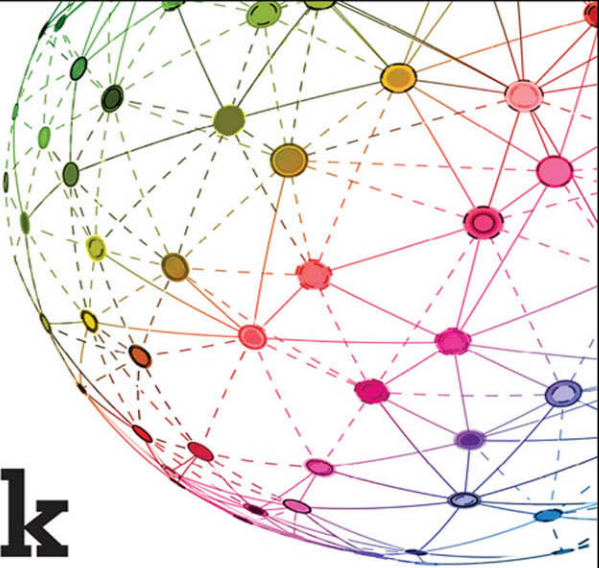


Common **OpenStack** Deployments

Real-World Examples for Systems Administrators and Engineers

Elizabeth K. Joseph

with **Matt Fischer**

Common OpenStack Deployments

This page intentionally left blank

Common OpenStack Deployments

Real-World Examples for Systems Administrators and Engineers

Elizabeth K. Joseph
with Matthew Fischer

Boston • Columbus • Indianapolis • New York • San Francisco • Amsterdam • Cape Town
Dubai • London • Madrid • Milan • Munich • Paris • Montreal • Toronto • Delhi • Mexico City
São Paulo • Sydney • Hong Kong • Seoul • Singapore • Taipei • Tokyo

Library of Congress Control Number: 2016944433

ISBN-13: 978-0-13-408623-1
ISBN-10: 0-13-408623-6
Text printed in the United States on recycled paper at RR Donnelley in Crawfordsville, Indiana.

1 16

❖

This book is dedicated to the OpenStack community. Of the community, I'd also like to specifically call out the help and support received from the Puppet OpenStack Team, whose work directly laid the foundation for the deployment scenarios in this book.

❖

This page intentionally left blank

Contents

Preface

And suddenly you know: It's time to start something new and trust the magic of beginnings.
Meister Eckhart

Companies today are heavily relying upon virtualized and cloud-based solutions in their infrastructures. Whether they are off-loading their work to third-party cloud providers, using a virtualized solution in-house or building clouds in their own data centers, OpenStack has a lot to offer. This book provides an introduction to using and deploying OpenStack, open source software for creating private and public clouds, in your own organization.

OpenStack as an open source project has only existed since 2010 but quickly gained support of companies around the world and the broader open source community. At open source conferences OpenStack talks quickly sprang up by the time the project was just three years old to the extent that some in the community joked that the "OS" in open source conference names no longer stood for "Open Source" but for "OpenStack." Demand for talent in the field has risen along with interest, with OpenStack experts demanding a premium as companies expand their private cloud deployments.

Audience

The audience for this book is Linux and Unix systems administrators and network engineers seeking to learn the basics of OpenStack and to run sample deployment scenarios that can be transitioned into real-world deployments. It also provides insight into the most popular ways OpenStack is being used and how your organization can get there.

Though detailed commands are given, literacy with Linux systems administration is expected so you can focus on learning about OpenStack and simplify the task of troubleshooting. If you are doing these deployments in a series of virtual machines rather than bare metal servers, familiarity with a virtual machine technology is expected. This book does provide a reference deployment using virtualization with KVM and QEMU on Ubuntu if you wish to have step-by-step instructions. However, the intent is to leave virtualization preference up to the reader and make it easier to transition to a physical setup.

Basic networking experience is recommended since networking is such an important part of OpenStack, but diagrams in Chapter 3, "Networking," will help guide you through the OpenStack network architecture we're demonstrating.

Goals and Vision

I was inspired to write this book after attending several open source conferences where OpenStack was becoming an increasingly popular topic. In spite of all these talks, I'd still get practical usage questions from friends and colleagues about how they could use Open-Stack in their organizations. To that end, each chapter with a deployment scenario begins with a series of real-world examples of how it is being used by organizations in production. From serving up fleets of web servers to log storage, backups and data processing, these usage examples help you to find a place for OpenStack to accomplish a variety of tasks that span organizations across various industries, universities and governments.

Whether you're seeking to do an OpenStack deployment yourself or work with a vendor, this book also provides a guide through these sample deployments. You learn the basics of how to configure various OpenStack components and then walk through interactions with them via a web dashboard and the OpenStack command-line client. The mechanics of how the components interact with each other are also explained, so you have an understanding of how you're interacting with the systems.

System Prerequisites

In order to run most of the deployment scenarios in this book you need, at minimum, two computers with combined resources of 6G of RAM and 50G of hard drive space. A laptop with 8G of RAM is sufficient if you're using virtualization, including use of our tested reference deployment described in Appendix A, "Reference Deployment." If you choose to use real hardware, you will need two computers and two switches. One of the switches must be connected to a network with access to the Internet so you can install system packages and pull in configuration management tooling on your systems.

Diagrams of both virtualized and physical environment options, including a breakdown of specifications for each, are included in Chapter 3.

Ubuntu

You need to download the latest Ubuntu 14.04 ISO image to complete the deployment scenarios. Ubuntu is a Linux distribution based on Debian, which had its first release in 2004. Initially aimed at making Linux easier for regular people, growth of Ubuntu on servers has exploded over the past five years. It's now the number one choice for cloud deployments, both in OpenStack and on other cloud platforms such as Amazon Elastic Compute Cloud (EC2).

OpenStack's beginnings are also intertwined with the Ubuntu community. A number of the early project contributors come from the Ubuntu project. The decision to use Ubuntu comes from the expertise of the authors and a desire to focus on understanding the types of deployments and basics surrounding OpenStack itself rather than the underlying operating system.

Although the OpenStack ecosystem is much broader than Ubuntu, with professional services built around Red Hat Enterprise Linux (RHEL) and even a move to other operating systems beyond Linux, this book uses Ubuntu 14.04 LTS (Long Term Support) as the base installation for OpenStack.

This will impact configuration to some extent, since the Ubuntu Cloud Archive (see Chapter 1, "What Is OpenStack?") referenced in this book will be different from that of RHEL, CentOS openSUSE, Fedora and others. However, the core OpenStack knowledge and the deployment examples will be exportable to other systems once you start making plans to move into production with your system of choice.

Puppet

With an initial release in 2005, today Puppet is one of the most popular configuration management systems in the world. The Puppet OpenStack modules were one of the first configuration management system projects to reach maturity in the OpenStack community. Puppet modules for each release are made available within weeks of the OpenStack release itself.

