Troubleshooting OpenStack

OpenStack is a collection of software projects that work together to provide a cloud fabric. It puts the cloud within reach of anyone willing to learn this technology.

Starting with an introduction to OpenStack troubleshooting tools, we'll walk through each OpenStack service and how you can quickly diagnose and troubleshoot OpenStack. Understanding the various projects and how they interact is essential for anyone attempting to troubleshoot an OpenStack cloud. We will explain the major components and dependencies between them, and move on to show you how to utilize an effective set of OpenStack troubleshooting tools and fix common Keystone problems. Next, we will expose you to common errors and problems you may encounter when using the OpenStack Block Storage service (Cinder). We will then examine Heat, the OpenStack Orchestration Service, where you will learn how to trace errors, determine their root cause, and effectively correct the issue.

Finally, you will get to know the best practices to architect your OpenStack cloud in order to achieve optimal performance, availability, and reliability.

Who this book is written for
You will need a basic understanding of OpenStack, Linux, and cloud computing. If you have an understanding of Linux, this book will help you leverage that knowledge in the world of OpenStack.

What you will learn from this book
- Diagnose and fix authentication and authorization problems in Keystone
- Fix common issues with images served through Glance
- Master the art of troubleshooting Neutron networking
- Navigate and overcome problems with Nova
- Troubleshoot and resolve Cinder block storage issues
- Identify and correct Swift object storage problems
- Isolate and fix issues caused by Heat orchestration
- Leverage Ceilometer and other metering and monitoring tools for effective troubleshooting

In this package, you will find:

- The author biography
- A preview chapter from the book, Chapter 1 'The Troubleshooting Toolkit'
- A synopsis of the book’s content
- More information on Troubleshooting OpenStack
About the Author

Tony Campbell grew up in the heart of Silicon Valley where he had access and exposure to many technology companies that led the Internet boom. He started programming in the early 90s and has been hooked since then. Tony is committed to helping others understand and successfully adopt OpenStack.
Preface

OpenStack is one of the fastest growing open source projects in the history. Its rapid adoption and popularity has led to an increase in the demand of OpenStack talent. The skills you will learn in this book will help you position yourself as an effective OpenStack troubleshooter.

What this book covers

Chapter 1, *The Troubleshooting Toolkit*, covers various tools that will provide invaluable as you troubleshoot OpenStack.

Chapter 2, *Troubleshooting OpenStack Identity*, helps you to quickly recognize and resolve identity and authentication issues.

Chapter 3, *Troubleshooting the OpenStack Image Service*, fixes problems with Glance, the OpenStack Image service.

Chapter 4, *Troubleshooting OpenStack Networking*, shows you how to resolve networking problems within your OpenStack cluster.

Chapter 5, *Troubleshooting OpenStack Compute*, advertises that the compute service is central to OpenStack, and this chapter includes helpful tips for quickly getting things resolved in case the service breaks.

Chapter 6, *Troubleshooting OpenStack Block Storage*, explores ways to resolve issues with persistent storage in an OpenStack cluster.

Chapter 7, *Troubleshooting OpenStack Object Storage*, introduces you to object storage in OpenStack and helps you master the techniques to resolve common issues.

Chapter 8, *Troubleshooting the OpenStack Orchestration Service*, discusses Heat, the OpenStack Orchestration service, and how to navigate through errors or problems with this service.
Preface

Chapter 9, *Troubleshooting the OpenStack Telemetry Service*, explores Ceilometer and how to troubleshoot issues with the telemetry processes and meters.

Chapter 10, *OpenStack Performance, Availability, and Reliability*, provides you with tips for maintaining the overall health of your OpenStack cluster.
OpenStack is one of the fastest growing open source projects in history. It is rapidly becoming the standard for open source, public and private clouds. Since its first release in 2010, there have been 12 major releases, with the thirteenth being planned as of the writing of this book. The project has grown from a few thousand lines of code written by dozens of developers to over 2.6 million lines of code from over 2,100 contributors. OpenStack originally started with two projects, Object Storage (Swift) and Compute (Nova). OpenStack has grown to include over 40 projects. This huge amount of commitment and contribution has led to the momentum that OpenStack enjoys today.

