please see the official documentation:

https://wiki.openstack.org/wiki/Rootwrap

Logging

Monitoring the output of the service controller is a powerful forensic tool, as described more thoroughly in Chapter 18, "Monitoring and logging" [193]. The Data processing service controller offers a few options for setting the location and level of logging.

Example. Setting the log level higher than warning and specifying an output file.

```
[DEFAULT]
verbose = true
log_file = /var/log/data-processing.log
```

References

Sahara project documentation: http://docs.openstack.org/developer/sahara

Hadoop project: https://hadoop.apache.org/

Hadoop secure mode docs: https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SecureMode.html

Hadoop HDFS documentation: https://hadoop.apache.org/docs/stable/hadoop-project-dist/hadoop-hdfs/HdfsUserGuide.html

Spark project: https://spark.apache.org/

Spark security documentation: https://spark.apache.org/docs/latest/security.html

Storm project: https://storm.apache.org/

Zookeeper project: https://zookeeper.apache.org/

Oozie project: https://oozie.apache.org/

Hive https://hive.apache.org/

Pig https://pig.apache.org/

Cloudera CDH documentation: https://www.cloudera.com/content/cloudera/en/documentation.html#CDH

Hortonworks Data Platform documentation: http://docs.hortonworks.com/

MapR project: https://www.mapr.com/products/mapr-distribution-including-apache-hadoop

Case studies

Continuing with the studies described in the section called "Introduction to case studies" [21], we present Alice and Bob's approaches to deploying the Data processing service for their users.

Alice's private cloud

Alice is deploying the Data processing service for a group of users that are trusted members of a collaboration. They are all placed in a single project and share the clusters, jobs, and data within. She deploys the controller with TLS enabled, using a certificate signed by the organization's root certificate. She configures the controller to provide floating IP addresses to

the cluster instances allowing for users to gain access to the instances in the event of errors. She enables the use of proxy domains to prevent the users from needing to enter their credentials into the Data processing service.

Bob's public cloud

Bob's public cloud contains users that will not necessarily know or trust each other. He puts all users into separate projects. Each user has their own clusters, jobs, and data which cannot be accessed by other users. He deploys the controller with TLS enabled, using a certificate signed by a well known public certificate authority. He configures a custom topology to ensure that access to the provisioned cluster instances will flow through a controlled gateway. He creates a security group that opens only the ports needed for the controller to access the frameworks deployed. He enables the use of proxy domains to prevent the users from needing to enter their credentials into the Data processing service. He configures the rootwrap command to allow the data processing controller user to run the proxy commands.

14. Databases

The choice of database server is an important consideration in the security of an OpenStack deployment. Multiple factors should be considered when deciding on a database server, however for the scope of this book only security considerations will be discussed. OpenStack supports PostgreSQL and MySQL database types.

Database back end considerations

PostgreSQL has a number of desirable security features such as Kerberos authentication, object-level security, and encryption support. The PostgreSQL community has done well to provide solid guidance, documentation, and tooling to promote positive security practices.

MySQL has a large community, widespread adoption, and provides high availability options. MySQL also has the ability to provide enhanced client authentication by way of plug-in authentication mechanisms. Forked distributions in the MySQL community provide many options for consideration. It is important to choose a specific implementation of MySQL based on a thorough evaluation of the security posture and the level of support provided for the given distribution.

Security references for database back ends

Those deploying MySQL or PostgreSQL are advised to refer to existing security guidance. Some references are listed below:

MySQL:

- OWASP MySQL Hardening
- MySQL Pluggable Authentication
- Security in MySQL

PostgreSQL:

- OWASP PostgreSQL Hardening
- Total security in a PostgreSQL database

Database access control

Each of the core OpenStack services (Compute, Identity, Networking, Block Storage) store state and configuration information in databases. In this chapter, we discuss how databases are used currently in OpenStack. We also explore security concerns, and the security ramifications of database back end choices.

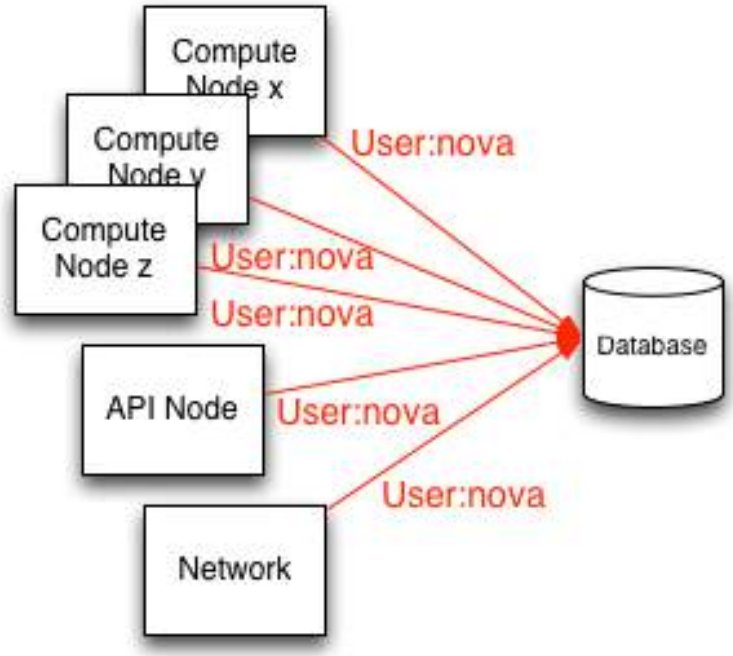
OpenStack database access model

All of the services within an OpenStack project access a single database. There are presently no reference policies for creating table or row based access restrictions to the database.

There are no general provisions for granular control of database operations in OpenStack. Access and privileges are granted simply based on whether a node has access to the database or not. In this scenario, nodes with access to the database may have full privileges to DROP, INSERT, or UPDATE functions.

Granular access control

By default, each of the OpenStack services and their processes access the database using a shared set of credentials. This makes auditing database operations and revoking access privileges from a service and its processes to the database particularly difficult.

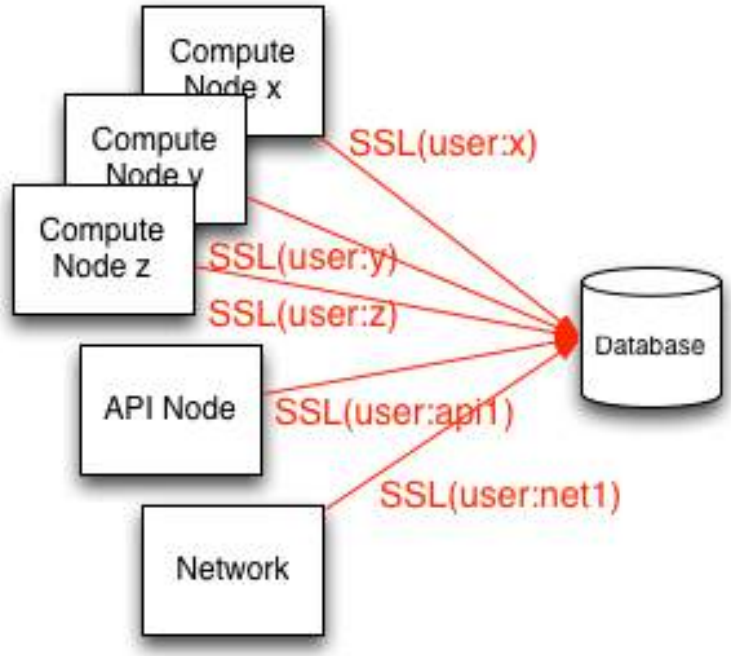


Nova-conductor

The compute nodes are the least trusted of the services in OpenStack because they host tenant instances. The `nova-conductor` service has been introduced to serve as a database proxy, acting as an intermediary between the compute nodes and the database. We discuss its ramifications later in this chapter.

We strongly recommend:

- All database communications be isolated to a management network
- Securing communications using TLS
- Creating unique database user accounts per OpenStack service endpoint (illustrated below)



Database authentication and access control

Given the risks around access to the database, we strongly recommend that unique database user accounts be created per node needing access to the database. Doing this facilitates better analysis and auditing for ensuring compliance or in the event of a compromise of a node allows you to isolate the compromised host by removing access for that node to the database upon detection. When creating these per service endpoint database user accounts, care should be taken to ensure that they are configured to require TLS. Alternatively, for increased security it is recommended that the database accounts be configured using X.509 certificate authentication in addition to user names and passwords.

Privileges

A separate database administrator (DBA) account should be created and protected that has full privileges to create/drop databases, create user accounts, and update user privileges. This simple means of separation of responsibility helps prevent accidental misconfiguration, lowers risk and lowers scope of compromise.

The database user accounts created for the OpenStack services and for each node should have privileges limited to just the database relevant to the service where the node is a member.

Require user accounts to require SSL transport

Configuration example #1: (MySQL)

```
GRANT ALL ON dbname.* to 'compute01'@'hostname' IDENTIFIED BY
 'NOVA_DBPASS' REQUIRE SSL;
```

Configuration example #2: (PostgreSQL)

In file `pg_hba.conf`:

```
hostssl dbname compute01 hostname md5
```

Note this command only adds the ability to communicate over SSL and is non-exclusive. Other access methods that may allow unencrypted transport should be disabled so that SSL is the sole access method.

The `md5` parameter defines the authentication method as a hashed password. We provide a secure authentication example in the section below.

OpenStack service database configuration

If your database server is configured for TLS transport, you will need to specify the certificate authority information for use with the initial connection string in the SQLAlchemy query.

Example of a `:sql_connection` string to MySQL:

```
sql_connection = mysql://compute01:NOVA_DBPASS@localhost/nova?
charset=utf8&ssl_ca=/etc/mysql/cacert.pem
```

Authentication with X.509 certificates

Security may be enhanced by requiring X.509 client certificates for authentication. Authenticating to the database in this manner provides greater identity assurance of the client making the connection to the database and ensures that the communications are encrypted.

Configuration example #1: (MySQL)

```
GRANT ALL on dbname.* to 'compute01'@'hostname' IDENTIFIED BY
 'NOVA_DBPASS' REQUIRE SUBJECT
'/C=XX/ST=YYY/L=ZZZZ/O=cloudycloud/CN=compute01' AND ISSUER
'/C=XX/ST=YYY/L=ZZZZ/O=cloudycloud/CN=cloud-ca';
```

Configuration example #2: (PostgreSQL)

```
hostssl dbname compute01 hostname cert
```

OpenStack service database configuration

If your database server is configured to require X.509 certificates for authentication you will need to specify the appropriate SQLAlchemy query parameters for the database back end. These parameters specify the certificate, private key, and certificate authority information for use with the initial connection string.

Example of a `:sql_connection` string for X.509 certificate authentication to MySQL:

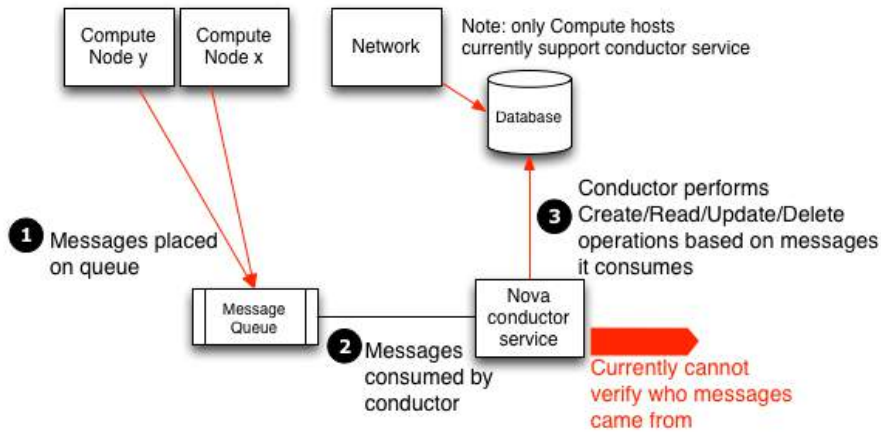
```
sql_connection = mysql://compute01:NOVA_DBPASS@localhost/nova?
charset=utf8&ssl_ca=/etc/mysql/cacert.pem&ssl_cert=/etc/mysql/
server-cert.pem&ssl_key=/etc/mysql/server-key.pem
```

Nova-conductor

OpenStack Compute offers a sub-service called `nova-conductor` which proxies database connections, with the primary purpose of having the nova compute nodes interfacing with `nova-conductor` to meet data persistence needs as opposed to directly communicating with the database.

Nova-conductor receives requests over RPC and performs actions on behalf of the calling service without granting granular access to the database, its tables, or data within. Nova-conductor essentially abstracts direct database access away from compute nodes.

This abstraction offers the advantage of restricting services to executing methods with parameters, similar to stored procedures, preventing a large number of systems from directly accessing or modifying database data. This is accomplished without having these procedures stored or executed within the context or scope of the database itself, a frequent criticism of typical stored procedures.



Unfortunately, this solution complicates the task of more fine-grained access control and the ability to audit data access. Because the `nova-conductor` service receives requests over RPC, it highlights the importance of improving the security of messaging. Any node with access to the message queue may execute these methods provided by the `nova-conductor` and effectively modifying the database.

Note, as `nova-conductor` only applies to OpenStack Compute, direct database access from compute hosts may still be necessary for the operation of other OpenStack components such as Telemetry (ceilometer), Networking, and Block Storage.

To disable the `nova-conductor`, place the following into your `nova.conf` file (on your compute hosts):

```
[conductor]
use_local = true
```

Database transport security

This chapter covers issues related to network communications to and from the database server. This includes IP address bindings and encrypting network traffic with TLS.

Database server IP address binding

To isolate sensitive database communications between the services and the database, we strongly recommend that the database server(s) be configured to only allow communications to and from the database over an

isolated management network. This is achieved by restricting the interface or IP address on which the database server binds a network socket for incoming client connections.