Like the selection of Ubuntu, the selection of Puppet for configuration management was made so we can focus less on fundamental deployment and management, and more on learning about OpenStack. While you will be using Puppet commands for these deployments, the basic concepts are explained and prior knowledge of Puppet is not required. Additionally, the creators of the Puppet modules for OpenStack are a diverse group of developers and operators from around the world and are formally supported by multiple organizations. They are known to be flexible enough for a variety of environments.

We will be using the default installed Puppet version on Ubuntu 14.04. If you're seeking to run Puppet in production, the OpenStack Puppet team recommends using the Puppet version directly from Puppet instead. This is covered in Appendix C, "Long-Lived Puppet." The OpenStack Puppet modules for OpenStack are currently tested on both Ubuntu and CentOS.

Appendix B, "Other Deployment Mechanisms," has been provided to give you an overview of other configuration management and orchestration services you may be interested in using, should your organization prefer Chef, Ansible or something else.

Tour

The following is a short tour of what to expect from each chapter and the appendices.

- **Chapter 1: What Is OpenStack?** This first chapter provides a brief introduction to cloud computing before moving into an introduction to OpenStack itself. It goes on to provide descriptions of each component of OpenStack that are explored in depth in later chapters. The chapter concludes by talking about the OpenStack release cycle and how Ubuntu and Puppet factor into this cycle and their usage in this book.

- **Chapter 2: DevStack** Built as a non-production development tool, DevStack is also a great introductory tool for a single-server deployment of OpenStack and for getting familiar with it quickly. You learn how to use it, launch your first compute instance and execute basic debugging techniques.
- **Chapter 3: Networking** Networking is an important and complicated component of OpenStack and will drive decisions you make as you build your own deployments. This chapter is devoted to explaining key concepts for networking in OpenStack and to dive into the networking decisions and requirements used in our deployment scenarios. Diagrams and written descriptions help guide you through these concepts.
- **Chapter 4: Your First OpenStack** Before getting into chapters using configuration management, this chapter walks you through a manual install of the basic components of OpenStack, Nova compute, Keystone Identity, Glance image storage and Neutron networking. This will give you a firm understanding of how the pieces fit together, from the databases to the service users in Keystone, which are handled in later chapters automatically by configuration management to the queuing system.
- **Chapter 5: Foundations for Deployments** This chapter serves as a basis for all your subsequent deployment scenarios using Puppet. It explains the core components and sets up your basic controller and compute node and concludes with some basic usage tests to confirm it is working.
- **Chapter 6: Private Compute Cloud** The first of our Puppet-driven deployment scenarios, this chapter provides usage examples and then walks you through the basics of interacting with a private compute cloud. You learn how to add a compute flavor and your first operating system image, how to launch and interact with a Nova compute instance from both the Horizon dashboard and the command line client and then complete a basic web service demonstration.
- **Chapter 7: Public Compute Cloud** Your next deployment scenario adds metering to your cloud with Ceilometer. Ceilometer tracks usage of RAM, CPU, networking and more for your deployments, which you can then feed into systems to do monitoring and billing. Usage examples are given, as well as a basic introduction to Ceilometer itself and a walkthrough of how to use it with a strong focus on the command-line client.
- **Chapter 8: Block Storage Cloud** Moving on from compute-focused deployments, this chapter introduces the concept of block storage and provides example usage. The basics of OpenStack Cinder block storage architecture are explained, and then you are walked through configuration. You then attach a Cinder block storage device to a compute instance, partition it, give it a filesystem and mount it inside your compute instance so you can add files to it.
- **Chapter 9: Object Storage Cloud** Continuing with storage, this chapter introduces you to the concept of object storage using Swift. You learn about basic Swift concepts and deployment considerations, and then build your own tiny Swift

deployment. Using this deployment scenario, you create storage containers, upload files and build upon your earlier web service demonstration by including an image served by object storage on a compute instance.

- **Chapter 10: Bare Metal Provisioning** Moving on from our deployment scenarios, usage examples and an architecture overview of bare metal provisioning with OpenStack Ironic are provided. Though you aren't doing an actual deployment for this chapter since we couldn't make assumptions about your hardware, guidance is given for how you might.
- **Chapter 11: Controlling Containers** In this, another non-deployment chapter, you learn why you may wish to use containers in an OpenStack deployment. The chapter continues with a basic introduction to OpenStack Magnum and considerations for your own deployments.
- **Chapter 12: A Whole Cloud** Coming back to our deployment scenarios, this chapter provides one final scenario where all the components from Chapters 6-9 are brought together in a single scenario. This demonstrates how they can be used together, and you're encouraged to do your own tests with this feature-rich cloud scenario.
- **Chapter 13: Troubleshooting** OpenStack is a complicated infrastructure project, and every engineer running it needs to get very good at troubleshooting. You are walked through understanding error messages and log files, tooling for troubleshooting network problems, common mistakes in configuration files and basic Puppet debugging. The chapter concludes with tips for how you can mitigate breakage in your deployment and tips for asking for help.
- **Chapter 14: Vendors and Hybrid Clouds** The final chapter of this book introduces you to the broader OpenStack ecosystem through vendors and hybrid clouds, which blend a local OpenStack deployment with hosted solutions. Evaluation considerations for choices you make from cost to data sovereignty and security are covered.
- **Appendix A: Reference Deployment** In case you run into trouble with your own environment selections, or simply don't have a preference, this appendix provides a tested, virtualized reference deployment you may use.
- **Appendix B: Other Deployment Mechanisms** We use Puppet throughout this book, but this appendix introduces you to other ways you can deploy OpenStack, from Chef and Ansible to where to find vendor-specific tooling.
- **Appendix C: Long-Lived Puppet** The Puppet examples in this book are triggered manually. This appendix gives direction for your options when building a proper, maintainable Puppet system.
- **Appendix D: Contributing Code to OpenStack** Feel inspired to contribute back to OpenStack? Or need a feature or bug fix? This appendix gives an introduction to how you go about contributing code to the OpenStack open source project, including how community members communicate and how to use the development tooling.

- **Appendix E: OpenStack Client (OSC)** The OpenStack Client is rapidly replacing individually maintained clients for each project. This appendix provides some background and a quick reference of some common commands.
- **Appendix F: Finding Help with OpenStack** The final appendix provides a quick tour of the support options in the OpenStack community, both online and in person. It concludes with tips for finding paid support as well.

Conventions

Instead of using the root user, sudo is used throughout this book. As such, all commands are prefixed with a dollar sign to indicate that it's a command you should be typing into a shell. For instance, when you're preparing your Ubuntu systems and want to update the Ubuntu sources before installing anything, we show that as:

```
$ sudo apt-get update
```

When lines are wrapping, we use the bash syntax of \ to indicate that the command wraps to the next line. The creation of a compute instance is a good example of this:

```
$ openstack server create --flavor m1.tiny --image "CirrOS 0.3.4" \
  --security-group default --nic net-id=Network1 \
  --availability-zone nova my_first_instance
```

For most of the OpenStack commands, we have provided sample output of what to expect when you run each command. For the output of standard tooling for things like Ubuntu package installation, git clones and MySQL commands, this output is generally not included.