OpenStack has become very popular among companies and organizations because it allows them to provide public and private clouds to their employees, partners, customers, and constituents. In addition, the vibrant community around OpenStack allows its adopters to avoid lock in and gives them freedom to work with the technologies of their choice. As an open source project, those who adopt it have the freedom to work with the community and add functionalities and features as they see fit. This flexibility has enticed hundreds of organizations to join this community, many dedicating developers to the cause.

OpenStack is extremely powerful, but it is not without complexity. One of the side effects of its rapid growth and large community involvement is the fact that things often change quickly. New projects are added regularly, and along with those projects, come new functionalities. As the community finds better ways to implement things, it often necessitates change. As the projects begin to get more and more integrated, it becomes very important to understand how these projects flow and interrelate. While the growth of OpenStack has been rapid, the development of OpenStack talent has not kept pace. As a result, individuals with OpenStack skills are in high demand.
OpenStack requires operators with the ability to identify, isolate, and troubleshoot errors that might arise in the environment. Troubleshooting OpenStack is not always straightforward because the functionality of an OpenStack cloud is delivered by several different projects all working together under the OpenStack umbrella. In addition, the OpenStack projects are further augmented by external open source technologies. With OpenStack’s power and flexibility comes the challenge of pinpointing the source of errors and problems. While this challenge is real, it is by no means insurmountable.

In this book, we will show you how to find success with OpenStack troubleshooting. We will introduce you to inside tips and a winning methodology to troubleshoot your OpenStack cluster. It is assumed that you are familiar with the basic Linux administration, cloud computing in general, and OpenStack in particular. We will walk you through a set of useful tools to troubleshoot OpenStack, and we will provide a step-by-step guide to address common problems in installation, performance, availability, and automation. We will focus on central OpenStack projects, including those providing compute, storage, and networking. By the time we reach the end of this book, you will be better prepared to tackle the OpenStack troubleshooting challenges that may come your way. You will have a better understanding of how OpenStack works under the hood, and this understanding, along with the tips and methodologies presented in this book, will make you an efficient and confident OpenStack troubleshooter.

In this chapter, we will cover the following topics:

- The project overview of OpenStack
- Basic troubleshooting methods and tools
- Installing packages

The project overview of OpenStack

The more you understand about OpenStack, how it is organized and architected, the more successful you will be at troubleshooting it. In this section, we provide you with a strong foundation of understanding about OpenStack. Throughout the book, we will build on this foundation, going deeper into each project as we encounter them in future chapters. To start the journey, we will introduce you to some of the projects that are commonly deployed in an OpenStack cluster. It’s worth pointing out that we won’t cover every OpenStack project, but we will attempt to adequately cover each of the commonly deployed projects.
Keystone
Keystone is the OpenStack Identity service. It is responsible for authentication and is involved in authorization for an OpenStack cluster. Keystone is also responsible for service discovery, allowing users to see which services are available in a cluster. A user-initiated request will typically flow through Keystone; so, learning to troubleshoot this service is a wise investment.

Glance
Glance is the OpenStack Image service. Glance is primarily responsible for image registration and storage. As an example, compute instances can be created based on machine images. These images are typically stored through and retrieved via Glance.

Neutron
Neutron is the OpenStack Networking service. Networking is hard, and it is no different in OpenStack. Neutron is responsible for abstracting the network-related functionality in OpenStack. This is an area where many operators may run into trouble. Learning how to skillfully troubleshoot Neutron will serve you well as an OpenStack administrator.

