



Identity, Authentication & Access Management in OpenStack

IMPLEMENTING AND DEPLOYING KEYSTONE

Steve Martinelli,
Henry Nash & Brad Topol

Identity, Authentication, and Access Management in OpenStack

Keystone—OpenStack's Identity service—provides secure controlled access to a cloud's resources. In OpenStack environments, Keystone performs many vital functions, such as authenticating users and determining what resources users are authorized to access.

Whether the cloud is private, public, or dedicated, access to cloud resources and security is essential. This practical guide to using Keystone provides detailed, step-by-step guidance to creating a secure cloud environment at the Infrastructure-as-a-Service layer—as well as key practices for safeguarding your cloud's ongoing security.

- Learn about Keystone's fundamental capabilities for providing Identity, Authentication, and Access Management
- Perform basic Keystone operations, using concrete examples and the latest version (v3) of Keystone's Identity API
- Understand Keystone's unique support for multiple token formats, including how it has evolved over time
- Get an in-depth explanation of Keystone's LDAP support and how to configure Keystone to integrate with LDAP
- Learn about one of Keystone's most sought-after features—support for federated identity

Steve Martinelli is an OpenStack Active Technical Contributor and a Keystone Core Contributor. He primarily focuses on enabling Keystone, which is OpenStack's Identity Manager, to better integrate into enterprise environments. He also contributes to OpenStackClient, pyCADF, and oslo.policy and is a core contributor in each of those projects.

Henry Nash works in IBM's Cloud division as an OpenStack Architect and a core contributor to OpenStack Keystone, driving enterprise capabilities into OpenStack as well as IBM's products that use OpenStack. He has a long history of developing enterprise software, graphics, and communication systems as well as nanotechnology.

Dr. Brad Topol, an IBM Distinguished Engineer in the IBM Cloud Architecture and Technology organization, leads a development team focused on contributing to and improving OpenStack. He has cross-IBM responsibility for coordinating its contributions to OpenStack.

OPERATIONS / SYSTEM ADMINISTRATION

US \$24.99 CAN \$28.99

ISBN: 978-1-491-94120-1



9 781491 941201



Twitter: @oreillymedia
facebook.com/oreilly

Identity, Authentication, and Access Management in OpenStack

*Implementing and Deploying Keystone,
OpenStack's Identity Service*

Steve Martinelli, Henry Nash, and Brad Topol

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Identity, Authentication, and Access Management in OpenStack

by Steve Martinelli, Henry Nash, and Brad Topol

Copyright © 2016 Steve Martinelli, Henry Nash, and Brad Topol. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://safaribooksonline.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Acquisitions Editor: Rachel Roumeliotis

Indexer: WordCo Indexing Services, Inc.

Editor: Nan Barber

Interior Designer: David Futato

Production Editor: Dan Fauxsmith

Cover Designer: Ellie Volckhausen

Proofreader: Christina Edwards

Illustrator: Rebecca Demarest

October 2015: First Edition

Revision History for the First Edition

2015-09-28: First Release

2015-12-07: Second Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781449370787> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Identity, Authentication, and Access Management in OpenStack*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

While the publisher and the authors have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-491-94120-1

[LSI]

To Dini, for your never-ending support and love, and to my parents and grandfather for their patience, selflessness and unwavering support.

—Steve Martinelli

To my wife Susan, for her incredible love and support (and for putting up with a steady stream of my crazy ideas over the years), and to everyone who strives to disrupt the status quo with new ways of thinking. Remember, if it was easy, someone else would have already done it.

—Henry Nash

I dedicate this book to my wife Janet, my daughter Morgan, and my son Ryan. I could not have done this without your love and support during this process.