Supplementary Materials

As discussed, you will need a copy of the Ubuntu 14.04 ISO to install Ubuntu on your initial OpenStack controller and compute nodes. Later, the Ubuntu 14.04 server QCOW2 cloud image will need to be loaded into Glance for our deployment example using Ubuntu as a compute instance. All Puppet modules and other packages are downloaded through scripts you're instructed to use or through Puppet itself.

This book also has an accompanying git project hosted on GitHub at https://github.com/DeploymentsBook.

This project is broken into several repositories:

- **http-files**—Used for our basic web server examples.
- **puppet-data**—The repository you clone to bootstrap your installation of Puppet on your OpenStack nodes. It also includes your core configuration file, hiera/common.yaml, which you will be editing.

- **puppet-deployments**—Pulled in automatically by setup.py in puppet-data, this is the composition module used for all of our deployment scenarios. It includes service profiles and the foundation roles used in each chapter. It also includes a README.md file for the latest known issues and work-arounds that will be updated throughout the life of this book.
- **scripts-and-configs**—Miscellaneous scripts, commands and configuration file examples provided so you have a place from which to view or copy them as needed. The commands provided in this directory for the deployment chapters are particularly valuable for viewing OpenStack client output that doesn't fit well on a printed page.

Finally, a blog and the latest updates to other materials being made available throughout the lifespan of this book can be found on our web site at http://deploymentsbook.com/. You can also follow us on Twitter for updates @DeploymentsBook.

Acknowledgments

When I began working on this book, I knew I had my work cut out for me and that I would need help from various members of the community. OpenStack is a huge infrastructure project. Every aspect of the project is continually being refined and revised, and even the official project documentation struggles to keep up. New projects are always being added, and the existing ones are reaching various states of maturity.

A few months into writing I brought in my contributing author, Matt Fischer. He put in a massive amount of work across three releases of OpenStack to get our Puppet composition module working. This book wouldn't have made it past the theoretical stage without his efforts. Colleen Murphy of the OpenStack Puppet team also spent time working with us on changes and by doing review of chapters and appendices. Clayton O'Neill, Eric Peterson and Adam Vinsh, working with Matt directly, assisted with Puppet configuration questions. We also had the project team lead of the OpenStack Puppet project, Emilien Macchi, pitch in with advice and getting required changes needed upstream.

Thanks to members of various teams who took time as subject matter experts to review individual chapters, including: Mike Perez, Gordon Chung, Donagh McCabe, Matthew Oliver, Hisashi Osanai, Christian Schwede, Kota Tsuyuzaki, Julia Kreger and Charlie Crawford. Pasi Lallinaho helped by turning our basic HTML page examples into something considerably more palatable. We also had help from my friends and fellow systems and network engineers Jonathan DeMasi, Ola Peters, Joe Gordon, Eric Windisch, James Downs and Brent Saner on several of the chapters.

We had several multi-chapter reviewers with varying backgrounds to thank, including José Antonio Rey, Mohammed Arafa, Doug Hellman and Christian Berendt.

Throughout this process, my husband Mike Joseph has been incredibly supportive of my work. Even during the most difficult times when I wasn't sure I could finish it, he was ready with encouragement.

Finally, I'd like to thank my editors at Pearson. Thanks to my editor and primary contact at Pearson, Debra Williams Cauley, for offering advice on how to approach each section and for keeping me on track throughout this process. Also to editor Chris Zahn, who made his way through editing the entire book.

About the Author

Elizabeth K. Joseph is a Systems Administrator working on the OpenStack Infrastructure project. In her role on this team, she supports OpenStack developers as they make contributions to the project and is active on OpenStack development mailing lists, and has worked on test engineering for the OpenStack on OpenStack (TripleO) project. She has given tutorials on the basics of OpenStack for San Francisco Bay Area organizations and regularly attends the bi-annual OpenStack Design Summits. A regular speaker on Open Source topics at conferences worldwide, her work beyond OpenStack includes contributions to the Ubuntu project and serving on the board of a non-profit that puts Linux-based computers in public schools.

Matthew Fischer has worked as a software developer for over 15 years in roles ranging from UNIX kernel to mobile phone development to DevOps. Matt currently works on a team deploying and running OpenStack and has been using Puppet to deploy OpenStack since 2013. When not solving automation problems, Matt enjoys hiking, camping, skiing, craft beer, and spending time with his family in Fort Collins, Colorado.

This page intentionally left blank

II

Deployments

In the following chapters, you are given examples of several types of OpenStack deployments to consider. Each deployment is based on a series of real-world scenarios that will be described in detail and followed up with instructions for how the deployment can be constructed in a small scale.

8
Block Storage Cloud

The road to the City of Emeralds is paved with yellow brick.
L. Frank Baum, *The Wonderful Wizard of Oz*

OpenStack provides two popular mechanisms for storage: object and block storage. Block storage is traditionally what you'd mount as a filesystem on your server. Object storage instead hosts individual files that are then referenced from within your application. In Chapter 9, "Object Storage Cloud," you learn about why you may want to use object storage with Swift to host files. This chapter covers block storage with Cinder.

Integrated with the rest of OpenStack, Cinder volumes can be created within an OpenStack cloud and then live mounted to a specified instance at the whim of a user. Additionally, you can unmount that volume from one instance and mount it on another with just a few commands.

Uses

One of the strengths of OpenStack is to avoid vendor lock-in, particularly when used in combination with versatile solutions like Cinder. Cinder provides an abstraction layer through the volume manager that hooks in to over 70 different proprietary and open source storage solutions. Additionally, it can be an interface to multiple back ends at once, enabling you to not only diversify your back ends across vendors but also change them out and do a planned migration as your organization sees fit.

Cloud Provider

Whether you're running a public cloud accessible to customers or a private cloud for use within your organization, offering the capability to extend the given filesystem requirements in place for the flavors can be a huge benefit.

With the capability to add block storage volumes on the fly, the storage requirements for default compute nodes can be kept small to preserve space for users who want to focus on compute power and give others the flexibility to add the storage they need on the fly. The flexibility to extend storage as needed helps scaling out resources without over-committing beforehand and makes migrations easier if data is kept on a single volume that can be moved to a new compute instance.

Another major benefit is that the compute nodes can be run on throw-away commodity hardware. The data needing to be stored can be either kept on expensive redundant enterprise hardware or on something that has built-in redundancy like Swift. As a result, compute nodes themselves could become independent, throw-away components of your infrastructure, spun up as needed and replaced with identical servers that attach to your storage back ends. Finally, upgrades are also simplified for your customers. If you have a new copy of your system you are testing, you can snapshot the production data from a Cinder volume and then attach it to your test system to see if it works.