Restricting bind address for MySQL

In `my.cnf`:

```
[mysqld]
...
bind-address <ip address or hostname of management network
 interface>
```

Restricting listen address for PostgreSQL

In `postgresql.conf`:

```
listen_addresses = <ip address or hostname of management network
 interface>
```

Database transport

In addition to restricting database communications to the management network, we also strongly recommend that the cloud administrator configure their database back end to require TLS. Using TLS for the database client connections protects the communications from tampering and eavesdropping. As will be discussed in the next section, using TLS also provides the framework for doing database user authentication through X.509 certificates (commonly referred to as PKI). Below is guidance on how TLS is typically configured for the two popular database back ends MySQL and PostgreSQL.

Note

When installing the certificate and key files, ensure that the file permissions are restricted, for example **chmod 0600**, and the ownership is restricted to the database daemon user to prevent unauthorized access by other processes and users on the database server.

MySQL SSL configuration

The following lines should be added in the system-wide MySQL configuration file:

In `my.cnf`:

```
[[mysqld]]
...
ssl-ca=/path/to/ssl/cacert.pem
ssl-cert=/path/to/ssl/server-cert.pem
ssl-key=/path/to/ssl/server-key.pem
```

Optionally, if you wish to restrict the set of SSL ciphers used for the encrypted connection. See http://www.openssl.org/docs/apps/ciphers.html for a list of ciphers and the syntax for specifying the cipher string:

```
ssl-cipher='cipher:list'
```

PostgreSQL SSL configuration

The following lines should be added in the system-wide PostgreSQL configuration file, `postgresql.conf`.

```
ssl = true
```

Optionally, if you wish to restrict the set of SSL ciphers used for the encrypted connection. See http://www.openssl.org/docs/apps/ciphers.html for a list of ciphers and the syntax for specifying the cipher string:

```
ssl-ciphers = 'cipher:list'
```

The server certificate, key, and certificate authority (CA) files should be placed in the $PGDATA directory in the following files:

- $PGDATA/`server.crt` - Server certificate
- $PGDATA/`server.key` - Private key corresponding to `server.crt`
- $PGDATA/`root.crt` - Trusted certificate authorities
- $PGDATA/`root.crl` - Certificate revocation list

Case studies

Earlier in the section called “Introduction to case studies” [21] we introduced the Alice and Bob case studies where Alice is deploying a private government cloud and Bob is deploying a public cloud each with different security requirements. Here we discuss how Alice and Bob would address database selection and configuration for their respective private and public clouds.

Alice's private cloud

Alice's organization has high availability concerns and so she has selected MySQL as the underlying database for the cloud services. She places the database on the Management network, utilizing SSL/TLS with mutual authentication among the services to ensure secure access. Based on the assumption that external access of the database will not be facilitated, she installs a certificate signed with the organization's root certificate on the database and its access endpoints. Alice creates separate user accounts for each database user then configures the database to use both passwords and X.509 certificates for authentication. She elects not to use the `nova-conductor` sub-service due to the desire for fine-grained access control policies and audit support.

Bob's public cloud

Bob is concerned about strong separation of his tenants' data, so he has elected to use the PostgreSQL database, known for its stronger security features. The database resides on the Management network and uses SSL/TLS with mutual authentication with the services. Since the database is on the Management network, the database uses certificates signed with the company's self-signed root certificate. Bob creates separate user accounts for each database user, and configures the database to use both passwords and X.509 certificates for authentication. He elects not to use the `nova-conductor` sub-service due to a desire for fine-grained access control.

15. Tenant data privacy

OpenStack is designed to support multitenancy and those tenants will most probably have different data requirements. As a cloud builder and operator you need to ensure your OpenStack environment can address various data privacy concerns and regulations. In this chapter we will address data residency and disposal as it pertains to OpenStack implementations.

Data privacy concerns

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.

Numerous OpenStack services maintain data and metadata belonging to tenants or reference tenant information.

Tenant data stored in an OpenStack cloud may 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
- Machine snapshots
- Data passed to OpenStack Compute's configuration-drive extension

Metadata stored by an OpenStack cloud includes the following non-exhaustive items:

- 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

Data disposal

OpenStack operators should strive to provide a certain level of tenant data disposal assurance. Best practices suggest that the operator sanitize cloud system media (digital and non-digital) prior to disposal, release out of organization control or release for reuse. Sanitization methods should implement an appropriate level of strength and integrity given the specific security domain and sensitivity of the information.

> "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." NIST Special Publication 800-53 Revision 4

General data disposal and sanitization guidelines as adopted from NIST recommended security controls. Cloud operators should:

1. Track, document and verify media sanitization and disposal actions.
2. Test sanitation equipment and procedures to verify proper performance.
3. Sanitize portable, removable storage devices prior to connecting such devices to the cloud infrastructure.
4. Destroy cloud system media that cannot be sanitized.

In an OpenStack deployment you will need to address the following:

- Secure data erasure
- Instance memory scrubbing
- Block Storage volume data
- Compute instance ephemeral storage
- Bare metal server sanitization

Data not securely erased

Within OpenStack some data may 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 may be remediated with database and/or system configuration for auto vacuuming and periodic free-space wiping.

Instance memory scrubbing

Specific to various hypervisors is the treatment of instance memory. This behavior is not defined in OpenStack 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.

Xen explicitly assigns dedicated memory regions to instances and scrubs data upon the destruction of instances (or domains in Xen parlance). KVM depends more greatly on Linux page management; A complex set of rules related to KVM paging is defined in the KVM documentation.

It is important to note that use of the Xen memory balloon feature is likely to result in information disclosure. We strongly recommended to avoid use of this feature.

For these and other hypervisors, we recommend referring to hypervisor-specific documentation.

Cinder volume data

Use of the OpenStack volume encryption feature is highly encouraged. This is discussed in the Data Encryption section below. When this feature is used, destruction of data is accomplished by securely deleting the encryption key.

If a backend plugin is being used, there may be independent ways of doing encryption or non-standard overwrite solutions. Plugins to OpenStack Block Storage will store data in a variety of ways. Many plug-ins are specific to a vendor or technology, whereas others are more DIY solutions around filesystems such as LVM or ZFS. Methods to securely destroy data will vary from one plugin to another, from one vendor's solution to another, and from one filesystem to another.

Some back ends such as ZFS will support copy-on-write to prevent data exposure. In these cases, reads from unwritten blocks will always return zero. Other back ends such as LVM may not natively support this, thus the Block Storage 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 mediations may be available for those assurances not provided.

Finally, while not a feature of OpenStack, vendors and implementors may choose to add or support encryption of volumes. In this case, destruction of data is as simple as throwing away the key.

Image service delay delete feature

OpenStack Image service has a delayed delete feature, which will pend the deletion of an image for a defined time period. It is recommended to disable this feature if it is a security concern, by editing the `etc/glance/glance-api.conf` file and setting the `delayed_delete` option as *`False`*.

Compute soft delete feature

OpenStack 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 `etc/nova/nova.conf` file and leave the `reclaim_instance_interval` option empty.

Compute instance ephemeral storage

The creation and destruction of ephemeral storage will be somewhat dependent on the chosen hypervisor and the OpenStack Compute plug-in.

The libvirt plug-in for compute may maintain ephemeral storage directly on a filesystem, or in LVM. Filesystem storage generally will not overwrite data when it is removed, although there is a guarantee that dirty extents are not provisioned to users.

When using LVM backed ephemeral storage, which is block-based, it is necessary that the OpenStack Compute software securely erases blocks to prevent information disclosure. There have in the past been information disclosure vulnerabilities related to improperly erased ephemeral block storage devices.

Filesystem storage is a more secure solution for ephemeral block storage devices than LVM as dirty extents cannot be provisioned to users. However, it is important to be mindful that user data is not destroyed, so it is suggested to encrypt the backing filesystem.

Bare metal server sanitization

A bare metal server driver for Compute was under development and has since moved into a separate project called ironic. At the time of this writing, ironic does not appear to address sanitization of tenant data resident the physical hardware.

Additionally, it is possible for tenants of a bare metal system to modify system firmware. TPM technology, described in the section called "Secure bootstrapping" [32], provides a solution for detecting unauthorized firmware changes.

Data encryption

The option exists for implementers to encrypt tenant 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 tenants is largely related to the risk assumed by a provider that an attacker could access tenant data. There may 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. It is recommended that a risk assessment and legal consul advised before choosing tenant encryption policies.

Per-instance or per-object encryption is preferable over, in descending order, per-project, per-tenant, 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-tenant. We recommend implementors make a best-effort in encrypting tenant data.

Often, data encryption relates positively to the ability to reliably destroy tenant 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

Volume encryption

A volume encryption feature in OpenStack supports privacy on a per-tenant basis. As of the Kilo release, 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 through a secure wrapper
  - Volume encryption is supported by back-end key storage for enhanced security (for example, a Hardware Security Module (HSM) or a KMIP server can be used as a Barbican back-end secret store)

Ephemeral disk encryption

An ephemeral disk encryption feature addresses data privacy. The ephemeral disk is a temporary work space used by the virtual host operating system. Without encryption, sensitive user information could be ac-

cessed on this disk, and vestigial information could remain after the disk is unmounted. As of the Kilo release, the following ephemeral disk encryption features are supported:

- Creation and usage of encrypted LVM ephemeral disks
  - Compute configuration enables encryption and specifies encryption parameters such as algorithm and key size
- Interface with the Key management service through a secure wrapper
  - Key management service will support data isolation by providing ephemeral disk encryption keys on a per-tenant basis
  - Ephemeral disk encryption is supported by back-end key storage for enhanced security (for example, an HSM or a KMIP server can be used as a Barbican back-end secret store)
  - With the Key management service, when an ephemeral disk is no longer needed, simply deleting the key may take the place of overwriting the ephemeral disk storage area

Object Storage objects

The ability to encrypt objects in Object Storage is presently limited to disk-level encryption per node. However, there does exist third-party extensions and modules for per-object encryption. These modules have been proposed upstream, but have not per this writing been formally accepted. Below are some pointers:

https://github.com/Mirantis/swift-encrypt

http://www.mirantis.com/blog/on-disk-encryption-prototype-for-openstack-swift/

Block Storage volumes and instance ephemeral filesystems

Block Storage supports a variety of mechanisms for supplying mountable volumes. The ability to encrypt volumes on the storage host depends on the service back ends chosen. Some back ends may not support this at all. It is outside the scope of this guide to specify recommendations for each Block Storage back-end driver.

For the purpose of performance, many storage protocols are unencrypted. Some protocols such as iSCSI can provide authentication and encrypted sessions, it is our recommendation to enable these features.

As both block storage and compute support LVM backed storage, we can easily provide an example applicable to both systems. In deployments using LVM, encryption may be performed against the backing physical volumes. An encrypted block device would be created using the standard Linux tools, with the LVM physical volume (PV) created on top of the decrypted block device using pvcreate. Then, the vgcreate or vgmodify tool may be used to add the encrypted physical volume to an LVM volume group (VG).

Network data

Tenant data for compute could be encrypted over IPsec or other tunnels. This is not functionality common or standard in OpenStack, but is an option available to motivated and interested implementors.

Likewise, encrypted data will remain encrypted as it is transferred over the network.

Key management

To address the often mentioned concern of tenant data privacy and limiting cloud provider liability, there is greater 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 tenant objects such as media files, database archives among others. In some instances, client-side encryption is utilized to encrypt data held by the virtualization technologies which requires client interaction, such as presenting keys, to decrypt data for future use. To seamlessly secure the data and have it accessible without burdening the client with having to manage their keys and interactively provide them calls for a key management service within OpenStack. Providing encryption and key management services as part of OpenStack eases data-at-rest security adoption and addresses customer concerns about privacy or misuse of data, while also limiting cloud provider liability. This can help reduce a provider's liability when handling tenant data during an incident investigation in multi-tenant public clouds.

The volume encryption and ephemeral disk encryption features rely on a key management service (for example, Barbican) for the creation and se-

cure storage of keys. The key manager is pluggable to facilitate deployments that need a third-party Hardware Security Module (HSM) or the use of the Key Management Interchange Protocol (KMIP), which is supported by an open-source project called PyKMIP.

Bibliography:

- OpenStack.org, Welcome to Barbican's Developer Documentation!. 2014. Barbican developer documentation
- oasis-open.org, OASIS Key Management Interoperability Protocol (KMIP). 2014. KMIP
- PyKMIP library https://github.com/OpenKMIP/PyKMIP

Case studies

Earlier in the section called "Introduction to case studies" [21] we introduced the Alice and Bob case studies where Alice is deploying a private government cloud and Bob is deploying a public cloud each with different security requirements. Here we dive into their particular tenant data privacy requirements. Specifically, we will look into how Alice and Bob both handle tenant data, data destruction, and data encryption.

Alice's private cloud

As stated during the introduction to Alice's case study, data protection is of an extremely high priority. She needs to ensure that a compromise of one tenant's data does not cause loss of other tenant data. She also has strong regulator requirements that require documentation of data destruction activities. Alice does this using the following:

- Establishing procedures to sanitize tenant data when a program or project ends.
- Track the destruction of both the tenant data and metadata through ticketing in a CMDB.
- For Volume storage:
  - Physical server issues
  - To provide secure ephemeral instance storage, Alice implements qcow2 files on an encrypted filesystem.

Bob's public cloud

As stated during the introduction to Bob's case study, tenant privacy is of an extremely high priority. In addition to the requirements and actions Bob will take to isolate tenants from one another at the infrastructure layer, Bob also needs to provide assurances for tenant data privacy. Bob does this using the following:

- Establishing procedures to sanitize customer data when a customer churns.
- Track the destruction of both the customer data and metadata through ticketing in a CMDB.
- For Volume storage:
  - Physical server issues
  - To provide secure ephemeral instance storage, Bob implements qcow2 files on an encrypted filesystems.

16. Hypervisor and virtualization layer

Virtualization can provide flexibility, improved resource utilization, faster provisioning, and other benefits that enable cloud computing. The virtualization stack can also provide isolation between guest virtual machines, however, appropriate security measures must be considered to reduce the risks associated with hypervisor breakout attacks.

This chapter discusses the hypervisor selection process, hardening the virtualization layer, and how to improve instance isolation.