Nova
Nova is the OpenStack Compute service. Nova provides compute instances in an OpenStack cloud. This is one of the largest OpenStack projects and one of the oldest. Nova is used heavily in an OpenStack cloud, and it is critical that troubleshooters understand this project, its concepts, components, and architecture.
Cinder
Cinder is the project that provides block storage services for OpenStack. Cinder abstracts and provides access to several backend storage options. Compute instances will often receive their block storage via the Cinder service.

Swift
Swift is the OpenStack Object Storage service. Swift provides object-based storage, which is accessed via an API. Unlike Cinder, Swift does not expose block-level storage, but it does offer a system that allows you to store petabytes of data on a cluster that is built on commodity hardware.

Heat
The OpenStack Orchestration service is named Heat. Heat allows users to leverage a declarative template language to describe, build, and deploy OpenStack resources. It is designed to allow users to manage the entire life cycle of their cloud resources.

Ceilometer
Ceilometer is the OpenStack Telemetry service, and it is responsible for collecting utilization measurements from physical and virtual resources in an OpenStack cloud.

Horizon
The OpenStack dashboard is named Horizon. Horizon provides the graphical user interface for OpenStack. It relies on the OpenStack APIs to present much of this functionality. It is an extremely useful tool when troubleshooting the APIs or OpenStack functionality in general.

Oslo
Oslo is the OpenStack project that contains the shared Python libraries that are leveraged across all projects. Examples of these include code that supports messaging, command-line programs, configuration, and logging.
Documentation

One of the strengths of the OpenStack community is that it treats documentation as a first-class citizen. In the community, documentation is just as important as code. The documentation project is structured like the others and receives a good amount of exposure and attention.

In addition to the these projects, there are several other popular projects worth mentioning. These projects include the following:

<table>
<thead>
<tr>
<th>IRONIC</th>
<th>MAGNUM</th>
<th>TROVE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bare-Metal Provisioning</td>
<td>Containers</td>
<td>Database</td>
</tr>
<tr>
<td>BARBICAN</td>
<td>CONGRESS</td>
<td>DESIGNATE</td>
</tr>
<tr>
<td>Key Management</td>
<td>Governance</td>
<td>DNS Service</td>
</tr>
</tbody>
</table>

**Ironic**

While Nova typically handles the provisioning of virtual machines, with Ironic, users can provision physical hardware in a *cloudy* way. The Ironic driver allows you to deploy bare metal hardware in a similar fashion to the way you deploy virtual machines.

**Magnum**

Magnum is a project designed to allow users to manage application containers in OpenStack. This allows container orchestration engines, such as Docker and Kubernetes, to be leveraged through OpenStack.

**Trove**

Trove is an OpenStack service that provides cloud databases. The Trove service supports the provisioning of both relational and non-relational databases.

**Barbican**

Barbican is a service that facilitates the management, provisioning, and storage of secrets. Secrets include things such as passwords, encryption keys, and certificates.
Congress
Congress provides a policy as a service for OpenStack. The aim of the project is to provide a framework for regulatory compliance and governance across cloud services. Its responsibility is policy enforcement.

Designate
Designate provides DNS as a service. This service provides zone and record management as well as support for multiple nameservers.

These are just some of the many projects under the Big Tent of OpenStack. New projects with the promise of a new functionality are created regularly. As these projects gain more and more adoption, the chance that you will need to troubleshoot them increases.

The supporting technologies
One of the design tenants of OpenStack, since its inception, is to not reinvent the wheel. In other words, when a solid technology existed that met the needs of the project, the original developers would leverage the existing technology as opposed to creating their own version. The result is that OpenStack is built upon many technologies that administrators already know and love. The tools used to troubleshoot these technologies can also be used to troubleshoot OpenStack. In this section, we will go over a few of the supporting technologies that are commonly used across OpenStack projects. Where different projects use specific supporting technologies, they will be covered in the respective chapters for those projects.