Pets versus Cattle

If you're unfamiliar with the pets versus cattle metaphor in cloud computing, it's time to get you up to speed.

Prior to the recent rise in cloud computing, systems administrators would work with our managers to spec out hardware for servers and work out a budget to purchase it. A new server would be delivered to the facility where we'd install the operating system and put it into a rack in our data center. A nice label would go on the server designating a name, and over time we'd work to maintain and update this server. The server would be upgraded for years through operating systems and hardware failures and upgrades. We'd start noticing specific quirks about the hardware: one server may have a flaky onboard NIC so our notes explained that we added an additional card, another machine may take a while to boot up. We became familiar with these servers we grew and nurtured over time, like pets.

When workloads and businesses began moving to the cloud for their workloads, everything changed. Individual servers no longer had quirks and were easy to move around. A complete replacement of a server became more common than an in-place upgrade, and tools were written to manage fleets of servers operating in a larger infrastructure rather than working with individual servers. Instead of pets, servers became a lot more like cattle.

As a systems engineer and an animal lover, I retain a love for both pets and cattle, metaphorically or not. But this metaphor does effectively demonstrate the differences in how servers are treated today in a cloud environment.

Data Processing

Whether you're a film production company or a research institution, using compute nodes to do data processing and analysis is a common use for cloud-based infrastructures. But where is this data stored and how is it shared in your organization? What do you do when the compute nodes you're working on run out of space? With Cinder block storage, users are able to extend volumes after creation or simply create new volumes as they are needed and attach them to their running instances and mount them in a matter of minutes. Running out of hard drive space on an instance because you have too much data is no longer a problem.

Additionally, if you realize you need more processors or memory, it's simple to create a new compute instance and move the volume over to the new instance. All your data moves with you to your new server.

Keeping Backups

Making sure your data is backed up and replicated is a common concern, and Cinder offers several options for this. As mentioned earlier, it provides an abstraction layer for dozens of back ends, so you have a lot of options in your environment and on top of that, the different types of backups that Cinder offers.

Many virtualization technologies supported by OpenStack have built-in snapshot capabilities, but OpenStack's block storage also has one. This enables volumes to be backed up as snapshots to other block storage volumes and can be used on its own, using the existing configuration. Whether offered as a backup service by a cloud provider or as an automatic service for users on an OpenStack cloud, this is a valuable service.

You can also clone volumes. Many back ends are smart enough to do a copy-on-write in where zero copying actually happens. The new volume references an existing volume it was cloned from and writes data on top of that. This makes it very fast. Some snapshot features also just do this in the back end, but every vendor solution is implemented differently.

Moving beyond snapshots, Cinder also offers a backup service that enables you to back up your block data to an object store. This helps with the scenario of your entire block storage back end going completely off-line. Differential and incremental backups can be performed and, unlike a snapshot, it is only backing up data that was actually used, not the entire volume and unused bits.

We won't spend a lot of time on backup scenarios in this chapter. Instead we'll be focusing on adding volumes in a more instance-focused environment where you're adding volumes to instances, but they are great options to keep in mind.

Requirements

In this deployment scenario you'll once again be using the controller and compute nodes you created in Chapter 5, "Foundations for Deployments." We will be creating a 10GB volume group for use by Cinder on the controller and the minimum specifications defined in that chapter will easily support this.

Select Components

In addition to the foundational tooling, we'll be adding Cinder block storage to your deployment in this scenario. This will extend our very basic installation to have the following services. Their placement is demonstrated in Figure 8.1.

- Compute (Nova)
- Identity (Keystone)
- Networking (Neutron)
- Image service (Glance)
- Dashboard (Horizon)
- Block Storage (Cinder)

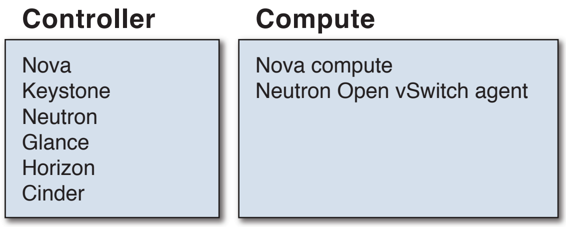


Figure 8.1 Components of a two-system OpenStack deployment with Cinder block storage

Architecture Overview

In Chapter 1, "What is OpenStack?" you were briefly introduced to the components that make up the block storage (Cinder) service: cinder-api, cinder-scheduler, cinder-volume and also cinder-backup. A user will likely only be exposed to the API. As operators though, understanding the architecture for the service is important as we seek to make decisions about how we build the system and debug problems.

When a user request comes in, either from the OpenStack dashboard (Horizon), the OpenStack Client (OSC) or through a Software Developer Kit (SDK), it interfaces with the API for Cinder. This API will talk to a database, for initially storing the request, and then set the status to creating and reserving quota usage. The API will also interact with a messaging queue. The messaging queue will pass requests on to the scheduler for Cinder, which makes decisions about where the change will be made. For example, if a user is requesting that a volume be created, the scheduler will determine which storage device meets the criteria the user is asking for, for the volume (size, disk type), and then send it to the appropriate volume manager. The volume manager for Cinder is what works directly with drivers to interface with the storage back ends. A storage back end may be a datacenter full of Ceph nodes or a proprietary Network-attached Storage (NAS) device that has a driver for Cinder. The volume manager will also be talking to the database to commit to the reserved quota once we know the volume is created successfully. The status of the volume is also set to "available" so the user knows the volume may be used. See Figure 8.2 for a view into how the individual services work together.

All official drivers that are available for Cinder go through verified testing by the upstream Cinder team in the OpenStack project. To accomplish this, every vendor is required to run continuous integration (CI) tests on all changes that report to the public OpenStack review system. To learn about the latest supported drivers, you can visit the OpenStack Marketplace Drivers page for an official listing: https://www.openstack.org/marketplace/drivers/.

Paying attention to the drivers and learning what is supported will be essential when you build out your production OpenStack deployment. When considering a solution, be sure to research the support for your storage back end of choice and look into factors like how long a solution or vendor has been supported in OpenStack and what they support when it comes to interacting with the Cinder volume manager.