Hypervisor selection

Hypervisors in OpenStack

Whether OpenStack is deployed within private data centers or as a public cloud service, the underlying virtualization technology provides enterprise-level capabilities in the realms of scalability, resource efficiency, and uptime. While such high-level benefits are generally available across many OpenStack-supported hypervisor technologies, there are significant differences in the security architecture and features for each hypervisor, particularly when considering the security threat vectors which are unique to elastic OpenStack environments. As applications consolidate into single Infrastructure-as-a-Service (IaaS) platforms, instance isolation at the hypervisor level becomes paramount. The requirement for secure isolation holds true across commercial, government, and military communities.

Within the OpenStack framework, you can choose among many hypervisor platforms and corresponding OpenStack plug-ins to optimize your cloud environment. In the context of this guide, hypervisor selection considerations are highlighted as they pertain to feature sets that are critical to security. However, these considerations are not meant to be an exhaustive investigation into the pros and cons of particular hypervisors. NIST provides additional guidance in Special Publication 800-125, "*Guide to Security for Full Virtualization Technologies*".

Selection criteria

As part of your hypervisor selection process, you must consider a number of important factors to help increase your security posture. Specifically, you must become familiar with these areas:

- Team expertise
- Product or project maturity
- Common criteria
- Certifications and attestations
- Hardware concerns
- Hypervisor vs. baremetal
- Additional security features

Additionally, the following security-related criteria are highly encouraged to be evaluated when selecting a hypervisor for OpenStack deployments:

- Has the hypervisor undergone Common Criteria certification? If so, to what levels?
- Is the underlying cryptography certified by a third-party?

Team expertise

Most likely, the most important aspect in hypervisor selection is the expertise of your staff in managing and maintaining a particular hypervisor platform. The more familiar your team is with a given product, its configuration, and its eccentricities, the fewer the configuration mistakes. Additionally, having staff expertise spread across an organization on a given hypervisor increases availability of your systems, allows segregation of duties, and mitigates problems in the event that a team member is unavailable.

Product or project maturity

The maturity of a given hypervisor product or project is critical to your security posture as well. Product maturity has a number of effects once you have deployed your cloud:

- Availability of expertise
- Active developer and user communities

- Timeliness and availability of updates
- Incidence response

One of the biggest indicators of a hypervisor's maturity is the size and vibrancy of the community that surrounds it. As this concerns security, the quality of the community affects the availability of expertise if you need additional cloud operators. It is also a sign of how widely deployed the hypervisor is, in turn leading to the battle readiness of any reference architectures and best practices.

Further, the quality of community, as it surrounds an open source hypervisor like KVM or Xen, has a direct impact on the timeliness of bug fixes and security updates. When investigating both commercial and open source hypervisors, you must look into their release and support cycles as well as the time delta between the announcement of a bug or security issue and a patch or response. Lastly, the supported capabilities of OpenStack compute vary depending on the hypervisor chosen. See the OpenStack Hypervisor Support Matrix for OpenStack compute feature support by hypervisor.

Certifications and attestations

One additional consideration when selecting a hypervisor is the availability of various formal certifications and attestations. While they may not be requirements for your specific organization, these certifications and attestations speak to the maturity, production readiness, and thoroughness of the testing a particular hypervisor platform has been subjected to.

Common criteria

Common Criteria is an internationally standardized software evaluation process, used by governments and commercial companies to validate software technologies perform as advertised. In the government sector, NSTISSP No. 11 mandates that U.S. Government agencies only procure software which has been Common Criteria certified, a policy which has been in place since July 2002. It should be specifically noted that OpenStack has not undergone Common Criteria certification, however many of the available hypervisors have.

In addition to validating a technologies capabilities, the Common Criteria process evaluates *how* technologies are developed.

- How is source code management performed?
- How are users granted access to build systems?

- Is the technology cryptographically signed before distribution?

The KVM hypervisor has been Common Criteria certified through the U.S. Government and commercial distributions, which have been validated to separate the runtime environment of virtual machines from each other, providing foundational technology to enforce instance isolation. In addition to virtual machine isolation, KVM has been Common Criteria certified to

> "*provide system-inherent separation mechanisms to the resources of virtual machines. This separation ensures that large software component used for virtualizing and simulating devices executing for each virtual machine cannot interfere with each other. Using the SELinux multi-category mechanism, the virtualization and simulation software instances are isolated. The virtual machine management framework configures SELinux multi-category settings transparently to the administrator*"

While many hypervisor vendors, such as Red Hat, Microsoft, and VMware have achieved Common Criteria Certification their underlying certified feature set differs. It is recommended to evaluate vendor claims to ensure they minimally satisfy the following requirements:

Identification and Authentication	Identification and authentication using pluggable authentication modules (PAM) based upon user passwords. The quality of the passwords used can be enforced through configuration options.
Audit	The system provides the capability to audit a large number of events including individual system calls as well as events generated by trusted processes. Audit data is collected in regular files in ASCII format. The system provides a program for the purpose of searching the audit records. The system administrator can define a rule base to restrict auditing to the events they are interested in. This includes the ability to restrict auditing to specific events, specific users, specific objects or a combination of all of this. Audit records can be transferred to a remote audit daemon.
Discretionary Access Control	Discretionary Access Control (*DAC*) restricts access to file system objects

	based on *Access Control Lists* (ACLs) that include the standard UNIX permissions for user, group and others. Access control mechanisms also protect IPC objects from unauthorized access. The system includes the ext4 file system, which supports POSIX ACLs. This allows defining access rights to files within this type of file system down to the granularity of a single user.
Mandatory Access Control	Mandatory Access Control (MAC) restricts access to objects based on labels assigned to subjects and objects. Sensitivity labels are automatically attached to processes and objects. The access control policy enforced using these labels is derived from the *Bell-LaPadula access control model*. SELinux categories are attached to virtual machines and its resources. The access control policy enforced using these categories grant virtual machines access to resources if the category of the virtual machine is identical to the category of the accessed resource. The TOE implements non-hierarchical categories to control access to virtual machines.
Role-Based Access Control	Role-based access control (RBAC) allows separation of roles to eliminate the need for an all-powerful system administrator.
Object Reuse	File system objects and memory and IPC objects are cleared before they can be reused by a process belonging to a different user.
Security Management	The management of the security critical parameters of the system is performed by administrative users. A set of commands that require root privileges (or specific roles when RBAC is used) are used for system management. Security parameters are stored in specific files that are protected by the access control mechanisms of the system against unauthorized access by users that are not administrative users.
Secure Communication	The system supports the definition of trusted channels using SSH. Pass-

	word based authentication is supported. Only a restricted number of cipher suites are supported for those protocols in the evaluated configuration.
Storage Encryption	The system supports encrypted block devices to provide storage confidentiality via dm_crypt.
TSF Protection	While in operation, the kernel software and data are protected by the hardware memory protection mechanisms. The memory and process management components of the kernel ensure a user process cannot access kernel storage or storage belonging to other processes. Non-kernel TSF software and data are protected by DAC and process isolation mechanisms. In the evaluated configuration, the reserved user ID root owns the directories and files that define the TSF configuration. In general, files and directories containing internal TSF data, such as configuration files and batch job queues, are also protected from reading by DAC permissions. The system and the hardware and firmware components are required to be physically protected from unauthorized access. The system kernel mediates all access to the hardware mechanisms themselves, other than program visible CPU instruction functions. In addition, mechanisms for protection against stack overflow attacks are provided.

Cryptography standards

Several cryptography algorithms are available within OpenStack for identification and authorization, data transfer and protection of data at rest. When selecting a hypervisor, the following are recommended algorithms and implementation standards to ensure the virtualization layer supports:

Algorithm	Key length	Intended purpose	Security function	Implementation standard
AES	128, 192, or 256 bits	Encryption / decryption	Protected data transfer, protection for data at rest	RFC 4253

Algorithm	Key length	Intended purpose	Security function	Implementation standard
TDES	168 bits	Encryption / decryption	Protected data transfer	RFC 4253
RSA	1024, 2048, or 3072 bits	Authentication, key exchange	Identification and authentication, protected data transfer	U.S. NIST FIPS PUB 186-3
DSA	L=1024, N=160 bits	Authentication, key exchange	Identification and authentication, protected data transfer	U.S. NIST FIPS PUB 186-3
Serpent	128, 192, or 256 bits	Encryption / decryption	Protection of data at rest	http://www.cl.cam.ac.uk/~rja14/Papers/serpent.pdf
Twofish	128, 192, or 256 bit	Encryption / decryption	Protection of data at rest	http://www.schneier.com/paper-twofish-paper.html
SHA-1	-	Message Digest	Protection of data at rest, protected data transfer	U.S. NIST FIPS PUB 180-3
SHA-2 (224, 256, 384, or 512 bits)	-	Message Digest	Protection for data at rest, identification and authentication	U.S. NIST FIPS PUB 180-3

FIPS 140-2

In the United States the National Institute of Science and Technology (NIST) certifies cryptographic algorithms through a process known the Cryptographic Module Validation Program. NIST certifies algorithms for conformance against Federal Information Processing Standard 140-2 (FIPS 140-2), which ensures:

> *Products validated as conforming to FIPS 140-2 are accepted by the Federal agencies of both countries [United States and Canada] for the protection of sensitive information (United States) or Designated Information (Canada). The goal of the CMVP is to promote the use of validated cryptographic modules and provide Federal agencies with a security metric to use in procuring equipment containing validated cryptographic modules.*

When evaluating base hypervisor technologies, consider if the hypervisor has been certified against FIPS 140-2. Not only is conformance against FIPS 140-2 mandated per U.S. Government policy, formal certification indicates that a given implementation of a cryptographic algorithm has been reviewed for conformance against module specification, cryptographic module ports and interfaces; roles, services, and authentication; finite state model; physical security; operational environment; cryptographic key management; electromagnetic interference/electromagnetic compatibility (EMI/EMC); self-tests; design assurance; and mitigation of other attacks.

Hardware concerns

Further, when you evaluate a hypervisor platform, consider the supportability of the hardware on which the hypervisor will run. Additionally, consider the additional features available in the hardware and how those features are supported by the hypervisor you chose as part of the OpenStack deployment. To that end, hypervisors each have their own hardware compatibility lists (HCLs). When selecting compatible hardware it is important to know in advance which hardware-based virtualization technologies are important from a security perspective.

Description	Technology	Explanation
I/O MMU	VT-d / AMD-Vi	Required for protecting PCI-passthrough
Intel Trusted Execution Technology	Intel TXT / SEM	Required for dynamic attestation services
PCI-SIG I/O virtualization	SR-IOV, MR-IOV, ATS	Required to allow secure sharing of PCI Express devices
Network virtualization	VT-c	Improves performance of network I/O on hypervisors

Hypervisor vs. baremetal

It is important to recognize the difference between using LXC (Linux Containers) or baremetal systems vs using a hypervisor like KVM. Specifically, the focus of this security guide is largely based on having a hypervisor and virtualization platform. However, should your implementation require the use of a baremetal or LXC environment, you must pay attention to the particular differences in regard to deployment of that environment.

In particular, you must assure your end users that the node has been properly sanitized of their data prior to re-provisioning. Additionally, prior to

reusing a node, you must provide assurances that the hardware has not been tampered or otherwise compromised.

Note

While OpenStack has a baremetal project, a discussion of the particular security implications of running baremetal is beyond the scope of this book.

Finally, due to the time constraints around a book sprint, the team chose to use KVM as the hypervisor in our example implementations and architectures.

Note

There is an OpenStack Security Note pertaining to the use of LXC in Compute.

Hypervisor memory optimization

Many hypervisors use memory optimization techniques to overcommit memory to guest virtual machines. This is a useful feature that allows you to deploy very dense compute clusters. One way to achieve this is through de-duplication or "sharing" of memory pages. When two virtual machines have identical data in memory, there are advantages to having them reference the same memory.

Typically this is achieved through Copy-On-Write (COW) mechanisms. These mechanisms have been shown to be vulnerable to side-channel attacks where one VM can infer something about the state of another and might not be appropriate for multi-tenant environments where not all tenants are trusted or share the same levels of trust.

KVM Kernel Samepage Merging

Introduced into the Linux kernel in version 2.6.32, Kernel Samepage Merging (KSM) consolidates identical memory pages between Linux processes. As each guest VM under the KVM hypervisor runs in its own process, KSM can be used to optimize memory use between VMs.

XEN transparent page sharing

XenServer 5.6 includes a memory overcommitment feature named Transparent Page Sharing (TPS). TPS scans memory in 4 KB chunks for any dupli-

cates. When found, the Xen Virtual Machine Monitor (VMM) discards one of the duplicates and records the reference of the second one.

Security considerations for memory optimization

Traditionally, memory de-duplication systems are vulnerable to side channel attacks. Both KSM and TPS have demonstrated to be vulnerable to some form of attack. In academic studies attackers were able to identify software packages and versions running on neighboring virtual machines as well as software downloads and other sensitive information through analyzing memory access times on the attacker VM.

If a cloud deployment requires strong separation of tenants, as is the situation with public clouds and some private clouds, deployers should consider disabling TPS and KSM memory optimizations.

Additional security features

Another thing to look into when selecting a hypervisor platform is the availability of specific security features. In particular, we are referring to features like Xen Server's XSM or Xen Security Modules, sVirt, Intel TXT, and AppArmor. The presence of these features increase your security profile as well as provide a good foundation.

The following table calls out these features by common hypervisor platforms.

	XSM	sVirt	TXT	AppArmor	cgroups	MAC Policy
KVM		✓	✓	✓	✓	✓
Xen	✓		✓			✓
ESXi			✓			
Hyper-V						

MAC Policy: Mandatory Access Control; may be implemented with SELinux or other operating systems

* Features in this table might not be applicable to all hypervisors or directly mappable between hypervisors.