Linux
The OpenStack software runs on Linux. The primary OpenStack services are Linux services. Rest-assured that all your experience in troubleshooting the Linux operating systems will serve you well in the world of OpenStack. You will come across OpenStack clusters running on just about every Linux distribution. Some deployments will leverage Linux networking, and experience in this area is extremely valuable in OpenStack. Many of the most popular Linux distributions offer packages to deploy OpenStack. Operators may optionally deploy from source or leverage one of the installers available in the market. Either way, Linux is critical to any OpenStack deployment, and we will make use of many common Linux tools when troubleshooting.
Databases
Most OpenStack services are backed by a database. The Oslo project in OpenStack provides common Python code for OpenStack projects that need to access a database. Oslo provides libraries to connect to a Postgres or MySQL database. Experience with these database engines, and others like them, is very useful when troubleshooting. As you understand the different projects and what they store in the database, you can trace a request to ensure that the state recorded in the database matches the state reported elsewhere.

Message queue
OpenStack often leverages a message broker to facilitate communication between its components. To avoid tight coupling, most components do not communicate directly with one another, but instead communication is routed through a message broker. With the message broker playing a central role in component communication, it is a powerful resource for the troubleshooter. It is possible to trace messages from one component to another and spot messages that may not be generated or delivered. This information can help lead you down the right path when attempting to isolate an issue.

The Apache web server
OpenStack projects have begun to use Web Server Gate Interface (WSGI) servers to deploy their APIs. The Apache web server is a popular choice to handle these WSGI applications. Apache troubleshooting tools and techniques are directly transferable when working with OpenStack.

Basic troubleshooting methodology and tools
There are many paths an OpenStack troubleshooter can follow when attempting to resolve an issue. It is worth arguing that there is more than one way to approach any troubleshooting problem. Operators and administrators will need to find a methodology that works well for them and the context in which they operate. With this in mind, I would like to share a methodology that I have found useful when working with OpenStack, specifically the following methodologies:

- **Services**: Confirm that the required services are up and running.
- **Authentication**: Ensure that authentication is properly configured.
- **CLI Debug**: Run the CLI commands in the debug mode, looking for errors.
• Execute the request against the API directly, looking for issues.
• **Check Logs**: Check log files for traces or errors.

I have found that working through these steps when troubleshooting OpenStack will yield useful clues that will help identify, isolate, and resolve issues.

There are many tools available when troubleshooting OpenStack. In the following sections, we will cover a few of the tools that we leverage frequently. I would recommend that you add these to your toolbox if you are not already using them.

### General Linux tools

OpenStack is deployed in a Linux environment; therefore, administrators can leverage popular Linux tools when troubleshooting. If you are an experienced Linux administrator, you should be comfortable with most of these tools, and you should find that your existing Linux experience will serve you well as you troubleshoot OpenStack. In this section, we will walk you through some of the more common tools that are used. We will explain how each tool can be leveraged in an OpenStack environment specifically, but if you are interested in learning how the tools work generally, much can be learned by researching each tool on the Internet.

### Linux processes

OpenStack runs several processes that are critical to its smooth operation. Understanding each process can be very helpful to quickly identify and resolve problems in your cluster. It is not uncommon for the source of your problems to be rooted in the fact that a process has died or not started successfully. Bringing your cluster back to health may be as simple as restarting the necessary process. As we tackle each OpenStack project, we will introduce you to the key processes for that project's service. Like any Linux process, there are several commands that we can leverage to check these processes. Some of the common commands that we will leverage are detailed in the following sections.