—Brad Topol

Table of Contents

Preface.....	ix
Introduction.....	xv
1. Fundamental Keystone Topics.....	1
1.1 Keystone Concepts	1
1.1.1 What's a Project?	1
1.1.2 What's a Domain?	2
1.1.3 Users and User Groups (Actors)	2
1.1.4 Roles	3
1.1.5 Assignment	3
1.1.6 Targets	3
1.1.7 What's a Token?	4
1.1.8 What's a Catalog?	5
1.2 Identity	5
1.2.1 SQL	6
1.2.2 LDAP	6
1.2.3 Multiple Backends	7
1.2.4 Identity Providers	8
1.2.5 Use Cases for Identity Backends	9
1.3 Authentication	9
1.3.1 Password	10
1.3.2 Token	11
1.4 Access Management and Authorization	12
1.5 Backends and Services	13
1.6 FAQs	14

2. Let's Use Keystone!	17
2.1 Getting DevStack	17
2.2 Basic Keystone Operations Using OpenStackClient	18
2.2.1 Getting a Token	19
2.2.2 Listing Users	21
2.2.3 Listing Projects	22
2.2.4 Listing Groups	24
2.2.5 Listing Roles	25
2.2.6 Listing Domains	26
2.2.7 Creating Another Domain	27
2.2.8 Create a Project within the Domain	27
2.2.9 Create a User within the Domain	28
2.2.10 Assigning a Role to a User for a Project	29
2.2.11 Authenticating as the New User	30
2.3 Basic Keystone Operations Using Horizon	31
2.3.1 What Keystone Operations Are Available through Horizon?	31
2.3.2 Accessing the Identity Operations	32
2.3.3 List, Set, Delete, Create, and View a Project	32
2.3.4 List, Set, Delete, Create, and View a User	32
2.4 Tips, Common Pitfalls, and Troubleshooting	33
Check Your Scope: A Common Authentication Problem	33
Check Your Policy and Role: A Common Authorization Problem	34
Getting Additional Information	34
3. Token Formats.....	35
3.1 History of Keystone Token Formats	35
3.2 UUID Tokens	37
3.3 PKI Tokens	37
3.4 Fernet Tokens	40
3.5 Tips, Common Pitfalls, and Troubleshooting	42
3.5.1 UUID Token Performance Degradation for Authentication Operations	42
3.5.2 Using PKI Token and Swift or Horizon Not Working?	42
4. LDAP.....	45
4.1 Approach to LDAP Integration	45
4.2 Configuring Keystone to Integrate with LDAP	46
4.2.1 Other Keystone Configuration Options in Classic LDAP Support	50
4.3 Multiple Domains and LDAP	55
4.3.1 Requirements for Multi-Domain Corporate Directory Support	55
4.3.2 Setting Up Multi-Domain Using the Configuration File-Based Approach	57

4.3.3 Setting Up Multi-Domain Using the Keystone API-Based Approach	58
4.3.4 Restrictions When Using Multi-Domain Identity	61
Use SQL for the Default Domain	62
Use LDAP for All Domains, Except an SQL Service Domain	63
Use LDAP for All Domains	64
4.4 A Practical Guide to Using Multi-Domains and Keystone	64
4.4.1 Setting Up LDAP	64
4.4.2 Running Admin Commands	67
4.4.3 Running LDAP User Commands	68
4.4.4 Authenticating with Horizon	70
4.5 Projects, Roles, and Assignments from LDAP (Just Say NO!)	72
4.6 Tips, Common Pitfalls, and Troubleshooting	73
4.6.1 General LDAP Issues	73
4.6.2 Tips for Using Multi-Domain LDAP	74
5. Federated Identity.....	77
5.1 Approach to Federation	78
5.1.1 Leveraging Existing Technology	78
5.1.2 Keystone-Specific Federation Concepts	79
5.2 Translating User Attributes to Keystone Concepts	80
5.2.1 OpenID Connect Claims	80
5.2.2 SAML Assertions	81
5.2.3 The Mapping Engine	82
5.2.4 Mapping Rules	83
5.3 Authentication Flow: What's It Look Like?	85
5.4 Single Sign-On	86
Single Sign-On Flow	86
5.5 A Practical Guide to Federating Identities for IBM WebSphere Liberty and Bluepages	87
5.5.1 Download, Install, and Configure IBM WebSphere Liberty	87
5.5.2 Configuring Keystone to Use OpenID Connect	90
5.5.3 Testing It All Out	92
5.6 A Practical Guide to Setting Up SSO with Google	93
5.6.1 Configure Keystone to Use OpenID Connect	93
5.6.2 Configure Horizon for Single Sign-On	95
5.6.3 Let's See It with Screenshots!	96
5.7 Tips, Common Pitfalls, and Troubleshooting	99
Ensure All Libraries Are Installed	99
Known Limitations of Social Media Logins	99
Using SAML from the Command Line	99