Bibliography

- Sunar, Eisenbarth, Inci, Gorka Irazoqui Apecechea. Fine Grain Cross-VM Attacks on Xen and VMware are possible!. 2014. https://eprint.iacr.org/2014/248.pdf

- Artho, Yagi, Iijima, Kuniyasu Suzaki. Memory Deduplication as a Threat to the Guest OS. 2011. https://staff.aist.go.jp/c.artho/papers/EuroSec2011-suzaki.pdf
- KVM: Kernal-based Virtual Machine. Kernal Samepage Merging. 2010. KVM: Kernel Samepage Merging
- Xen Project, Xen Security Modules: XSM-FLASK. 2014. XSM: Xen Security Modules
- SELinux Project, SVirt. 2011. xVirt: Mandatory Access Control for Linux-based virtualization
- Intel.com, Trusted Compute Pools with Intel Trusted Execution Technology (Intel TXT). TXT: Intel Trusted Execution Technology
- AppArmor.net, AppArmor Main Page. 2011. AppArmor: Linux security module implementing MAC
- Kernal.org, CGroups. 2004. cgroups: Linux kernel feature to control resource usage
- Computer Security Resource Centre. Guide to Security for Full Virtualization Technologies. 2011. Guide to Security for Full Virtualization Technologies
- National Information Assurance Partnership, National Security Telecommunications and Information Systems Security Policy. 2003. National Security Telecommunications and Information Systems Security Policy No. 11

Hardening the virtualization layers

In the beginning of this chapter we discuss the use of both physical and virtual hardware by instances, the associated security risks, and some recommendations for mitigating those risks. We conclude the chapter with a discussion of sVirt, an open source project for integrating SELinux mandatory access controls with the virtualization components.

Physical hardware (PCI passthrough)

Many hypervisors offer a functionality known as PCI passthrough. This allows an instance to have direct access to a piece of hardware on the node. For example, this could be used to allow instances to access video cards or

GPUs offering the compute unified device architecture (CUDA) for high performance computation. This feature carries two types of security risks: direct memory access and hardware infection.

Direct memory access (DMA) is a feature that permits certain hardware devices to access arbitrary physical memory addresses in the host computer. Often video cards have this capability. However, an instance should not be given arbitrary physical memory access because this would give it full view of both the host system and other instances running on the same node. Hardware vendors use an input/output memory management unit (IOMMU) to manage DMA access in these situations. Therefore, cloud architects should ensure that the hypervisor is configured to utilize this hardware feature.

- KVM: How to assign devices with VT-d in KVM
- Xen: VTd Howto

Note

The IOMMU feature is marketed as VT-d by Intel and AMD-Vi by AMD.

A hardware infection occurs when an instance makes a malicious modification to the firmware or some other part of a device. As this device is used by other instances or the host OS, the malicious code can spread into those systems. The end result is that one instance can run code outside of its security domain. This is a significant breach as it is harder to reset the state of physical hardware than virtual hardware, and can lead to additional exposure such as access to the management network.

Solutions to the hardware infection problem are domain specific. The strategy is to identify how an instance can modify hardware state then determine how to reset any modifications when the instance is done using the hardware. For example, one option could be to re-flash the firmware after use. Clearly there is a need to balance hardware longevity with security as some firmwares will fail after a large number of writes. TPM technology, described in the section called “Secure bootstrapping” [32], provides a solution for detecting unauthorized firmware changes. Regardless of the strategy selected, it is important to understand the risks associated with this kind of hardware sharing so that they can be properly mitigated for a given deployment scenario.

Additionally, due to the risk and complexities associated with PCI passthrough, it should be disabled by default. If enabled for a specific

need, you will need to have appropriate processes in place to ensure the hardware is clean before re-issue.

Virtual hardware (QEMU)

When running a virtual machine, virtual hardware is a software layer that provides the hardware interface for the virtual machine. Instances use this functionality to provide network, storage, video, and other devices that may be needed. With this in mind, most instances in your environment will exclusively use virtual hardware, with a minority that will require direct hardware access. The major open source hypervisors use QEMU for this functionality. While QEMU fills an important need for virtualization platforms, it has proven to be a very challenging software project to write and maintain. Much of the functionality in QEMU is implemented with low-level code that is difficult for most developers to comprehend. Furthermore, the hardware virtualized by QEMU includes many legacy devices that have their own set of quirks. Putting all of this together, QEMU has been the source of many security problems, including hypervisor breakout attacks.

Therefore, it is important to take proactive steps to harden QEMU. Three specific steps are recommended: minimizing the code base, using compiler hardening, and using mandatory access controls such as sVirt, SELinux, or AppArmor.

Additionally, ensure iptables has the default policy filtering network traffic, and consider examining the existing rule set to understand each rule and determine if the policy needs to be expanded upon.

Minimizing the QEMU code base

The first recommendation is to minimize the QEMU code base by removing unused components from the system. QEMU provides support for many different virtual hardware devices, however only a small number of devices are needed for a given instance. The most common hardware devices are the virtio devices. Some legacy instances will need access to specific hardware, which can be specified using glance metadata:

```
$ glance image-update \
    --property hw_disk_bus=ide \
    --property hw_cdrom_bus=ide \
    --property hw_vif_model=e1000 \
    f16-x86_64-openstack-sda
```

A cloud architect should decide what devices to make available to cloud users. Anything that is not needed should be removed from QEMU. This step requires recompiling QEMU after modifying the options passed to the

QEMU configure script. For a complete list of up-to-date options simply run **./configure –help** from within the QEMU source directory. Decide what is needed for your deployment, and disable the remaining options.

Compiler hardening

The next step is to harden QEMU using compiler hardening options. Modern compilers provide a variety of compile time options to improve the security of the resulting binaries. These features, which we will describe in more detail below, include relocation read-only (RELRO), stack canaries, never execute (NX), position independent executable (PIE), and address space layout randomization (ASLR).

Many modern Linux distributions already build QEMU with compiler hardening enabled, so you may want to verify your existing executable before proceeding with the information below. One tool that can assist you with this verification is called `checksec.sh`.

RELocation Read-Only (RELRO)	Hardens the data sections of an executable. Both full and partial RELRO modes are supported by gcc. For QEMU full RELRO is your best choice. This will make the global offset table read-only and place various internal data sections before the program data section in the resulting executable.
Stack canaries	Places values on the stack and verifies their presence to help prevent buffer overflow attacks.
Never eXecute (NX)	Also known as Data Execution Prevention (DEP), ensures that data sections of the executable can not be executed.
Position Independent Executable (PIE)	Produces a position independent executable, which is necessary for ASLR.
Address Space Layout Randomization (ASLR)	This ensures that placement of both code and data regions will be randomized. Enabled by the kernel (all modern Linux kernels support ASLR), when the executable is built with PIE.

The following compiler options are recommend for GCC when compiling QEMU:

```
CFLAGS="-arch x86_64 -fstack-protector-all -Wstack-protector \
 --param ssp-buffer-size=4 -pie -fPIE -ftrapv -D_FORTIFY_SOURCE=
2 -O2 \
 -Wl,-z,relro,-z,now"
```

We recommend testing your QEMU executable file after it is compiled to ensure that the compiler hardening worked properly.

Most cloud deployments will not want to build software such as QEMU by hand. It is better to use packaging to ensure that the process is repeatable and to ensure that the end result can be easily deployed throughout the cloud. The references below provide some additional details on applying compiler hardening options to existing packages.

- DEB packages: Hardening Walkthrough
- RPM packages: How to create an RPM package

Mandatory access controls

Compiler hardening makes it more difficult to attack the QEMU process. However, if an attacker does succeed, we would like to limit the impact of the attack. Mandatory access controls accomplish this by restricting the privileges on QEMU process to only what is needed. This can be accomplished using sVirt / SELinux or AppArmor. When using sVirt, SELinux is configured to run each QEMU process under a separate security context. AppArmor can be configured to provide similar functionality. We provide more details on sVirt and instance isolation in the section below.

sVirt: SELinux and virtualization

With unique kernel-level architecture and National Security Agency (NSA) developed security mechanisms, KVM provides foundational isolation technologies for multi-tenancy. With developmental origins dating back to 2002, the Secure Virtualization (sVirt) technology is the application of SELinux against modern day virtualization. SELinux, which was designed to apply separation control based upon labels, has been extended to provide isolation between virtual machine processes, devices, data files and system processes acting upon their behalf.

OpenStack's sVirt implementation aspires to protect hypervisor hosts and virtual machines against two primary threat vectors:

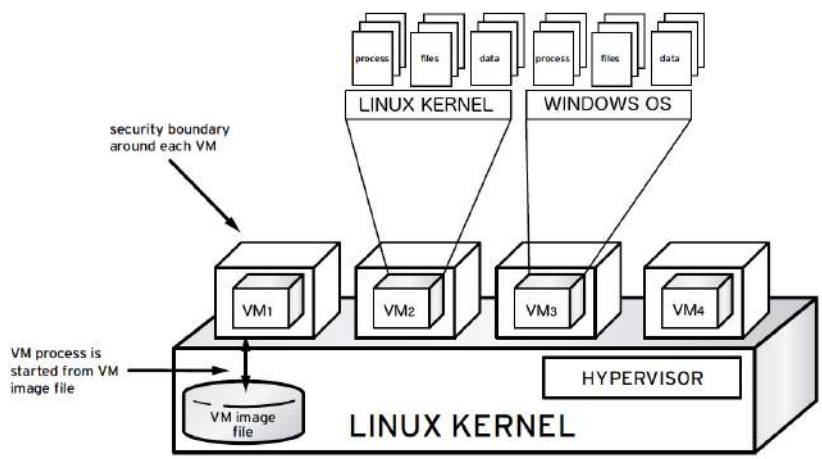
Hypervisor threats A compromised application running within a virtual machine attacks the hy-

pervisor to access underlying resources. For example, when a virtual machine is able to access the hypervisor OS, physical devices, or other applications. This threat vector represents considerable risk as a compromise on a hypervisor can infect the physical hardware as well as exposing other virtual machines and network segments.

Virtual Machine (multi-tenant) threats

A compromised application running within a VM attacks the hypervisor to access or control another virtual machine and its resources. This is a threat vector unique to virtualization and represents considerable risk as a multitude of virtual machine file images could be compromised due to vulnerability in a single application. This virtual network attack is a major concern as the administrative techniques for protecting real networks do not directly apply to the virtual environment.

Each KVM-based virtual machine is a process which is labeled by SELinux, effectively establishing a security boundary around each virtual machine. This security boundary is monitored and enforced by the Linux kernel, restricting the virtual machine's access to resources outside of its boundary such as host machine data files or other VMs.



As shown above, sVirt isolation is provided regardless of the guest Operating System running inside the virtual machine—Linux or Windows VMs can be used. Additionally, many Linux distributions provide SELinux within the operating system, allowing the virtual machine to protect internal virtual resources from threats.

Labels and categories

KVM-based virtual machine instances are labelled with their own SELinux data type, known as svirt_image_t. Kernel level protections prevent unauthorized system processes, such as malware, from manipulating the virtual machine image files on disk. When virtual machines are powered off, images are stored as svirt_image_t as shown below:

```
system_u:object_r:svirt_image_t:SystemLow image1
system_u:object_r:svirt_image_t:SystemLow image2
system_u:object_r:svirt_image_t:SystemLow image3
system_u:object_r:svirt_image_t:SystemLow image4
```

The `svirt_image_t` label uniquely identifies image files on disk, allowing for the SELinux policy to restrict access. When a KVM-based Compute image is powered on, sVirt appends a random numerical identifier to the image. sVirt is capable of assigning numeric identifiers to a maximum of 524,288 virtual machines per hypervisor node, however most OpenStack deployments are highly unlikely to encounter this limitation.

This example shows the sVirt category identifier:

```
system_u:object_r:svirt_image_t:s0:c87,c520 image1
system_u:object_r:svirt_image_t:s0:419,c172 image2
```

SELinux users and roles

SELinux can also manage user roles. These can be viewed through the `-Z` flag, or with the **semanage** command. On the hypervisor, only administrators should be able to access the system, and should have an appropriate context around both the administrative users and any other users that are on the system.

- SELinux users documentation: SELinux.org Users and Roles Overview

Booleans

To ease the administrative burden of managing SELinux, many enterprise Linux platforms utilize SELinux Booleans to quickly change the security posture of sVirt.

Red Hat Enterprise Linux-based KVM deployments utilize the following sVirt booleans:

sVirt SELinux Boolean	Description
virt_use_common	Allow virt to use serial/parallel communication ports.
virt_use_fusefs	Allow virt to read FUSE mounted files.
virt_use_nfs	Allow virt to manage NFS mounted files.
virt_use_samba	Allow virt to manage CIFS mounted files.
virt_use_sanlock	Allow confined virtual guests to interact with the sanlock.
virt_use_sysfs	Allow virt to manage device configuration (PCI).
virt_use_usb	Allow virt to use USB devices.
virt_use_xserver	Allow virtual machine to interact with the X Window System.

Case studies

Earlier in the section called "Introduction to case studies" [21] we introduced the Alice and Bob case studies where Alice is deploying a private government cloud and Bob is deploying a public cloud each with different security requirements. Here we discuss how Alice and Bob would ensure that their instances are properly isolated. First we consider hypervisor selection, and then techniques for hardening QEMU and applying mandatory access controls.

Alice's private cloud

Alice chooses Xen for the hypervisor in her cloud due to a strong internal knowledge base and a desire to use the Xen security modules (XSM) for fine-grained policy enforcement.

Alice is willing to apply a relatively large amount of resources to software packaging and maintenance. She will use these resources to build a highly customized version of QEMU that has many components removed, thereby reducing the attack surface. She will also ensure that all compiler hardening options are enabled for QEMU. Alice accepts that these decisions will increase long-term maintenance costs.

Alice writes XSM policies (for Xen) and SELinux policies (for Linux domain 0, and device domains) to provide stronger isolation between the in-

stances. Alice also uses the Intel TXT support in Xen to measure the hypervisor launch in the TPM.

Bob's public cloud

Bob is very concerned about instance isolation since the users in a public cloud represent anyone with a credit card, meaning they are inherently untrusted. Bob has just started hiring the team that will deploy the cloud, so he can tailor his candidate search for specific areas of expertise. With this in mind, Bob chooses a hypervisor based on its technical features, certifications, and community support. KVM has an EAL 4+ common criteria rating, with a labeled security protection profile (LSPP) to provide added assurance for instance isolation. This, combined with the strong support for KVM within the OpenStack community drives Bob's decision to use KVM.