**ps**

Hopefully, the `ps` command is already familiar to you as a Linux administrator. We leverage this command in OpenStack to get a snapshot of the current processes running on our host machines. The command will quickly allow us to see which OpenStack processes are running, and more importantly when troubleshooting, which OpenStack processes are not running.
We typically use the `ps` command with the standard `-aux` options and then pipe that to `grep` in order to find the OpenStack process we are interested in:

```
root@ost-controller:~$ ps -aux | grep nova
root  864  0.0  0.0 139180  8520 ?  Ss  11:35  0:00 /usr/sbin
       /nova-agent -q -p /var/nov/nova-agent.pid -o /var/log/nova-agent.log -l de
bug /var/share/nova-agent/nova-agent.py
nova 11988  0.5  2.5 261696 96204 ?  S  15:58  0:02 /usr/bin/
       python /usr/local/bin/nova-cert --config-file=/etc/nova/nova.conf
nova 11726  0.5  1.9 245042 74200 ?  S  15:58  0:01 /usr/bin/
       python /usr/local/bin/nova-conductor --config-file=/etc/nova/nova.conf
nova 11799  1.0  1.7 154460 66572 ?  Ss  15:58  0:03 /usr/bin/
       python /usr/local/bin/nova-consoleauth --config-file=/etc/nova/nova.conf
nova 11793  0.6  1.9 245252 74236 ?  Ss  15:58  0:02 /usr/bin/
       python /usr/local/bin/nova-scheduler --config-file=/etc/nova/nova.conf
nova 11843  1.8  1.9 249292 74936 ?  S  15:58  0:05 /usr/bin/
       python /usr/local/bin/nova-conductor --config-file=/etc/nova/nova.conf
nova 11844  1.8  1.9 249257 74930 ?  S  15:58  0:05 /usr/bin/
       python /usr/local/bin/nova-conductor --config-file=/etc/nova/nova.conf
nova 2030  3.0  2.9 145326 115760 ?  Ss  16:01  0:04 /usr/bin/
       python /usr/local/bin/nova-compute --config-file=/etc/nova/nova.conf --con
fig-file=/etc/nova/nova-compute.conf
root  28304  0.0  0.0 11904  912 pts/0  S  16:03  0:00 grep nova
```

For example, the preceding code would list each of the OpenStack Nova processes, which, by convention, are prefixed with `nova-`. It's also worth pointing out that this command may also reveal the `--log-file` option set when the process was launched. This will give you the location of the log files for each process, which will be extremely valuable during our troubleshooting.

**pgrep**

In addition to the `ps` command that is used to look at processes, you can also leverage the `pgrep` command. This command allows you to look up processes based on a pattern. For example, you can list processes based on their names:

```
root@ost-controller:~$ pgrep -l nova
664 nova-agent
11698 nova-cert
11726 nova-consoleauth
11759 nova-conductor
11793 nova-scheduler
11843 nova-conductor
11844 nova-conductor
28304 nova-compute
```
This command will list all the processes that have `nova` in their name. Without the `-l` option, the command would only list the process ID. If we want to see the process name too, we simply add `-l`. If you'd like to see the full-line output like we saw with `ps`, then you can add the `-a` option. With this option, you will be able to see extra attributes that are used when starting the process, including log file locations.

**pkill**
Along with the `pgrep` command, there is the `pkill` command. This command allows you to kill processes that match the name pattern that you provide. Take a look at the following as an example:

```
root@host-controller:~# pkill 20069
```

The preceding command would kill the process with PID 20069. This can be useful in situations where you have process hanging and you need to restart them. This is an alternative to the standard `kill` command.

**top and htop**
While `ps` and `pgrep` provide us with a snapshot of the running processes, `top` and `htop` will give us an interactive view of our processes. The `top` and `htop` commands are very similar, but `htop` provides you with a little added interface sugar, including the ability to scroll data. You may need to install `htop` on your servers if you decide to use it. Using either of these commands, you will be able to see the processes interactively sorted by things, such as percentage of CPU used by the process or percentage of memory. If you find your cluster in a situation where there is resource contention on the host, this tool can begin to give you an idea of which process to focus on first. The following screenshot is a sample output from `htop`:
Hard drives

It’s likely you will need to troubleshoot an issue that is related to hard drives when dealing with OpenStack. You can leverage standard Linux tools to interrogate the hard drive and assist you in troubleshooting.