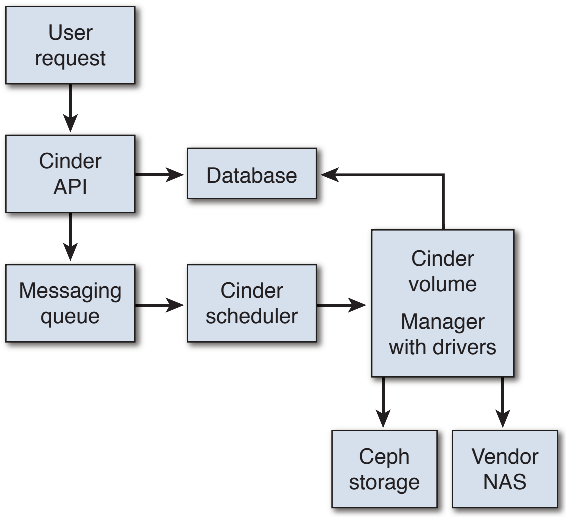


Figure 8.2 Cinder overview

Scenario

Extending beyond compute-focused deployments, like we saw in Chapter 6, "Private Compute Cloud," and Chapter 7, "Public Compute Cloud," with Cinder block storage means you can now offer real persistent storage to your users. Since we're adding an additional component, a bit of setup needs to be done with Puppet again to configure this storage.

Controller Node Setup

If you have your Controller and Compute nodes available from Chapter 5 you will only need to run a single command to add the support for Cinder block storage. In this scenario you will only need to make changes to the controller node. No modifications need to be made to the compute node.

> **Tip**
>
> Did you go through Chapter 7 before this chapter? You should create a new environment. Even though OpenStack is modular, we haven't designed our foundations modules to function together until you get to Chapter 12, "A Whole Cloud."

The command is another Puppet `apply` command, which will process our foundational block storage role in Puppet:

```
$ sudo puppet apply /etc/puppet/modules/deployments/manifests/role/foundations_block_storage.pp
```

This will take some time as it downloads everything required for Cinder and sets up configurations. If anything goes wrong and this command fails, remember that Puppet can also be run with the `--debug` flag in order to show more detail.

While this is running, we can take a look at what this file contains:

```
class deployments::role::foundations_block_storage {
  include deployments::role::foundations
  include deployments::profile::cinder
}

include deployments::role::foundations_block_storage
```

This is calling out to our foundations role, which means if you didn't set up a foundations role yet for your controller, it will do it now. This is mostly a safety measure; we would still recommend that you run it independently in case you need to do any troubleshooting.

It then calls our Cinder block storage profile, which you can view on the controller filesystem at /etc/puppet/modules/deployments/manifests/profile/cinder.pp, and it contains the following:

```
class deployments::profile::cinder
{
  include ::cinder
  include ::cinder::api
  include ::cinder::ceilometer
  include ::cinder::config
  include ::cinder::db::mysql
  include ::cinder::keystone::auth
  include ::cinder::scheduler
  include ::cinder::volume
  include ::cinder::setup_test_volume

  file { '/etc/init/cinder-loopback.conf':
    owner  => 'root',
    group  => 'root',
    mode   => '0644',
    content => template('deployments/cinder-loopback.conf.erb'),
  }
```

The profile pulls in various components to Cinder that we will need. Just like other services in OpenStack, Cinder requires an API, database and Keystone authentication. In case you wish to track usage with Ceilometer's telemetry service, we also include that. The config is pulled in to help manage arbitrary Cinder configurations you may wish to have. A scheduler in block storage is used in much the same way other OpenStack services use schedulers, to view the requirements the user is requesting for the volume, and then randomly pick a storage device back end that the volume can be created on that meets that criteria. As you may expect, pulling in `cinder::volume` is for the Cinder volume manager. As explained earlier in the chapter, this is what interacts with the drivers actually controlling the storage back end, whether it's a simple loopback device with LVM (Linux Volume Manager) like we will be using or a proprietary NAS device.

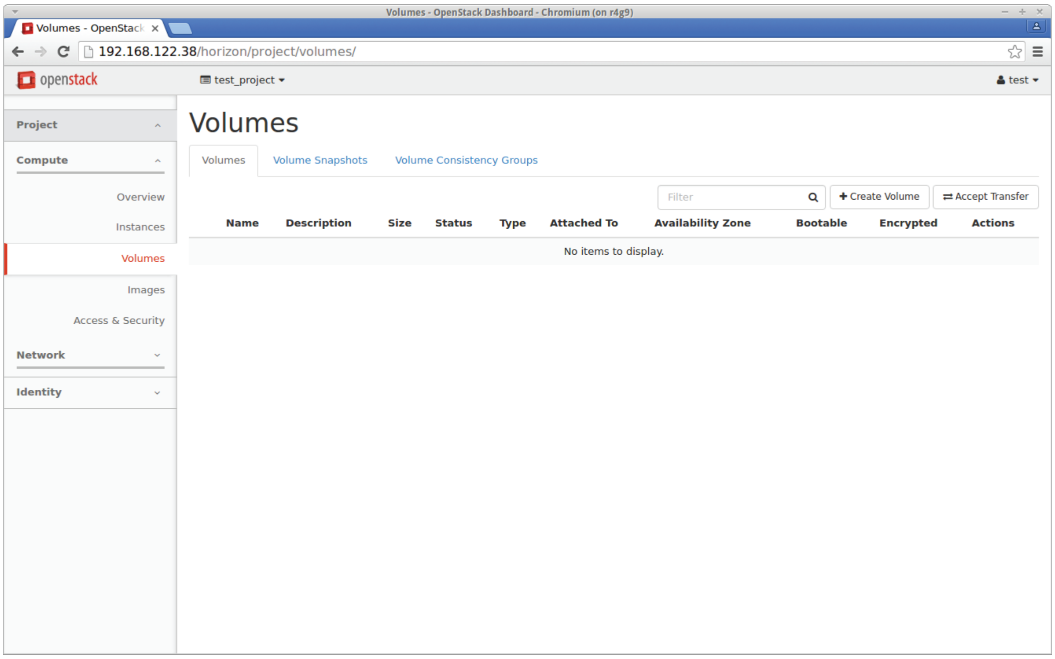
The final lines of this file use the Puppet module's capability to configure a test volume. For simplicity's sake we use this setup_test_volume, which creates a simple 10GB file mounted to a loopback (by default, /dev/loop2) device and added to LVM as a single logical group. An init file is also created in our cinder.pp profile to make sure the file is mounted and the volume group is activated if your controller reboots.

> **Note**
>
> What Is LVM? The official page for LVM is at: http://www.sourceware.org/lvm2/ and there are various resources and how-tos available for free online, particularly for basic control.

Once your `puppet apply` command completes, you're ready to start creating volumes and attaching them to instances!

Creating and Attaching a Volume: Dashboard

We will begin with the process for creating and attaching a volume using the OpenStack dashboard (Horizon). With the block storage (Cinder) component now installed, when you log into the dashboard with your test user you will see a section for Volumes in the left under Project in Compute, as show in Figure 8.3.



Name	Description	Size	Status	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions

No items to display.