Bob weighs the added cost of repackaging QEMU and decides that he cannot commit those resources to the project. Fortunately, his Linux distribution has already enabled the compiler hardening options. So he decides to use this QEMU package. Finally, Bob leverages sVirt to manage the SELinux polices associated with the virtualization stack.

17. Instance security management

One of the virtues of running instances in a virtualized environment is that it opens up new opportunities for security controls that are not typically available when deploying onto bare metal. There are several technologies that can be applied to the virtualization stack that bring improved information assurance for cloud tenants.

Deployers or users of OpenStack with strong security requirements may want to consider deploying these technologies. Not all are applicable in every situation, indeed in some cases technologies may be ruled out for use in a cloud because of prescriptive business requirements. Similarly some technologies inspect instance data such as run state which may be undesirable to the users of the system.

In this chapter we explore these technologies and describe the situations where they can be used to enhance security for instances or underlying instances. We also seek to highlight where privacy concerns may exist. These include data pass through, introspection, or providing a source of entropy. In this section we highlight the following additional security services:

- Entropy to instances
- Scheduling instances to nodes
- Trusted images
- Instance migrations
- Monitoring, alerting, and reporting
- Updates and patches
- Firewalls and other host-based security controls

Security services for instances

Entropy to instances

We consider entropy to refer to the quality and source of random data that is available to an instance. Cryptographic technologies typically rely heavily on randomness, requiring a high quality pool of entropy to draw

from. It is typically hard for a virtual machine to get enough entropy to support these operations, which is referred to as entropy starvation. Entropy starvation can manifest in instances as something seemingly unrelated. For example, slow boot time may be caused by the instance waiting for ssh key generation. Entropy starvation may also motivate users to employ poor quality entropy sources from within the instance, making applications running in the cloud less secure overall.

Fortunately, a cloud architect may address these issues by providing a high quality source of entropy to the cloud instances. This can be done by having enough hardware random number generators (HRNG) in the cloud to support the instances. In this case, "enough" is somewhat domain specific. For everyday operations, a modern HRNG is likely to produce enough entropy to support 50-100 compute nodes. High bandwidth HRNGs, such as the RdRand instruction available with Intel Ivy Bridge and newer processors could potentially handle more nodes. For a given cloud, an architect needs to understand the application requirements to ensure that sufficient entropy is available.

The Virtio RNG is a random number generator that uses `/dev/random` as the source of entropy by default, however can be configured to use a hardware RNG or a tool such as the entropy gathering daemon (EGD) to provide a way to fairly and securely distribute entropy through a distributed system. The Virtio RNG is enabled using the `hw_rng` property of the metadata used to create the instance.

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 OpenStack Compute, although other schedulers exist (see the section Scheduling in the *OpenStack Configuration Reference*). This works in collaboration with 'filter hints' to decide where an instance should be started. This process of host selection allows administrators to fulfill many different security and compliance requirements. Depending on the cloud deployment type for example, one could choose to have tenant instances reside on the same hosts whenever possible if data isolation was a primary concern. Conversely one could attempt to have instances for a tenant reside on as many different hosts as possible for availability or fault tolerance reasons.

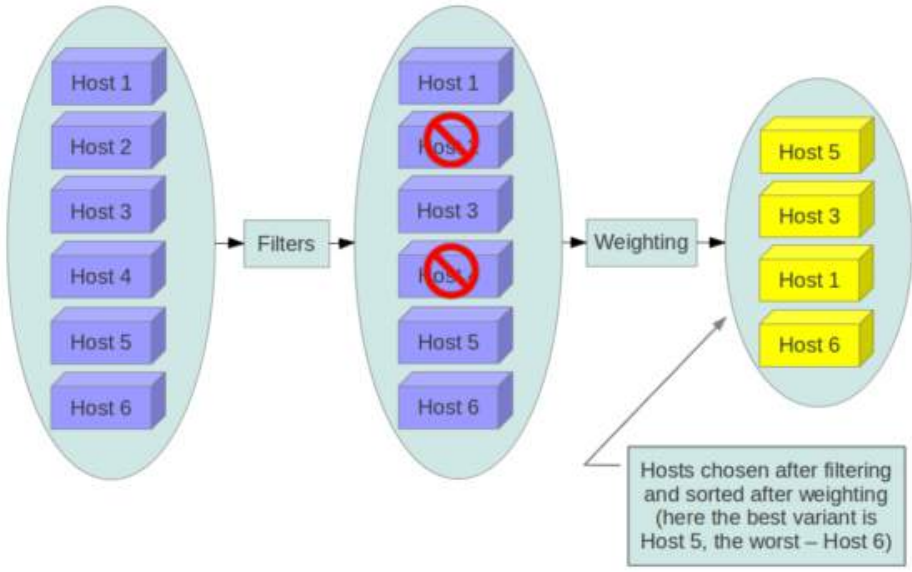
Scheduler filters may be used to segregate customer data, or even discard machines of the cloud that cannot be attested as secure. This generally ap-

plies to all OpenStack projects offering a scheduler. When building a cloud, you may choose to implement scheduling filters for a variety of security-related purposes.

Filter schedulers fall under four main categories:

Resource based filters	These filters will create an instance based on the utilizations of the hypervisor host sets and can trigger on free or used properties such as RAM, IO, or CPU utilization
Image based filters	This delegates instance creation based on the image used, such as the operating system of the VM or type of image used.
Environment based filters	This filter will create an instance based on external details such as in a specific IP range, across availability zones, or on the same host as another instance.
Custom criteria	This filter will delegate instance creation based on user or administrator provided criteria such as trusts or metadata parsing.

Multiple filters can be applied at once, such as the `ServerGroupAffinity` filter to ensure an instance is created on a member of a specific set of hosts and `ServerGroupAntiAffinity` filter to ensure that same instance is not created on another specific set of hosts. These filters should be analyzed carefully to ensure they do not conflict with each other and result in rules that prevent the creation of instances.



The `GroupAffinity` and `GroupAntiAffinity` filters conflict and should not both be enabled at the same time.

The `DiskFilter` filter is capable of oversubscribing disk space. While not normally an issue, this can be a concern on storage devices that are thinly provisioned, and this filter should be used with well-tested quotas applied.

We recommend you disable filters that parse things that are provided by users or are able to be manipulated such as metadata.

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 utilizing has not been tampered with. This requires a method of validation, such as a checksum for the known good image as well as verification of a running instance. While there are current best practices around these actions there are also several gaps in the process.

Image creation process

The OpenStack Documentation provides guidance on how to create and upload an image to the Image service. Additionally it is assumed that you

have a process by which you install and harden operating systems. Thus, the following items will provide additional guidance on how to ensure your images are transferred securely into OpenStack. There are a variety of options for obtaining images. Each has specific steps that help validate the image's provenance.

The first option is to obtain boot media from a trusted source.

```
$ mkdir -p /tmp/download_directorycd /tmp/download_directory
$ wget http://mirror.anl.gov/pub/ubuntu-iso/CDs/precise/
ubuntu-12.04.2-server-amd64.iso
$ wget http://mirror.anl.gov/pub/ubuntu-iso/CDs/precise/
SHA256SUMS
$ wget http://mirror.anl.gov/pub/ubuntu-iso/CDs/precise/
SHA256SUMS.gpg
$ gpg --keyserver hkp://keyserver.ubuntu.com --recv-keys
 0xFBB75451
$ gpg --verify SHA256SUMS.gpg SHA256SUMSsha256sum -c SHA256SUMS
 2>&1 | grep OK
```

The second option is to use the *OpenStack Virtual Machine Image Guide* . In this case, you will want to follow your organizations OS hardening guidelines or those provided by a trusted third-party such as the Linux STIGs.

The final option is to use an automated image builder. The following example uses the Oz image builder. The OpenStack community has recently created a newer tool worth investigating: disk-image-builder. We have not evaluated this tool from a security perspective.

Example of RHEL 6 CCE-26976-1 which will help implement NIST 800-53 Section*AC-19(d) in* Oz.

```
<template>
<name>centos64</name>
<os>
  <name>RHEL-6</name>
  <version>4</version>
  <arch>x86_64</arch>
  <install type='iso'>
  <iso>http://trusted_local_iso_mirror/isos/x86_64/RHEL-6.4-
x86_64-bin-DVD1.iso</iso>
  </install>
  <rootpw>CHANGE THIS TO YOUR ROOT PASSWORD</rootpw>
</os>
<description>RHEL 6.4 x86_64</description>
<repositories>
  <repository name='epel-6'>
    <url>http://download.fedoraproject.org/pub/epel/6/
$basearch</url>
```

```
    <signed>no</signed>
  </repository>
</repositories>
<packages>
  <package name='epel-release'/>
  <package name='cloud-utils'/>
  <package name='cloud-init'/>
</packages>
<commands>
  <command name='update'>
  yum update
  yum clean all
  sed -i '/^HWADDR/d' /etc/sysconfig/network-scripts/ifcfg-eth0
  echo -n > /etc/udev/rules.d/70-persistent-net.rules
  echo -n > /lib/udev/rules.d/75-persistent-net-generator.rules
  chkconfig --level 0123456 autofs off
  service autofs stop
  </command>
</commands>
</template>
```

It is recommended to avoid the manual image building process as it is complex and prone to error. Additionally, using an automated system like Oz for image building or a configuration management utility like Chef or Puppet for post-boot image hardening gives you the ability to produce a consistent image as well as track compliance of your base image to its respective hardening guidelines over time.

If subscribing to a public cloud service, you should check with the cloud provider for an outline of the process used to produce their default images. If the provider allows you to upload your own images, you will want to ensure that you are able to verify that your image was not modified before using it to create an instance. To do this, refer to the following section on Image Provenance.

Image provenance and validation

Unfortunately, it is not currently possible to force Compute to validate an image hash immediately prior to starting an instance. To understand the situation, we begin with a brief overview of how images are handled around the time of image launch.

Images come from the glance service to the nova service on a node. This transfer should be protected by running over TLS. Once the image is on the node, it is verified with a basic checksum and then it's disk is expanded based on the size of the instance being launched. If, at a later time, the same image is launched with the same instance size on this node, it will be

launched from the same expanded image. Since this expanded image is not re-verified before launching, it could be tampered with and the user would not have any way of knowing, beyond a manual inspection of the files in the resulting image.

We hope that future versions of Compute and/or the Image service will offer support for validating the image hash before each instance launch. An alternative option that would be even more powerful would be allow users to sign an image and then have the signature validated when the instance is launched.

Instance migrations

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 OpenStack 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 & sync disks
4. Transfer state
5. Start the guest

Live migration risks

At various stages of the live migration process the contents of an instances run time memory and disk are transmitted over the network in plain text. Thus there are several risks that need to be addressed when using live migration. The following in-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.

Live migration mitigations

There are several methods to mitigate some of the risk associated with live migrations, the following list details some of these:

- Disable live migration
- Isolated migration network
- Encrypted live migration

Disable live migration

At this time, live migration is enabled in OpenStack by default. 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": "!",
```

Migration network

As a general practice, live migration traffic should be restricted to the management security domain, see the section called “Management” [14]. With live migration traffic, due to its plain text nature and the fact that you are transferring the contents of disk and memory of a running instance, it is recommended you further separate live migration traffic onto a dedicated network. Isolating the traffic to a dedicated network can reduce the risk of exposure.

Encrypted live migration

If there is a sufficient business case for keeping live migration enabled, then libvirtd can provide encrypted tunnels for the live migrations. However, this feature is not currently 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. Destination libvirtd host copies the instances back to an underlying hypervisor.

Monitoring, alerting, and reporting

As an OpenStack virtual machine is a server image able to be replicated across hosts, best practice in logging applies similarly between physical and virtual hosts. Operating system-level and application-level events should be logged, including access events to hosts and data, user additions and removals, changes in privilege, and others as dictated by the environment. Ideally, you can configure these logs to export 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* .

These logs should be reviewed at a regular cadence such as a live view by a network operations center (NOC), or if the environment is not large enough to necessitate a NOC, then logs should undergo a regular log review process.

Many times interesting events trigger an alert which is sent to a responder for action. Frequently this alert takes the form of an email with the messages of interest. An interesting event could be a significant failure, or known health indicator of a pending failure. Two common utilities for managing alerts are *Nagios* and *Zabbix* .

Updates and patches

A hypervisor runs independent virtual machines. This hypervisor can run in an operating system or directly on the hardware (called baremetal). Updates to the hypervisor are not propogated down to the virtual machines. For example, if a deployment is using XenServer and has a set of Debian virtual machines, an update to XenServer will not update anything running on the Debian virtual machines.

Therefore, we recommend that clear ownership of virtual machines be assigned, and that those owners be responsible for the hardening, deployment, and continued functionality of the virtual machines. We also recommend that updates be deployed on a regular schedule. These patches should be tested in an environment as closely resembling production

as possible to ensure both stability and resolution of the issue behind the patch.

Firewalls and other host-based security controls

Most common operating systems include host-based firewalls for additional security. While we recommend that virtual machines run as few applications as possible (to the point of being single-purpose instances, if possible), all applications running on a virtual machine should be profiled to determine what 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 (or whitelisted), 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 virtual machines, 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.

OpenStack provides security groups for both hosts and the network to add defense in depth to the virtual machines 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 are able to 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. We recommend ensuring that security groups are configured correctly for the networking being used. See the section called "Security groups" [120] in this guide for more detail.

Case studies

Earlier in the section called "Introduction to case studies" [21] we introduced the Alice and Bob case studies where Alice is deploying a private government cloud and Bob is deploying a public cloud each with different security requirements. Here we discuss how Alice and Bob would architect their clouds with respect to instance entropy, scheduling instances, trusted images, and instance migrations.

Alice's private cloud

Alice has a need for lots of high quality entropy in the instances. For this reason, she decides to purchase hardware with Intel Ivy Bridge chip sets that support the RdRand instruction on each compute node. Using the entropy gathering daemon (EGD) and libvirt's EGD support, Alice ensures that this entropy pool is distributed to the instances on each compute node.

For instance scheduling, Alice uses the trusted compute pools to ensure that all cloud workloads are deployed to nodes that presented a proper boot time attestation. Alice decides to disable user permissions for image uploading to help ensure that the images used in the cloud are generated in a known and trusted manner by the cloud administrators.

Finally, Alice disables instance migrations as this feature is less critical for the high performance application workloads expected to run in this cloud. This helps avoid the various security concerns related to instance migrations.

Bob's public cloud

Bob is aware that entropy will be a concern for some of his customers, such as those in the financial industry. However, due to the added cost and complexity, Bob has decided to forgo integrating hardware entropy into the first iteration of his cloud. He adds hardware entropy as a fast-follow to do for a later improvement for the second generation of his cloud architecture.

Bob is interested in ensuring that customers receive a high quality of service. He is concerned that providing excess explicit user control over instance scheduling could negatively impact the quality of service. As a result, he disables this feature. Bob provides images in the cloud from a known trusted source for users to use. Additionally, he allows users to upload their own images. However, users generally cannot share their images. This helps prevent a user from sharing a malicious image, which could negatively impact the security of other users in the cloud.

For migrations, Bob wants to enable secure instance migrations in order to support rolling upgrades with minimal user downtime. Bob ensures that all migrations occur on an isolated VLAN. He plans to defer implementing encrypted migrations until this is better supported in **nova** client tools. As a result, he makes a note to track this carefully and switch to encrypted migrations as soon as possible.

18. Monitoring and logging

A lot of activity goes on within a cloud environment. It is a mix of hardware, operating systems, virtual machine managers, the OpenStack services, cloud-user activity such as creating instances and attaching storage, the network underlying the whole, and finally end-users using the applications running on the various instances.

The basics of logging: configuration, setting log level, location of the log files, and how to use and customize logs, as well as how to do centralized collections of logs is well covered in the *OpenStack Operations Guide*.

Forensics and incident response

The generation and collection of logs is an important component of securely monitoring an OpenStack infrastructure. Logs provide visibility into the day-to-day actions of administrators, tenants, and guests, in addition to the activity in the compute, networking, and storage and other components 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 investigating and responding to incidents.

For instance, analyzing the access logs of Identity service or its replacement authentication system would alert us to failed logins, frequency, origin IP, whether the events are restricted to select accounts and other pertinent information. Log analysis supports detection.

Actions may be taken to mitigate potential malicious activity such as blacklisting an IP address, recommending the strengthening of user passwords, or de-activating a user account if it is deemed dormant.

Monitoring use cases

Event monitoring is a more pro-active approach to securing an environment, providing real-time detection and response. Several tools exist which can aid in monitoring.

In the case of an OpenStack cloud instance, we need to monitor the hardware, the OpenStack services, and the cloud resource usage. The latter

stems from wanting to be elastic, to scale to the dynamic needs of the users.

Here are a few important use cases to consider when implementing log aggregation, analysis and monitoring. These use cases can be implemented and monitored through various applications, tools or scripts. There are open source and commercial solutions and some operators develop their own in-house solutions. These tools and scripts can generate events that can be sent to administrators through email or viewed in the integrated dashboard. It is important to consider additional use cases that may apply to your specific network and what you may consider anomalous behavior.

- Detecting the absence of log generation is an event of high value. Such an event would 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 would also be events to monitor and examine for possible security implications.

- Operating system events on the OpenStack service machines such as user logins or restarts also provide valuable insight into proper and improper usage of systems.

- Being able to detect the load on the OpenStack servers also enables responding by way of introducing additional servers for load balancing to ensure high availability.

- Other events that are actionable are networking bridges going down, ip tables being flushed on compute nodes and consequential loss of access to instances resulting in unhappy customers.

- To reduce security risks from orphan instances on a user, tenant, 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.

A cloud will host many virtual instances, and monitoring these instances goes beyond hardware monitoring and log files which may just contain CRUD events.

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. Deploying these tools on the cloud machines provides value and protection. Cloud users, those running instances on the cloud, may also want to run such tools on their instances.

Bibliography

Siwczak, Piotr. Some Practical Considerations for Monitoring in the OpenStack Cloud. 2012. http://www.mirantis.com/blog/openstack-monitoring

blog.sflow.com, sflow: Host sFlow distributed agent. 2012. http://blog.sflow.com/2012/01/host-sflow-distributed-agent.html

blog.sflow.com, sflow: LAN and WAN. 2009. http://blog.sflow.com/2009/09/lan-and-wan.html

blog.sflow.com, sflow: Rapidly detecting large flows sFlow vs. NetFlow/IPFIX. 2013. http://blog.sflow.com/2013/01/rapidly-detecting-large-flows-sflow-vs.html

Case studies

Earlier in the section called "Introduction to case studies" [21] we introduced the Alice and Bob case studies where Alice is deploying a private government cloud and Bob is deploying a public cloud each with different security requirements. Here we discuss how Alice and Bob would address monitoring and logging in the public vs a private cloud. In both instances, time synchronization and a centralized store of logs become extremely important for performing proper assessments and troubleshooting of anomalies. Just collecting logs is not very useful, a robust monitoring system must be built to generate actionable events.

Alice's private cloud

In the private cloud, Alice has a better understanding of the tenants' requirements thus she has the ability to add appropriate oversight, actively enforcing compliance for monitoring and logging activities. Alice should identify critical services and data to verify that logging is turned on for each of the services while ensuring the information is being aggregated to a central log server. She should start with simple, known use cases then implement correlation and alerting to limit the number of false positives. To implement correlation and alerting, she sends the log data to her organization's existing SIEM tool. Security monitoring should be an ongo-

ing process therefore she should continue to define use cases and alerts in order to have a better understanding of the network traffic activity and usage over time.

Bob's public cloud

When it comes to logging, as a public cloud provider, Bob is interested in the activities for situational awareness as well as compliance. In the aspect of compliance, as a provider, Bob is subject to adherence to various rules and regulations to include activities such as providing timely, relevant logs or reports to customers to meet the requirements of their compliance programs. With that in mind, Bob configures all of his instances, nodes, and infrastructure devices to perform time synchronization with an external, validated time device. Additionally, Bob's team has built a Django based web application for his customers to perform self-service log retrieval from the SIEM tool. Bob also uses this SIEM tool along with a robust set of alerts and integration with his CMDB to provide operational awareness to both customers and cloud administrators.

19. Compliance

An OpenStack deployment may require compliance activities for many purposes, such as regulatory and legal requirements, customer need, privacy considerations, and security best practices. The Compliance function is important for the business and its customers. Compliance means adhering to regulations, specifications, standards and laws. It is also used when describing an organizations status regarding assessments, audits, and certifications. Compliance, when done correctly, unifies and strengthens the other security topics discussed in this guide.

This chapter has several objectives:

- Review common security principles.
- Discuss common control frameworks and certification resources to achieve industry certifications or regulator attestations.
- Act as a reference for auditors when evaluating OpenStack deployments.
- Introduce privacy considerations specific to OpenStack and cloud environments.

Compliance overview

Security principles

Industry standard security principles provide a baseline for compliance certifications and attestations. If these principles are considered and referenced throughout an OpenStack deployment, certification activities may be simplified.

Layered defenses Identify where risks exist in a cloud architecture and apply controls to mitigate the risks. In areas

	of significant concern, layered defences provide multiple complementary controls to manage risk down to an acceptable level. For example, to ensure adequate isolation between cloud tenants, we recommend hardening QEMU, using a hypervisor with SELinux support, enforcing mandatory access control policies, and reducing the overall attack surface. The foundational principle is to harden an area of concern with multiple layers of defense such that if any one layer is compromised, other layers will exist to offer protection and minimize exposure.
Fail securely	In the case of failure, systems should be configured to fail into a closed secure state. For example, TLS certificate verification should fail closed by severing the network connection if the CNAME doesn't match the server's DNS name. Software often fails open in this situation, allowing the connection to proceed without a CNAME match, which is less secure and not recommended.
Least privilege	Only the minimum level of access for users and system services is granted. This access is based upon role, responsibility and job function. This security principal of least privilege is written into several international government security policies, such as NIST 800-53 Section AC-6 within the United States.
Compartmentalize	Systems should be segregated in a such way that if one machine, or system-level service, is compromised the security of the other systems will remain intact. Practically, the enablement and proper usage of SELinux helps accomplish this goal.
Promote privacy	The amount of information that can be gathered about a system and its users should be minimized.
Logging capability	Appropriate logging is implemented to monitor for unauthorized use, incident response

and forensics. It is highly recommended that selected audit subsystems be Common Criteria certified, which provides non-attestable event records in most countries.

Common control frameworks

The following is a list of Control Frameworks that an organization can use to build their security controls

Cloud Security Alliance (CSA) Common Control Matrix (CCM)

The CSA CCM is specifically designed to provide fundamental security principles to guide cloud vendors and to assist prospective cloud customers in assessing the overall security risk of a cloud provider. The CSA CCM provides a controls framework that are aligned across 16 security domains. The foundation of the Cloud Controls Matrix rests on its customized relationship to other industry standards, regulations, and controls frameworks such as: ISO 27001:2013, COBIT 5.0, PCI:DSS v3, AICPA 2014 Trust Service Principles and Criteria and augments internal control direction for service organization control reports attestations.

The CSA CCM strengthens existing information security control environments by enabling the reduction of security threats and vulnerabilities in the cloud, provides standardized security and operational risk management, and seeks to normalize security expectations, cloud taxonomy and terminology, and security measures implemented in the cloud.

ISO 27001/2:2013

The ISO 27001 Information Security standard and certification has been used for many years to evaluate and distinguish an organizations alignment with information Security best prac-

	tices. The standard is comprised of two parts: Mandatory Clauses that define the Information Security Management System (ISMS) and Annex A which contains a list of controls organized by domain. The information security management system preserves the confidentiality, integrity, and availability of information by applying a risk management process and gives confidence to interested parties that risks are adequately managed.
Trusted Security Principles	Trust Services are a set of professional attestation and advisory services based on a core set of principles and criteria that address the risks and opportunities of IT-enabled systems and privacy programs. Commonly known as the SOC audits, the principles define what the requirement is and it is the organizations responsibility to define the control that meets the requirement.

Audit reference

OpenStack is innovative in many ways however the process used to audit an OpenStack deployment is fairly common. Auditors will evaluate a process by two criteria: Is the control designed effectively and if the control is operating effectively. An understanding of how an auditor evaluates if a control is designed and operating effectively will be discussed in the section called "Understanding the audit process" [201].

The most common frameworks for auditing and evaluating a cloud deployment include the previously mentioned ISO 27001/2 Information Security standard, ISACA's Control Objectives for Information and Related Technology (COBIT) framework, Committee of Sponsoring Organizations of the Treadway Commission (COSO), and Information Technology Infrastructure Library (ITIL). It is very common for audits to include areas of focus from one or more of these frameworks. Fortunately there is a lot of overlap between the frameworks, so an organization that adopts one will be in a good position come audit time.

Understanding the audit process

Information system security compliance is reliant on the completion of two foundational processes:

1. **Implementation and operation of security controls.** Aligning the information system with in-scope standards and regulations involves internal tasks which must be conducted before a formal assessment. Auditors may be involved at this state to conduct gap analysis, provide guidance, and increase the likelihood of successful certification.