**df**

There will be several moments in our OpenStack journey where we will be concerned about storage and the hard drives in our cluster that provide some of that storage. The `df` command will be leveraged to report on the disk space used by our filesystem. We can add the `-h` option to make the values human readable:

```
root@post-controller:~ # df -h
Filesystem      Size  Used  Avail Use% Mounted on
udev            1.9G  6.0K  1.9G   1% /dev
tmpfs           377M  36K  377M   1% /run
/dev/xvda1      505M  4.0G  436M  9% /
none            4.0K  0 4.0K   0% /sys/fs/cgroup
none            5.0M  0 5.0M   0% /run/lock
none            1.9G  0 1.9G   0% /run/shm
none            100M   0 100M  0% /run/user
```

The output from this command tells us which filesystems are currently mounted and provides usage information for each, such as the size of the filesystem, the amount used, and the amount available. The command also tells us the mount point for each filesystem.

**fdisk**

In addition to `df`, we will leverage `fdisk`. The `fdisk` command allows us to work with the disk partition tables. This may become necessary when troubleshooting OpenStack Block Storage or working with images in OpenStack. Take the following code as an example:

```
root@post-controller:~ # fdisk -l

Disk /dev/xvda: 53.7 GB, 5367091200 bytes
255 heads, 63 sectors/track, 6527 cylinders, total 104057600 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x80000a0a

   Device   Boot  Start  End   Blocks   Id  System
/dev/xvda1   * 2040  104056259  52427103+  83  Linux
```
The preceding command will list the partition table. From the list, you can see the
details about the disk, including its name and size. You can also see which partitions
correspond to the disk. In addition to listing the partition table, you can also modify
the partitions.

```
[root@ost-contoller:~]# fdisk /dev/xvda
```

This command will allow you to change the partition table for the disk named
`/dev/xvda`. After running this command, type `m` to see the menu of commands.
Using this menu, you can create new partitions, delete existing ones, or change
existing partitions.

**parted**

As we will discover later in this book, there are some use cases where you can’t use
`fdisk`. In those situations, we will look to another partitioning tool named `parted`.
This tool also allows us to work with partitions. With `parted`, we can create, resize,
copy, move, and delete partitions. The `parted` tool allows you to work with many
different types of filesystems as compared to `fdisk`.

```
[root@ost-contoller:~]# parted
GNU Parted 2.3
Using /dev/xvda
Welcome to GNU Parted! Type ‘help’ to view a list of commands.
(parted) *
```
The preceding command will start the parted tool. Once the tool starts, you can type `help` in the prompt to see a list of menu items. Some of the functionalities listed include `makefs`, to make file systems; and `makepart`, to make a partition; or `makepartfs`, to make both at the same time.

**cat /proc/partitions**

It's worth noting that we can also run the following command to list the partitions:

```bash
$ cat /proc/partitions
```

<table>
<thead>
<tr>
<th>major</th>
<th>minor</th>
<th>#blocks</th>
<th>device</th>
</tr>
</thead>
<tbody>
<tr>
<td>202</td>
<td>0</td>
<td>52428800</td>
<td>xvda</td>
</tr>
<tr>
<td>202</td>
<td>1</td>
<td>52427163</td>
<td>xvda</td>
</tr>
</tbody>
</table>

The `/proc/partitions` file is dynamic and made on the fly. Viewing this file will give you similar information as what you would find by running `fdisk -l`.

**Installed packages**

On Debian and Ubuntu systems, we can leverage the advanced package tool by running the `apt` command. This tool can be used to provide insight into which packages are installed on our system. This knowledge can be useful when troubleshooting OpenStack problems related to packages:

```bash
$ apt search openstack | less
```

The preceding command will list the packages that have the word `openstack` in their description and paginate the output. Running this command will give you a sense of some of the packages that come together to create an OpenStack cloud. Not all the packages listed are required, but this will give you an idea of which packages are available:

```bash
$ apt list | grep nova
```

The preceding `apt list` command will list the packages on the system. For example, we can pipe `apt list` to `grep` for `nova` to see a list of packages with `nova` in the name. We can take any of these packages and run them through `apt show` to get the details about the package. Take this line of command, for example:

```bash
$ apt show nova-api
```
General tools
We will make use of several Linux utilities throughout this book. Some of these tools are outlined in the following section.