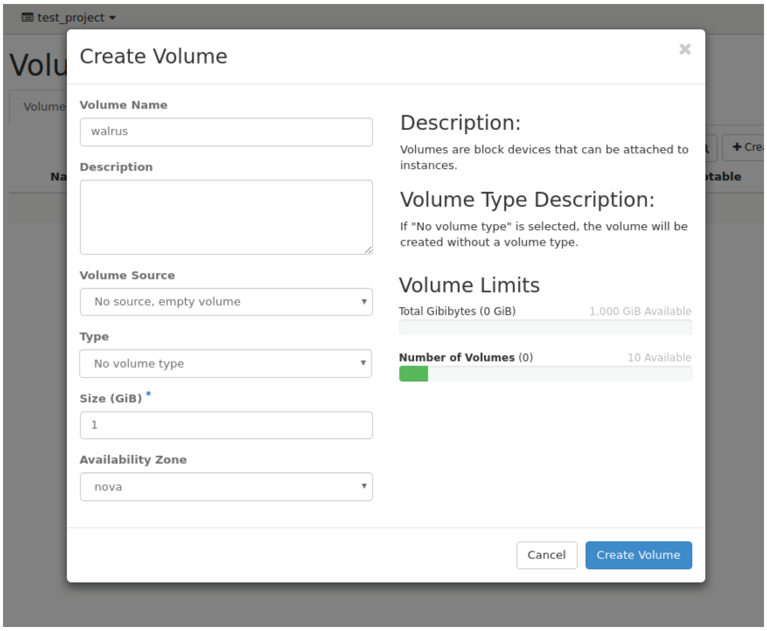
Figure 8.3 Empty, default Volumes page in the dashboard

Creating a Volume

On this page you'll want to click on the Create Volume button, which will bring up a dialog like the one in Figure 8.4 where you will put in information about the volume you wish to create. Some fields will be automatically filled out, but the rest will be up to you.

The volume name is what you will be using to refer to the volume. A description is optional and can be used for whatever you want, maybe as a reminder to yourself about what the volume is intended for. The volume source enables you to pre-populate the volume with a source of defined data. By default, it queries the Image Storage (Glance) service and enables you, as one of the options, to put an Image on your newly created volume. You may also want to create a volume source that has a basic filesystem and partition table for your new volume so it doesn't need to be created later after you mount it on an instance. For this scenario, we will just use No source, empty volume and will explain how to partition and format it after it is added to an instance.

The type of volume will inform the scheduler as to which type of storage back end you need to use. From the customer point of view, you want to define a type as tiered and varied storage with different properties, like how fast the storage device is, Quality of Service (QoS) requirements or whether a tier has replication. Prices may vary for the customer based on which options they select. From your perspective, this means one of these tiers may be using Ceph and another a proprietary NAS device that has the desired qualities for the tier being offered. We have not set a volume type, so it will remain as "No volume type" for this example. Our device only has 10GB, so we'll start out in



Volume Name: walrus

Description:

Volume Source: No source, empty volume

Type: No volume type

Size (GiB) *: 1

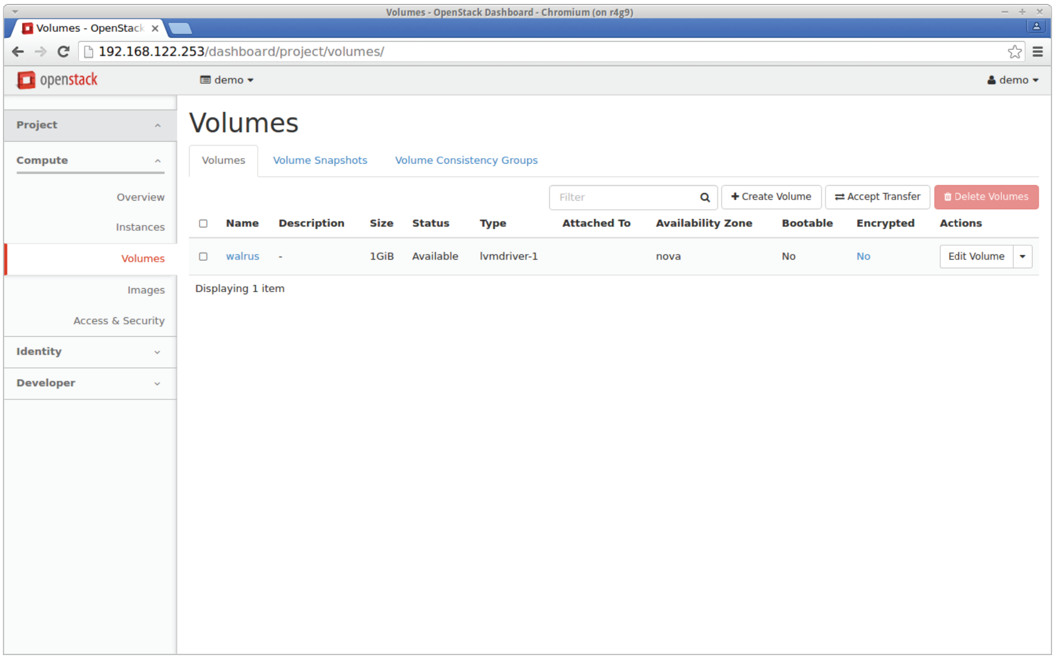
Availability Zone: nova

Description:
Volumes are block devices that can be attached to instances.

Volume Type Description:
If "No volume type" is selected, the volume will be created without a volume type.

Volume Limits
Total Gibibytes (0 GiB) 1,000 GiB Available
Number of Volumes (0) 10 Available

Figure 8.4 Create a volume in the dashboard.



Name	Description	Size	Status	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions
walrus	-	1GiB	Available	lvmdriver-1		nova	No	No	Edit Volume

Displaying 1 item

Figure 8.5 A volume called "walrus" has been created.

this test by creating a 1GB volume to attach to our instance. The availability zone is identical to the one in compute (Nova) and currently must match the zone where the instance you wish to attach it to resides. In our deployment scenario we only have a single availability zone, so the default of nova should remain selected.

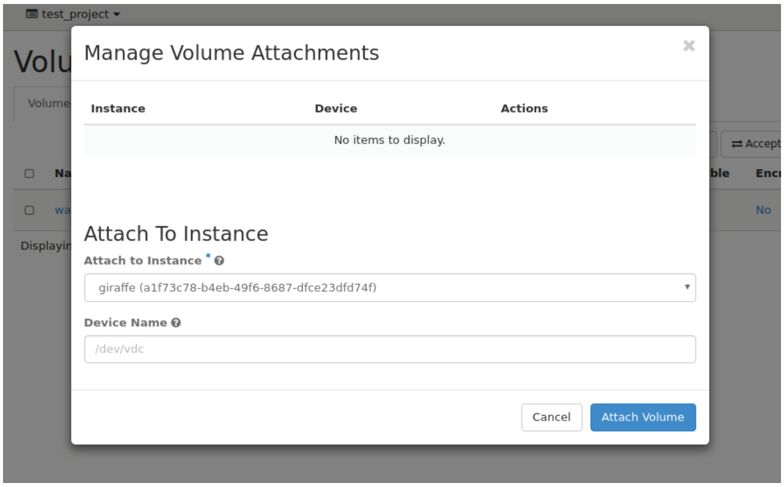
When you have finished, you can click on Create Volume in order to begin volume creation. You will be returned to the Volumes page of the dashboard, which will show your new volume as you can see in Figure 8.5.