2. **Independent verification and validation.** Demonstration to a neutral third-party that system security controls are implemented and operating effectively, in compliance with in-scope standards and regulations, is required before many information systems achieve certified status. Many certifications require periodic audits to ensure continued certification, considered part of an overarching continuous monitoring practice.

Determining audit scope

Determining audit scope, specifically what controls are needed and how to design or modify an OpenStack deployment to satisfy them, should be the initial planning step.

When scoping OpenStack deployments for compliance purposes, consider prioritizing controls around sensitive services, such as command and control functions and the base virtualization technology. Compromises of these facilities may impact an OpenStack environment in its entirety.

Scope reduction helps ensure OpenStack architects establish high quality security controls which are tailored to a particular deployment, however it is paramount to ensure these practices do not omit areas or features from security hardening. A common example is applicable to PCI-DSS guidelines, where payment related infrastructure may be scrutinized for security issues, but supporting services are left ignored, and vulnerable to attack.

When addressing compliance, you can increase efficiency and reduce work effort by identifying common areas and criteria that apply across multiple certifications. Much of the audit principles and guidelines discussed in this book will assist in identifying these controls, additionally a number of external entities provide comprehensive lists. The following are some examples:

The Cloud Security Alliance Cloud Controls Matrix (CCM) assists both cloud providers and consumers in assessing the overall security of a cloud

provider. The CSA CMM provides a controls framework that map to many industry-accepted standards and regulations including the ISO 27001/2, ISACA, COBIT, PCI, NIST, Jericho Forum and NERC CIP.

The SCAP Security Guide is another useful reference. This is still an emerging source, but we anticipate that this will grow into a tool with controls mappings that are more focused on the US federal government certifications and recommendations. For example, the SCAP Security Guide currently has some mappings for security technical implementation guides (STIGs) and NIST-800-53.

These control mappings will help identify common control criteria across certifications, and provide visibility to both auditors and auditees on problem areas within control sets for particular compliance certifications and attestations.

Internal audit

Once a cloud is deployed, it is time for an internal audit. This is the time compare the controls you identified above with the design, features, and deployment strategies utilized in your cloud. The goal is to understand how each control is handled and where gaps exist. Document all of the findings for future reference.

When auditing an OpenStack cloud it is important to appreciate the multi-tenant environment inherent in the OpenStack architecture. Some critical areas for concern include data disposal, hypervisor security, node hardening, and authentication mechanisms.

Prepare for external audit

Once the internal audit results look good, it is time to prepare for an external audit. There are several key actions to take at this stage, these are outlined below:

- Maintain good records from your internal audit. These will prove useful during the external audit so you can be prepared to answer questions about mapping the compliance controls to a particular deployment.
- Deploy automated testing tools to ensure that the cloud remains compliant over time.
- Select an auditor.

Selecting an auditor can be challenging. Ideally, you are looking for someone with experience in cloud compliance audits. OpenStack experience is another big plus. Often it is best to consult with people who have been through this process for referrals. Cost can vary greatly depending on the scope of the engagement and the audit firm considered.

External audit

This is the formal audit process. Auditors will test security controls in scope for a specific certification, and demand evidentiary requirements to prove that these controls were also in place for the audit window (for example SOC 2 audits generally evaluate security controls over a 6-12 months period). Any control failures are logged, and will be documented in the external auditors final report. Dependent on the type of OpenStack deployment, these reports may be viewed by customers, so it is important to avoid control failures. This is why audit preparation is so important.

Compliance maintenance

The process doesn't end with a single external audit. Most certifications require continual compliance activities which means repeating the audit process periodically. We recommend integrating automated compliance verification tools into a cloud to ensure that it is compliant at all times. This should be in done in addition to other security monitoring tools. Remember that the goal is both security *and* compliance. Failing on either of these fronts will significantly complicate future audits.

Compliance activities

There are a number of standard activities that will greatly assist with the compliance process. In this chapter we outline some of the most common compliance activities. These are not specific to OpenStack, however we provide references to relevant sections in this book as useful context.

Information Security Management system (ISMS)

An Information Security Management System (ISMS) is a comprehensive set of policies and processes that an organization creates and maintains to manage risk to information assets. The most common ISMS for cloud deployments is ISO/IEC 27001/2, which creates a solid foundation of security controls and practices for achieving more stringent compliance certifications. This standard was updated in 2013 to reflect the growing use

of cloud services and places more emphasis on measuring and evaluating how well an organization's ISMS is performing.

Risk assessment

A risk assessment framework identifies risks within an organization or service, and specifies ownership of these risks, along with implementation and mitigation strategies. Risks apply to all areas of the service, from technical controls to environmental disaster scenarios and human elements, for example a malicious insider (or rogue employee). Risks can be rated using a variety of mechanisms, for example likelihood vs impact. An OpenStack deployment risk assessment can include control gaps that are described in this book.

Access and log reviews

Periodic access and log reviews are required to ensure authentication, authorization, and accountability in a service deployment. Specific guidance for OpenStack on these topics are discussed in-depth in the logging section.

Backup and disaster recovery

Disaster Recovery (DR) and Business Continuity Planning (BCP) plans are common requirements for ISMS and compliance activities. These plans must be periodically tested as well as documented. In OpenStack key areas are found in the management security domain, and anywhere that single points of failure (SPOFs) can be identified. See the section on secure backup and recovery for additional details.

Security training

Annual, role-specific, security training is a mandatory requirement for almost all compliance certifications and attestations. To optimize the effectiveness of security training, a common method is to provide role specific training, for example to developers, operational personnel, and non-technical employees. Additional cloud security or OpenStack security training based on this hardening guide would be ideal.

Security reviews

As OpenStack is a popular open source project, much of the codebase and architecture has been scrutinized by individual contributors, organizations

and enterprises. This can be advantageous from a security perspective, however the need for security reviews is still a critical consideration for service providers, as deployments vary, and security is not always the primary concern for contributors. A comprehensive security review process may include architectural review, threat modelling, source code analysis and penetration testing. There are many techniques and recommendations for conducting security reviews that can be found publicly posted. A well-tested example is the Microsoft SDL, created as part of the Microsoft Trustworthy Computing Initiative.

Vulnerability management

Security updates are critical to any IaaS deployment, whether private or public. Vulnerable systems expand attack surfaces, and are obvious targets for attackers. Common scanning technologies and vulnerability notification services can help mitigate this threat. It is important that scans are authenticated and that mitigation strategies extend beyond simple perimeter hardening. Multi-tenant architectures such as OpenStack are particularly prone to hypervisor vulnerabilities, making this a critical part of the system for vulnerability management. See the section on instance isolation for additional details.

Data classification

Data Classification defines a method for classifying and handling information, often to protect customer information from accidental or deliberate theft, loss, or inappropriate disclosure. Most commonly this involves classifying information as sensitive or non-sensitive, or as personally identifiable information (PII). Depending on the context of the deployment various other classifying criteria may be used (government, health-care etc). The underlying principle is that data classifications are clearly defined and in-use. The most common protective mechanisms include industry standard encryption technologies. See the data security section for additional details.

Exception process

An exception process is an important component of an ISMS. When certain actions are not compliant with security policies that an organization has defined, they must be logged. Appropriate justification, description and mitigation details need to be included, and signed off by appropriate authorities. OpenStack default configurations may vary in meeting various compliance criteria, areas that fail to meet compliance requirements

should be logged, with potential fixes considered for contribution to the community.

Certification and compliance statements

Compliance and security are not exclusive, and must be addressed together. OpenStack deployments are unlikely to satisfy compliance requirements without security hardening. The listing below provides an OpenStack architect foundational knowledge and guidance to achieve compliance against commercial and government certifications and standards.

Commercial standards

For commercial deployments of OpenStack, it is recommended that SOC 1/2 combined with ISO 2700 1/2 be considered as a starting point for OpenStack certification activities. The required security activities mandated by these certifications facilitate a foundation of security best practices and common control criteria that can assist in achieving more stringent compliance activities, including government attestations and certifications.

After completing these initial certifications, the remaining certifications are more deployment specific. For example, clouds processing credit card transactions will need PCI-DSS, clouds storing health care information require HIPAA, and clouds within the federal government may require FedRAMP/FISMA, and ITAR, certifications.

SOC 1 (SSAE 16) / ISAE 3402

Service Organization Controls (SOC) criteria are defined by the American Institute of Certified Public Accountants (AICPA). SOC controls assess relevant financial statements and assertions of a service provider, such as compliance with the Sarbanes-Oxley Act. SOC 1 is a replacement for Statement on Auditing Standards No. 70 (SAS 70) Type II report. These controls commonly include physical data centers in scope.

There are two types of SOC 1 reports:

- Type 1 - report on the fairness of the presentation of management's description of the service organization's system and the suitability of the design of the controls to achieve the related control objectives included in the description as of a specified date.

- Type 2 - report on the fairness of the presentation of management's description of the service organization's system and the suitability of the

design and operating effectiveness of the controls to achieve the related control objectives included in the description throughout a specified period

For more details see the AICPA Report on Controls at a Service Organization Relevant to User Entities' Internal Control over Financial Reporting.

SOC 2

Service Organization Controls (SOC) 2 is a self attestation of controls that affect the security, availability, and processing integrity of the systems a service organization uses to process users' data and the confidentiality and privacy of information processed by these system. Examples of users are those responsible for governance of the service organization; customers of the service organization; regulators; business partners; suppliers and others who have an understanding of the service organization and its controls.

There are two types of SOC 2 reports:

- Type 1 - report on the fairness of the presentation of management's description of the service organization's system and the suitability of the design of the controls to achieve the related control objectives included in the description as of a specified date.

- Type 2 - report on the fairness of the presentation of management's description of the service organization's system and the suitability of the design and operating effectiveness of the controls to achieve the related control objectives included in the description throughout a specified period.

For more details see the AICPA Report on Controls at a Service Organization Relevant to Security, Availability, Processing Integrity, Confidentiality or Privacy.

SOC 3

Service Organization Controls (SOC) 3 is a trust services report for service organizations. These reports are designed to meet the needs of users who want assurance on the controls at a service organization related to security, availability, processing integrity, confidentiality, or privacy but do not have the need for or the knowledge necessary to make effective use of a SOC 2 Report. These reports are prepared using the AICPA/Canadian Institute of Chartered Accountants (CICA) Trust Services Principles, Criteria, and

Illustrations for Security, Availability, Processing Integrity, Confidentiality, and Privacy. Because they are general use reports, SOC 3 Reports can be freely distributed or posted on a website as a seal.

For more details see the AICPA Trust Services Report for Service Organizations.

ISO 27001/2

The ISO/IEC 27001/2 standards replace BS7799-2, and are specifications for an Information Security Management System (ISMS). An ISMS is a comprehensive set of policies and processes that an organization creates and maintains to manage risk to information assets. These risks are based upon the confidentiality, integrity, and availability (CIA) of user information. The CIA security triad has been used as a foundation for much of the chapters in this book.

For more details see ISO 27001.

HIPAA / HITECH

The Health Insurance Portability and Accountability Act (HIPAA) is a United States congressional act that governs the collection, storage, use and destruction of patient health records. The act states that Protected Health Information (PHI) must be rendered "unusable, unreadable, or indecipherable" to unauthorized persons and that encryption for data 'at-rest' and 'in-flight' should be addressed.

HIPAA is not a certification, rather a guide for protecting healthcare data. Similar to the PCI-DSS, the most important issues with both PCI and HIPPA is that a breach of credit card information, and health data, does not occur. In the instance of a breach the cloud provider will be scrutinized for compliance with PCI and HIPPA controls. If proven compliant, the provider can be expected to immediately implement remedial controls, breach notification responsibilities, and significant expenditure on additional compliance activities. If not compliant, the cloud provider can expect on-site audit teams, fines, potential loss of merchant ID (PCI), and massive reputation impact.

Users or organizations that possess PHI must support HIPAA requirements and are HIPAA covered entities. If an entity intends to use a service, or in this case, an OpenStack cloud that might use, store or have access to that PHI, then a Business Associate Agreement must be signed. The BAA is a contract between the HIPAA covered entity and the OpenStack service

provider that requires the provider to handle that PHI in accordance with HIPAA requirements. If the service provider does not handle the PHI, such as with security controls and hardening, then they are subject to HIPAA fines and penalties.

OpenStack architects interpret and respond to HIPAA statements, with data encryption remaining a core practice. Currently this would require any protected health information contained within an OpenStack deployment to be encrypted with industry standard encryption algorithms. Potential future OpenStack projects such as object encryption will facilitate HIPAA guidelines for compliance with the act.

For more details see the Health Insurance Portability And Accountability Act.

PCI-DSS

The Payment Card Industry Data Security Standard (PCI DSS) is defined by the Payment Card Industry Standards Council, and created to increase controls around card holder data to reduce credit card fraud. Annual compliance validation is assessed by an external Qualified Security Assessor (QSA) who creates a Report on Compliance (ROC), or by a Self-Assessment Questionnaire (SAQ) dependent on volume of card-holder transactions.

OpenStack deployments which stores, processes, or transmits payment card details are in scope for the PCI-DSS. All OpenStack components that are not properly segmented from systems or networks that handle payment data fall under the guidelines of the PCI-DSS. Segmentation in the context of PCI-DSS does not support multi-tenancy, but rather physical separation (host/network).

For more details see PCI security standards.

Government standards

FedRAMP

"The Federal Risk and Authorization Management Program (FedRAMP) is a government-wide program that provides a standardized approach to security assessment, authorization, and continuous monitoring for cloud products and services". NIST 800-53 is the basis for both FISMA and FedRAMP which mandates security controls specifically selected to provide protection in cloud environments. FedRAMP can be extremely intensive from specifici-

ty around security controls, and the volume of documentation required to meet government standards.

For more details see http://www.gsa.gov/portal/category/102371.

ITAR

The International Traffic in Arms Regulations (ITAR) is a set of United States government regulations that control the export and import of defense-related articles and services on the United States Munitions List (USML) and related technical data. ITAR is often approached by cloud providers as an "operational alignment" rather than a formal certification. This typically involves implementing a segregated cloud environment following practices based on the NIST 800-53 framework, as per FISMA requirements, complemented with additional controls restricting access to "U.S. Persons" only and background screening.

For more details see https://www.pmddtc.state.gov/regulations_laws/itar.html.

FISMA

The Federal Information Security Management Act requires that government agencies create a comprehensive plan to implement numerous government security standards, and was enacted within the E-Government Act of 2002. FISMA outlines a process, which utilizing multiple NIST publications, prepares an information system to store and process government data.

This process is broken apart into three primary categories:

- **System categorization:** The information system will receive a security category as defined in Federal Information Processing Standards Publication 199 (FIPS 199). These categories reflect the potential impact of system compromise.