The watch command
One of the many commands you may find useful when troubleshooting is the watch command. This command provides a convenient way to execute a command on a given time interval. I often use it to keep an eye on my processes when I'm trying to get them to restart. I've also leveraged this command when troubleshooting instance creation, as it allows me to check whether and when the instance becomes active:

watch pgrep -l nova

The preceding command will run the `pgrep -l nova` command every two seconds by default. You can adjust the interval at which the command is run by passing the `-n` option:

watch -n 3 nova list

This command will run the `nova list` command every 3 seconds.

File tools
We will leverage some commonly used Linux tools when troubleshooting. These tools include the following:

- **cat**: This is used to print files or input to the standard output.
- **less**: This is used to view files and allows you to page through those files.
- **find**: This allows you to search through files in a hierarchy of directories.
- **grep**: This allows you to search through files for lines that match a given pattern. We will use the `grep` command quite a bit as we are searching through logs for different types of messages.
- **tail**: This allows you to output the last part of a file. We will leverage the `tail` command often with the `-f` argument, which will allow us to follow a file as it is updated. This is used to watch logs live as we run different services or commands.
Message broker tools

One of the central components of any OpenStack cluster is the messaging broker. OpenStack uses a message broker to pass information back and forth between its components. The message queue facilitates intra-component communication, and as a result, it can often be a useful place to search for troubleshooting clues in an OpenStack cluster. While the default message broker installed with OpenStack is RabbitMQ, deployers have the ability to select from several other messaging technologies including ZeroMQ or QPid. We will explore some high-level troubleshooting tools for RabbitMQ in the following sections.

RabbitMQ

The RabbitMQ message system comes with a handy utility named rabbitmqctl. This tool allows operators to complete several useful tasks, but here we will highlight a few that are particularly helpful to troubleshoot OpenStack.

```
[root@host-controller:~]# rabbitmqctl status
Status of node 'rabbit@host-controller' ...
[(pid,22954),
 (running_applications,
 [rabbit,"RabbitMQ","3.2.4"],
 [os_mon,"CPQ" CXC 138 46","2.2.11"],
 [mnesia,"NHESIA CXC 156 12","4.11"],
 [zeml,"XML parser","1.3.5"],
 [sasl,"SASL CXC 136 11","2.3.11"],
 [stelib,"ERTS CXC 138 10","1.19.4"],
 [kernel,"ERTS CXC 138 10","2.16.4"]],
```

The preceding command will return the status of your RabbitMQ message broker. It can be helpful to check the output of this command for any errors. For example, if you see an error that starts with `Error: unable to connect to node`, then this means that RabbitMQ is likely not running. You can run the following command on Ubuntu to try and start it:

```
sudo service rabbitmq-server start
```

```
rabbitmqctl stop
```
The preceding command will stop your RabbitMQ message broker. You can use the same service command from the one we used here to restart it.

```bash
root@ost-controller:~# rabbitmqctl list_queues
Listing queues ...
cent 0
cent.ost-controller 0
cent_fanout_cc5d6a126b08+9b7953c+3ff7ed917 0
calculate 0
calculate.ost-controller 0
```

The `list_queues` command will list the queues in your message broker. When you run this on an OpenStack cluster, you will be able to see the decent number of queues used by the software to pass messages back and forth between OpenStack components. In addition to the name of the queues, this command can show you several attributes for each queue. Select the attributes you want to see by passing them after your `list_queues` command.

```bash
root@ost-controller:~# rabbitmqctl list_queues name durable messages_ready consumers
Listing queues ...
cent false 0 1
cent.ost-controller false 0 1
cent_fanout_cc5d6a126b08+9b7953c+3ff7ed917 false 0 1
calculate false 0 1
calculate.ost-controller false 0 1
```