Attaching a Volume

A volume on its own is not of much value, so we'll now want to attach it to a compute instance. If you do not have an instance running, you can create a basic one with a CirrOS image now in the Instances dashboard. Refer back to Chapter 6 if you need a refresher on the steps to create an instance.

Attaching a volume in the dashboard is done by going to the drop-down menu on the right side of where your volume is listed. From that menu, select Manage Attachments to bring up the screen, where you can attach the volume to an instance (Figure 8.6).

In this example we have an instance running called "giraffe" and the UUID is also included, since names can be reused in compute (Nova). There is also an optional Device Name section where you can define what you want the device to be named when it's attached to the instance. This can safely be left blank and a name will be assigned automatically. When you're done selecting the instance to attach to, click on Attach Volume.



Manage Volume Attachments

Instance	Device	Actions
No items to display.		

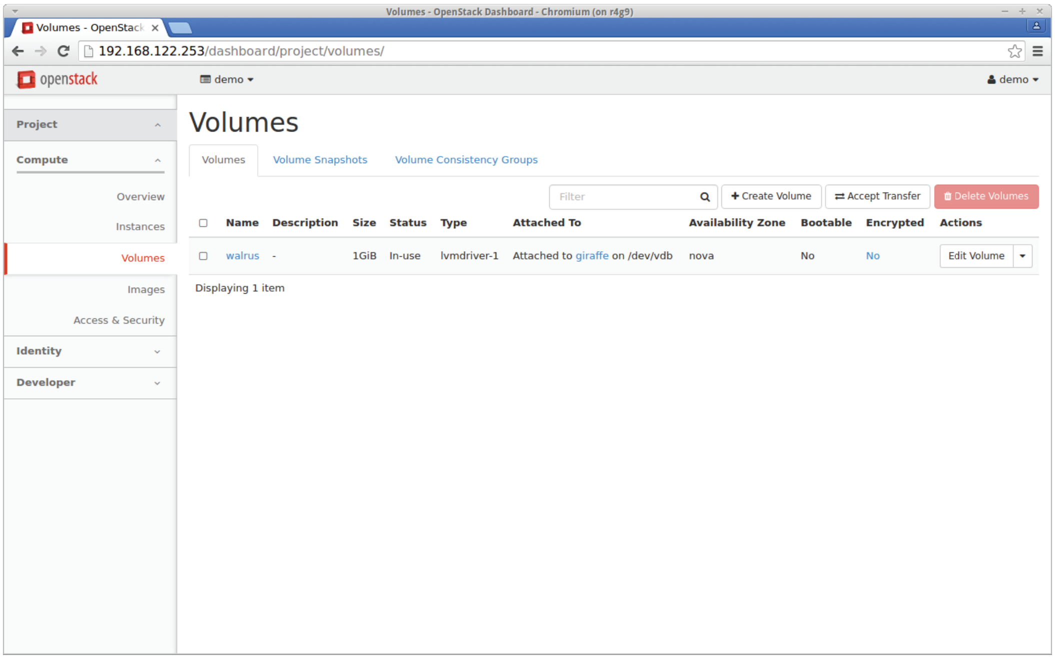
Attach To Instance

Attach to Instance *: giraffe (a1f73c78-b4eb-49f6-8687-dfce23dfd74f)

Device Name: /dev/vdc

Figure 8.6 Managing volume attachments

When the volume completes attaching, you will be able to see it in the dashboard as "Attached to" with the instance name and the device it has shown up as (see Figure 8.7).



Volumes - OpenStack Dashboard - Chromium (on r4g9)

192.168.122.253/dashboard/project/volumes/

Volumes

Name	Description	Size	Status	Type	Attached To	Availability Zone	Bootable	Encrypted	Actions
walrus	-	1GiB	In-use	lvmdriver-1	Attached to giraffe on /dev/vdb	nova	No	No	Edit Volume

Displaying 1 item

Figure 8.7 A volume has been attached.

You'll next want to log into the instance to see that the device has been attached successfully, but this process is the same whether you're completing this process with the dashboard or through the command line. You can continue to learn the process for attaching a volume using the OpenStack Client on the command line, or skip to the "Using the Volume" section later in this chapter to see what you can do to use your new volume.

Creating and Attaching a Volume: OpenStack Client

As we've discussed previously, the dashboard can be a convenient way to interact with OpenStack to complete most of the simple operations you may need to do. You will find, however, that most operators prefer using the command line clients or SDKs to interface with the tooling. As such, we'll now walk through the same process we did with the dashboard but instead using the OpenStack Client (OSC).

The OSC is small and can easily be run from any system that has access to the API endpoints for the services. In our deployment scenarios, this means it must be on the same network as your controller node. You must also have access to the /etc/openrc.test file that was created on your controller and compute nodes, so for these commands we will assume you're running everything on your controller.

Creating a Volume

We will be using the test user in order to create this volume, since it will also be attaching to a compute instance owned by the test user. To begin, we'll bring the environment variables for the test user in from the openrc file. Then we can issue the command to create a 1GB instance using that storage back end. Aside from the name, we will be using the same specifications for creation of the volume as was used with the OpenStack dashboard (Horizon), which means creating a 1GB volume that is empty (no partition table, filesystem or data) and is in our default availability zone, called nova.

> **Tip**
>
> You'll notice that OpenStack commands often output a lot of detail that doesn't fit well on the pages of a book. A GitHub repository of much of this command output sorted by chapter is available at https://github.com/DeploymentsBook/scripts-and-configs.

```
$ source /etc/openrc.test
$ openstack volume create --size 1 --availability-zone nova seaotter
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| consistencygroup_id | None                                 |
| created_at          | 2016-04-15T04:19:46.086611           |
| description         | None                                 |
| encrypted           | False                                |
| id                  | 53372cc5-087a-4342-a67b-397477e1a4f2 |
| multiattach         | False                                |
| name                | seaotter                             |
| properties          |                                      |
| replication_status  | disabled                             |
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | None                                 |
| updated_at          | None                                 |
| user_id             | aa347b98f1734f66b1331784241fa15a     |
+---------------------+--------------------------------------+
```

To confirm this volume has been created, you can run a command to list the volumes (Listing 8.1).

As you can see, both the walrus and the seaotter volumes are listed here since they were both created in this chapter. The walrus volume is showing that it is attached to the giraffe instance.