- **Control selection:**Based upon system security category as defined in FIPS 199, an organization utilizes FIPS 200 to identify specific security control requirements for the information system. For example, if a system is categorized as "moderate" a requirement may be introduced to mandate "secure passwords".
- **Control tailoring:** Once system security controls are identified, an OpenStack architect will utilize NIST 800-53 to extract tailored control selection. For example, specification of what constitutes a "secure password".

Privacy

Privacy is an increasingly important element of a compliance program. Businesses are being held to a higher standard by their customers, who have increased interest in understanding how their data is treated from a privacy perspective.

An OpenStack deployment will likely need to demonstrate compliance with an organization's Privacy Policy, with the U.S.-E.U. Safe Harbor framework, the ISO/IEC 29100:2011 privacy framework or with other privacy-specific guidelines. In the U.S. the AICPA has defined 10 privacy areas of focus, OpenStack deployments within a commercial environment may desire to attest to some or all of these principles.

To aid OpenStack architects in the protection of personal data, it is recommended that OpenStack architects review the NIST publication 800-122, titled "*Guide to Protecting the Confidentiality of Personally Identifiable Information (PII)*." This guide steps through the process of protecting:

> "*any information about an individual maintained by an agency, including (1) any information that can be used to distinguish or trace an individual's identity, such as name, social security number, date and place of birth, mother's maiden name, or biometric records; and (2) any other information that is linked or linkable to an individual, such as medical, educational, financial, and employment information*"

Comprehensive privacy management requires significant preparation, thought and investment. Additional complications are introduced when building global OpenStack clouds, for example navigating the differences between U.S. and more restrictive E.U. privacy laws. In addition, extra care needs to be taken when dealing with sensitive PII that may include information such as credit card numbers or medical records. This sensitive data is not only subject to privacy laws but also regulatory and governmental regulations. By deferring to established best practices, including those published by governments, a holistic privacy management policy may be created and practiced for OpenStack deployments.

Case studies

Earlier in the section called “Introduction to case studies” [21] we introduced the Alice and Bob case studies where Alice is deploying a private

government cloud and Bob is deploying a public cloud each with different security requirements. Here we discuss how Alice and Bob would address common compliance requirements. The preceding chapter refers to a wide variety of compliance certifications and standards. Alice will address compliance in a private cloud, while Bob will be focused on compliance for a public cloud.

Alice's private cloud

Alice is building an OpenStack private cloud for the United States government, specifically to provide elastic compute environments for signal processing. Alice has researched government compliance requirements, and has identified that her private cloud will be required to certify against FISMA and follow the FedRAMP accreditation process, which is required for all federal agencies, departments and contractors to become a Certified Cloud Provider (CCP). In this particular scenario for signal processing, the FISMA controls required will most likely be FISMA High, which indicates possible "severe or catastrophic adverse effects" should the information system become compromised. In addition to FISMA Moderate controls Alice must ensure her private cloud is FedRAMP certified, as this is a requirement for all agencies that currently utilize, or host federal information within a cloud environment.

To meet these strict government regulations Alice undertakes a number of activities. Scoping of requirements is particularly important due to the volume of controls that must be implemented, which will be defined in NIST Publication 800-53.

All technology within her private cloud must be FIPS certified technology, as mandated within NIST 800-53 and FedRAMP. As the U.S. Department of Defense is involved, Security Technical Implementation Guides (STIGs) will come into play, which are the configuration standards for DOD IA and IA-enabled devices / systems. Alice notices a number of complications here as there is no STIG for OpenStack, so she must address several underlying requirements for each OpenStack service; for example, the networking SRG and Application SRG will both be applicable (list of SRGs). Other critical controls include ensuring that all identities in the cloud use PKI, that SELinux is enabled, that encryption exists for all wire-level communications, and that continuous monitoring is in place and clearly documented. Alice is not concerned with object encryption, as this will be the tenants responsibility rather than the provider.

If Alice has adequately scoped and executed these compliance activities, she may begin the process to become FedRAMP compliant by hiring an

approved third-party auditor. Typically this process takes up to 6 months, after which she will receive an Authority to Operate and can offer OpenStack cloud services to the government.

Bob's public cloud

Bob is tasked with compliance for a new OpenStack public cloud deployment, that is focused on providing cloud services to both small developers and startups, as well as large enterprises. Bob recognizes that individual developers are not necessarily concerned with compliance certifications, but to larger enterprises certifications are critical. Specifically Bob desires to achieve SOC 1, SOC 2 Security, as well as ISO 27001/2 as quickly as possible. Bob references the Cloud Security Alliance Cloud Control Matrix (CCM) to assist in identifying common controls across these three certifications (such as periodic access reviews, auditable logging and monitoring services, risk assessment activities, security reviews, etc). Bob then engages an experienced audit team to conduct a gap analysis on the public cloud deployment, reviews the results and fills any gaps identified. Bob works with other team members to ensure that these security controls and activities are regularly conducted for a typical audit period (~6-12 months).

At the end of the audit period Bob has arranged for an external audit team to review in-scope security controls at randomly sampled points of time over a 6 month period. The audit team provides Bob with an official report for SOC 1 and SOC 2, and separately for ISO 27001/2. As Bob has been diligent in ensuring security controls are in place for his OpenStack public cloud, there are no additional gaps exposed on the report. Bob can now provide these official reports to his customers under NDA, and advertise that he is SOC 1, SOC 2 and ISO 27001/2 compliant on his website.

Appendix A. Community support

Table of Contents

The following resources are available to help you run and use OpenStack. The OpenStack community constantly improves and adds to the main features of OpenStack, but if you have any questions, do not hesitate to ask. Use the following resources to get OpenStack support, and troubleshoot your installations.

Documentation

For the available OpenStack documentation, see docs.openstack.org.

To provide feedback on documentation, join and use the <openstack-docs@lists.openstack.org> mailing list at OpenStack Documentation Mailing List, or report a bug.

The following books explain how to install an OpenStack cloud and its associated components:

- *Installation Guide for openSUSE 13.2 and SUSE Linux Enterprise Server 12*
- *Installation Guide for Red Hat Enterprise Linux 7, CentOS 7, and Fedora 21*
- *Installation Guide for Ubuntu 14.04 (LTS)*

The following books explain how to configure and run an OpenStack cloud:

- *Architecture Design Guide*
- *Cloud Administrator Guide*
- *Configuration Reference*
- *Operations Guide*
- *Networking Guide*
- *High Availability Guide*
- *Security Guide*
- *Virtual Machine Image Guide*

The following books explain how to use the OpenStack dashboard and command-line clients:

- *API Quick Start*
- *End User Guide*
- *Admin User Guide*
- *Command-Line Interface Reference*

The following documentation provides reference and guidance information for the OpenStack APIs:

- OpenStack API Complete Reference (HTML)
- API Complete Reference (PDF)

The *Training Guides* offer software training for cloud administration and management.

ask.openstack.org

During the set up or testing of OpenStack, you might have questions about how a specific task is completed or be in a situation where a feature does not work correctly. Use the ask.openstack.org site to ask questions and get answers. When you visit the http://ask.openstack.org site, scan the recently asked questions to see whether your question has already been answered. If not, ask a new question. Be sure to give a clear, concise

summary in the title and provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.

OpenStack mailing lists

A great way to get answers and insights is to post your question or problematic scenario to the OpenStack mailing list. You can learn from and help others who might have similar issues. To subscribe or view the archives, go to http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack. You might be interested in the other mailing lists for specific projects or development, which you can find on the wiki. A description of all mailing lists is available at http://wiki.openstack.org/MailingLists.

The OpenStack wiki

The OpenStack wiki contains a broad range of topics but some of the information can be difficult to find or is a few pages deep. Fortunately, the wiki search feature enables you to search by title or content. If you search for specific information, such as about networking or nova, you can find a large amount of relevant material. More is being added all the time, so be sure to check back often. You can find the search box in the upper-right corner of any OpenStack wiki page.

The Launchpad Bugs area

The OpenStack community values your set up and testing efforts and wants your feedback. To log a bug, you must sign up for a Launchpad account at https://launchpad.net/+login. You can view existing bugs and report bugs in the Launchpad Bugs area. Use the search feature to determine whether the bug has already been reported or already been fixed. If it still seems like your bug is unreported, fill out a bug report.

Some tips:

- Give a clear, concise summary.
- Provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.
- Be sure to include the software and package versions that you are using, especially if you are using a development

branch, such as, `"Juno release" vs git commit bc79c3ecc55929bac585d04a03475b72e06a3208`.

- Any deployment-specific information is helpful, such as whether you are using Ubuntu 14.04 or are performing a multi-node installation.

The following Launchpad Bugs areas are available:

- Bugs: OpenStack Block Storage (cinder)
- Bugs: OpenStack Compute (nova)
- Bugs: OpenStack Dashboard (horizon)
- Bugs: OpenStack Identity (keystone)
- Bugs: OpenStack Image service (glance)
- Bugs: OpenStack Networking (neutron)
- Bugs: OpenStack Object Storage (swift)
- Bugs: Bare metal service (ironic)
- Bugs: Data processing service (sahara)
- Bugs: Database service (trove)
- Bugs: Orchestration (heat)
- Bugs: Telemetry (ceilometer)
- Bugs: Message Service (zaqar)
- Bugs: OpenStack API Documentation (developer.openstack.org)
- Bugs: OpenStack Documentation (docs.openstack.org)

The OpenStack IRC channel

The OpenStack community lives in the #openstack IRC channel on the Freenode network. You can hang out, ask questions, or get immediate feedback for urgent and pressing issues. To install an IRC client or use a browser-based client, go to https://webchat.freenode.net/. You can also use Colloquy (Mac OS X, http://colloquy.info/), mIRC (Windows, http://www.mirc.com/), or XChat (Linux). When you are in the IRC chan-

nel and want to share code or command output, the generally accepted method is to use a Paste Bin. The OpenStack project has one at http://paste.openstack.org. Just paste your longer amounts of text or logs in the web form and you get a URL that you can paste into the channel. The OpenStack IRC channel is `#openstack` on `irc.freenode.net`. You can find a list of all OpenStack IRC channels at https://wiki.openstack.org/wiki/IRC.

Documentation feedback

To provide feedback on documentation, join and use the <`openstack-docs@lists.openstack.org`> mailing list at OpenStack Documentation Mailing List, or report a bug.

OpenStack distribution packages

The following Linux distributions provide community-supported packages for OpenStack:

- **Debian:** http://wiki.debian.org/OpenStack
- **CentOS, Fedora, and Red Hat Enterprise Linux:** https://www.rdoproject.org/
- **openSUSE and SUSE Linux Enterprise Server:** http://en.opensuse.org/Portal:OpenStack
- **Ubuntu:** https://wiki.ubuntu.com/ServerTeam/CloudArchive

Glossary

access control list
: A list of permissions attached to an object. An ACL specifies which users or system processes have access to objects. It also defines which operations can be performed on specified objects. Each entry in a typical ACL specifies a subject and an operation. For instance, the ACL entry `(Alice, delete)` for a file gives Alice permission to delete the file.

ACL
: See access control list.

Advanced Message Queuing Protocol (AMQP)
: The open standard messaging protocol used by OpenStack components for intra-service communications, provided by RabbitMQ, Qpid, or ZeroMQ.

API
: Application programming interface.

Bell-LaPadula model
: A security model that focuses on data confidentiality and controlled access to classified information. This model divide the entities into subjects and objects. The clearance of a subject is compared to the classification of the object to determine if the subject is authorized for the specific access mode. The clearance or classification scheme is expressed in terms of a lattice.

Block Storage
: The OpenStack core project that enables management of volumes, volume snapshots, and volume types. The project name of Block Storage is cinder.

BMC
: Baseboard Management Controller. The intelligence in the IPMI architecture, which is a specialized micro-controller that is embedded on the motherboard of a computer and acts as a server. Manages the interface between system management software and platform hardware.

CA
: Certificate Authority or Certification Authority. In cryptography, an entity that issues digital certificates. The digital certificate certifies the ownership of a public key by the named subject of the certificate. This enables others (relying parties) to rely upon signatures or assertions made by the private key that corresponds to the certified public key. In this model of trust relationships, a CA is a trusted third party for both the subject (owner) of the certificate and the party relying

upon the certificate. CAs are characteristic of many public key infrastructure (PKI) schemes.

Chef
: An operating system configuration management tool supporting OpenStack deployments.

cinder
: A core OpenStack project that provides block storage services for VMs.

CMDB
: Configuration Management Database.

Compute
: The OpenStack core project that provides compute services. The project name of Compute service is nova.

DAC
: Discretionary access control. Governs the ability of subjects to access objects, while enabling users to make policy decisions and assign security attributes. The traditional UNIX system of users, groups, and read-write-execute permissions is an example of DAC.

dashboard
: The web-based management interface for OpenStack. An alternative name for horizon.

Data processing service
: OpenStack project that provides a scalable data-processing stack and associated management interfaces. The code name for the project is sahara.

DHCP
: Dynamic Host Configuration Protocol. A network protocol that configures devices that are connected to a network so that they can communicate on that network by using the Internet Protocol (IP). The protocol is implemented in a client-server model where DHCP clients request configuration data, such as an IP address, a default route, and one or more DNS server addresses from a DHCP server.

Django
: A web framework used extensively in horizon.

DNS
: Domain Name Server. A hierarchical and distributed naming system for computers, services, and resources connected to the Internet or a private network. Associates a human-friendly names to IP addresses.

federated identity
: A method to establish trusts between identity providers and the OpenStack cloud.

glance
: A core project that provides the OpenStack Image service.

horizon
: OpenStack project that provides a dashboard, which is a web interface.

HTTPS
: Hypertext Transfer Protocol Secure (HTTPS) is a communications protocol for secure communication over a computer network, with especially wide deployment on the Internet. Technically, it is not a protocol in and of itself; rather, it is the result of simply layering the Hypertext Transfer Protocol (HTTP) on top of the SSL/TLS protocol, thus adding the security capabilities of SSL/TLS to standard HTTP communications.

identity provider
: A directory service, which allows users to login with a user name and password. It is a typical source of authentication tokens.

Identity Service
: The OpenStack core project that provides a central directory of users mapped to the OpenStack services they can access. It also registers endpoints for OpenStack services. It acts as a common authentication system. The project name of the Identity Service is keystone.

Image service
: An OpenStack core project that provides discovery, registration, and delivery services for disk and server images. The project name of the Image service is glance.

keystone
: The project that provides OpenStack Identity services.

Networking
: A core OpenStack project that provides a network connectivity abstraction layer to OpenStack Compute. The project name of Networking is neutron.

neutron
: A core OpenStack project that provides a network connectivity abstraction layer to OpenStack Compute.

nova
: OpenStack project that provides compute services.

Object Storage
: The OpenStack core project that provides eventually consistent and redundant storage and retrieval of fixed digital content. The project name of OpenStack Object Storage is swift.

OpenStack
: OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data center, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. OpenStack is an open source project licensed under the Apache License 2.0.

Puppet
: An operating system configuration-management tool supported by OpenStack.

Qpid
: Message queue software supported by OpenStack; an alternative to RabbitMQ.

RabbitMQ
: The default message queue software used by OpenStack.

sahara
: OpenStack project that provides a scalable data-processing stack and associated management interfaces.

SAML assertion
: Contains information about a user as provided by the identity provider. It is an indication that a user has been authenticated.

scoped token
: An Identity Service API access token that is associated with a specific tenant.

service provider
: A system that provides services to other system entities. In case of federated identity, OpenStack Identity is the service provider.

SPICE
: The Simple Protocol for Independent Computing Environments (SPICE) provides remote desktop access to guest virtual machines. It is an alternative to VNC. SPICE is supported by OpenStack.

swift
: An OpenStack core project that provides object storage services.

unscoped token
: Alternative term for an Identity Service default token.

Virtual Network Computing (VNC)
: Open source GUI and CLI tools used for remote console access to VMs. Supported by Compute.

ZeroMQ
: Message queue software supported by OpenStack. An alternative to RabbitMQ. Also spelled 0MQ.