In this command, we are requesting several columns of data directly after the `list_queues` command. Running the command this way will return a tab-delimited list including the name of the queue, whether or not the queue is durable, the number of messages ready to be read from the queue, the number of consumers listening to the queue, and the current queue state.

```bash
root@ost-controller:~# rabbitmqctl list_exchanges
Listing exchanges ...
direct
direct
calculate
calculate.headers
calculate.body
```

```bash
root@ost-controller:~# rabbitmqctl list_exchanges
Listing exchanges ...
direct
direct
calculate
calculate.headers
calculate.body
```

```
root@ost-controller:~# rabbitmqctl list_exchanges
Listing exchanges ...
direct
direct
calculate
calculate.headers
calculate.body
```

```
root@ost-controller:~# rabbitmqctl list_exchanges
Listing exchanges ...
direct
direct
calculate
calculate.headers
calculate.body
```

```
root@ost-controller:~# rabbitmqctl list_exchanges
Listing exchanges ...
direct
direct
calculate
calculate.headers
calculate.body
```
Like the list_queues command, there is also a list_exchanges command, which allows you to see the exchanges in your message broker. Running this command on an OpenStack cluster will allow you to see the exchanges that OpenStack leverages. Exchanges sit between message producers and the queues where those messages will eventually reside. The exchange is responsible for taking the message from the message producer and delivering that message to the appropriate queue if there is any queue at all:

```bash
rabbitmqctl list_exchanges name type durable policy
```

Running the list_exchanges command with the name, type, durable, and policy column headers, as demonstrated in the preceding line of code, will output the exchanges and their respective values for each column. Specifically, the name of the exchange; the exchange type which is either direct, topic, headers, or fanout; whether or not the exchange is durable, meaning it will survive a server restart; and finally, the exchange's policy. Policies are a method by which administrators can control the behavior of queues and exchanges across the entire RabbitMQ cluster. You can see which policies, if any, are configured by running this:

```bash
rabbitmqctl list_policies
```

In RabbitMQ, exchanges are related to queues through bindings. Queues use bindings to tell exchanges that they are interested in messages flowing through that exchange. You can list the bindings by running this code:

```bash
root@ost-controller:~ $ rabbitmqctl list_bindings
Listing bindings ...
    exchange cert_queue cert []
    exchange cert.ost-controller_queue cert.ost-controller_queue
    exchange cert_fanout_ac5d6a128b0849b79153c43f7f7ed17_queue
cue cert_fanout_ac5d6a128b0849b79153c43f7f7ed17_qu
    exchange compute_queue compute []
    exchange compute.ost-controller_queue compute.ost-contro
    ller []
```

The list_bindings command will display each of the bindings between exchanges and queues. By default, it will list the name of the source, the type of the source, the name of the destination, the type of the destination, a routing key, and any binding arguments.

To view a list of RabbitMQ clients with connections to the RabbitMQ server, run the list_connections command:

```bash
rabbitmqctl list_connections
```
This command will list the username associated with the connection, the hostname of the connected peer, the port of the peer, and the state of the connection.

**Summary**

In this chapter, we explored the OpenStack projects at a high level. You learned a little about each of the core projects and some of the optional projects that are deployed. We looked at the main supporting technologies that OpenStack leverages to provide things like, data persistence, and messaging. Finally, we introduced the troubleshooting methodology that we will use throughout this book and a few of the troubleshooting tools that we will take advantage of while working with OpenStack. In the next chapter, we will dive into troubleshooting OpenStack Keystone, the identity service.
Where to buy this book
You can buy Troubleshooting OpenStack from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.
Click here for ordering and shipping details.