If you need to make changes to a volume, use the `openstack volume set` command. Running that command alone will give you help output to assist you with making changes to all the parameters before the volume is attached.

Attaching a Volume

As mentioned earlier, you can't do much with a volume if it's not attached to an instance. You'll now want to add your new volume to an instance. First you'll want to see what instances are available:

```
$ openstack server list
+--------------------------------------+---------+--------+------------------+
| ID                                   | Name    | Status | Networks         |
+--------------------------------------+---------+--------+------------------+
| 823f2d7a-f186-4453-874d-4021ff2b22e4 | giraffe | ACTIVE | private=10.0.0.3 |
+--------------------------------------+---------+--------+------------------+
```

With confirmation that you have an instance running, you can now run the command to attach the seaotter volume to the giraffe instance:

```
$ openstack server add volume giraffe seaotter
```

This command will have no output, but the next time you run `volume list` you will see that the volume has been attached (Listing 8.2).

Since the giraffe instance already had the walrus volume attached as /dev/vdb, you will notice that it has attached the seaotter volume as /dev/vdc.

Congratulations, you have successfully added a Cinder block storage volume to an instance on the command line!

Using the Volume

Whether you used the OpenStack dashboard or the command line to create and attach your volume, we will now want to actually confirm the volume was attached and then go ahead and use it with our instance. It may be easiest to use the console in dashboard in order to run the following commands, but if you followed instructions in an earlier chapter so that your CirrOS instance has been set up for SSH (Secure Shell), feel free to use SSH instead.

Listing 8.1

```
$ openstack volume list
+--------------------------------------+--------------+-----------+------+----------------------------------+
| ID                                   | Display Name | Status    | Size | Attached to                      |
+--------------------------------------+--------------+-----------+------+----------------------------------+
| 53372cc5-087a-4342-a67b-397477e1a4f2 | seaotter     | available |    1 |                                  |
| 54447e7a-d39d-4186-a5b4-3a5fc1e773aa | walrus       | in-use    |    1 | Attached to giraffe on /dev/vdb  |
+--------------------------------------+--------------+-----------+------+----------------------------------+
```

Listing 8.2

```
$ openstack volume list
+--------------------------------------+--------------+--------+------+----------------------------------+
| ID                                   | Display Name | Status | Size | Attached to                      |
+--------------------------------------+--------------+--------+------+----------------------------------+
| 53372cc5-087a-4342-a67b-397477e1a4f2 | seaotter     | in-use |    1 | Attached to giraffe on /dev/vdc  |
| 54447e7a-d39d-4186-a5b4-3a5fc1e773aa | walrus       | in-use |    1 | Attached to giraffe on /dev/vdb  |
+--------------------------------------+--------------+--------+------+----------------------------------+
```

Assuming you're using the dashboard, navigate to the Instances screen in the OpenStack dashboard and in the drop-down menu to the right of the instance you attached it to, select Console to bring you to a console for your instance. Once you're on the console page, if you're unable to type in the console, click Click here to show only console and you will be brought to a page that only has the console.

Follow the instructions to log into the instance, and run the following command:

```
$ dmesg
```

There will likely be a lot of output, but the last thing you are likely to see should be something like the following:

```
[  648.143431]  vdb: unknown partition table
```

This vdb device is your new block storage (Cinder) volume! At this phase it has no partition table or filesystem, so this will need to be set up using fdisk. Assuming the device is vdb in this example, partitioning can be done with `fdisk`:

```
$ sudo fdisk /dev/vdb
Device contains neither a valid DOS partition table, nor Sun, SGI or OSF disklabel
Building a new DOS disklabel with disk identifier 0xcf80b0a5.
Changes will remain in memory only, until you decide to write them.
After that, of course, the previous content won't be recoverable.

Warning: invalid flag 0x0000 of partition table 4 will be corrected by w(rite)

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-2097151, default 2048): 2048
Last sector, +sectors or +size{K,M,G} (2048-2097151, default 2097151): 2097151

Command (m for help): p

Disk /dev/vdb: 1073 MB, 1073741824 bytes
16 heads, 63 sectors/track, 2080 cylinders, total 2097152 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: 0xcf80b0a5

   Device Boot      Start         End      Blocks   Id  System
/dev/vdb1            2048     2097151     1047552   83  Linux
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
```

Now you'll want to create a basic filesystem on the new disk. It's only a 1GB volume, and this is a demonstration, so we'll use the ext2 filesystem:

```
$ sudo mkfs.ext2 /dev/vdb1
mke2fs 1.42.2 (27-Mar-2012)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
Stride=0 blocks, Stripe width=0 blocks
65536 inodes, 261888 blocks
13094 blocks (5.00%) reserved for the super user
First data block=0
Maximum filesystem blocks=268435456
8 block groups
32768 blocks per group, 32768 fragments per group
8192 inodes per group
Superblock backups stored on blocks:
           32768, 98304, 163840, 229376

Allocating group tables: done
Writing inode tables: done
Writing superblocks and filesystem accounting information: done
```

The last step is creating a mount point and mounting your new volume. Let's say that you want to use this volume for photos and create a directory for that. Then we'll check to confirm it's the size we expect it to be.

```
$ mkdir photos
$ sudo mount /dev/vdb1 photos/
$ df -h | grep vdb1
/dev/vdb1              1006.9M      1.3M    954.5M   0% /home/cirros/photos
$ df -h /dev/vdb1
Filesystem                Size      Used Available Use% Mounted on
/dev/vdb1              1006.9M      1.3M    954.5M   0% /home/cirros/photos
```

Congratulations! A 1GB volume from the block storage service Cinder is now mounted on your system. Note that this was mounted using the root user, so you will need to either change the ownership to your user or use root to place files on it.

Tip

File permissions with block storage in Cinder can be a tricky thing to master. When using a Linux filesystem, files are referenced by user and group ID (UID and GID). Unless you are very diligent about keeping things consistent across your instances through something like custom images with default users and group or configuration management where IDs are specifically defined, these IDs can easily be different between machines.

As a result of these potential differences in IDs, mounting a volume from the block storage device on one instance and then detaching it to add it to another instance may land you in a situation where the ownership of files looks all wrong. Keep this in mind as you begin experimenting with moving volumes and always make checking permissions a step in your plans to move a volume to another instance.

Automation

As we explained in our chapters about private and public clouds, you don't only need to interact with OpenStack through the OpenStack dashboard or OpenStack client. Instead you may interact with the APIs through various SDKs, which you can learn about at http://developer.openstack.org/.

Summary

The need for expanding and moving storage of data is growing in modern environments, and Cinder block storage fits that need. It offers a variety of storage back-end drivers to support everything from open source tooling like LVM and Ceph to a host of tested proprietary storage solutions from an array of vendors. The deployment scenario using Puppet walked you through creating a volume and then attaching it to an instance where you could use it.

Index