6. Future Work.....	101
6.1 Multi-Factor Authentication	101
6.2 Integration with Horizon for Multi-Region Keystone to Keystone Federation Support	102
6.3 Using LDAP as a Federated Identity Provider	102
6.4 Replacement of Service Users with X.509 Certificates and Barbican Integration	102
6.5 Centralized Policy and Distribution	103
6.6 Integrating with Other Technologies	103
Index.....	105

Preface

Prologue

A key aspect to setting up a cloud, whether it be private, public, or dedicated, is ensuring that access to cloud resources and security are in place. For OpenStack environments, the focal point for securing the cloud is Keystone, OpenStack’s Identity service. Keystone provides many key functions, such as authenticating users and determining what resources users are authorized to access.

Keystone started from humble beginnings. In the early days, it provided basic user management and constructs for organized access to resources. As enterprise customers became more interested in OpenStack, it became readily apparent that Keystone needed significant enhancements to meet the demanding needs of enterprise customers before it would be adopted in the enterprise.

Early enterprise requirements were focused on improving Keystone’s Lightweight Directory Access Protocol (LDAP) and Microsoft’s Active Directory support. Enterprise customers want to reuse their existing identity-management tools and don’t want a separate new identity tool to manage their OpenStack users. Support was added to ensure Keystone could reuse existing LDAPs and Active Directories that were read only and only contained user and group information. Next, secure connections were added so Keystone could connect to LDAP and Active Directories over a TLS connection.

This basic support for ease of integration with existing enterprise identity managers helped OpenStack to distinguish itself from competing cloud infrastructures. It then led to a second phase of advanced enterprise integration where customers demanded support for integration with multiple LDAPs and Active Directories. This feature was critical for large enterprises that, through means such as acquisitions, had multiple identity servers they needed to support. Also as part of this phase, customers started requesting true federated identity support whereby they expected Keystone to integrate with federated identity-management tools that supported well-known and stan-

dard identity protocols such as the Security Assertion Markup Language (SAML) and OpenID Connect. Around this time Keystone started to add audit support to better enable it to meet the compliance requirements of many enterprise customers.

With the foundations of federated support, Keystone has moved to its current phase, focusing on federated support for hybrid clouds. With this federation support foundation in place, multiple Keystones can work together using standard federated protocols to support interoperable hybrid clouds. In this book, we describe all of these enhancements. We begin by providing an overview of how to perform basic Keystone operations, and we provide concrete examples using the latest version (v3) of the Keystone Identity API. We then cover Keystone’s support for multiple token formats and describe how its preferred token formats have evolved over time. After we discuss these fundamentals, we move on to advanced topics of LDAP integration and federation. We conclude with a discussion on topics of future work for Keystone.

Conventions Used in This Book

The following typographical conventions are used in this book:

Italic

Indicates new terms, URLs, email addresses, filenames, and file extensions.

Constant width

Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, databases, data types, environment variables, statements, and keywords.

Constant width bold

Shows commands or other text that should be typed literally by the user.

Constant width italic

Shows text that should be replaced with user-supplied values or by values determined by context.



This element signifies a tip or suggestion.



This element signifies a general note.



This element indicates a warning or caution.

Using Code Examples

This book is here to help you get your job done. In general, if example code is listed in this book, you may use it in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Identity, Authentication, and Access Management in OpenStack* by Steve Martinelli, Henry Nash, and Brad Topol (O'Reilly). Copyright 2015 Steve Martinelli, Henry Nash, and Brad Topol, 978-1-491-94120-1."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at permissions@oreilly.com.

Safari® Books Online



Safari Books Online is an on-demand digital library that delivers expert **content** in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of **plans and pricing** for **enterprise, government, education**, and individuals.

Members have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kauf-

mann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and hundreds **more**. For more information about Safari Books Online, please visit us [online](#).

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at <http://www.oreilly.com/catalog/0636920045960>.

To comment or ask technical questions about this book, send email to bookquestions@oreilly.com.

For more information about our books, courses, conferences, and news, see our website at <http://www.oreilly.com>.

Find us on Facebook: <http://facebook.com/oreilly>

Follow us on Twitter: <http://twitter.com/oreillymedia>

Watch us on YouTube: <http://www.youtube.com/oreillymedia>

Acknowledgments

We would like to thank the entire OpenStack Keystone community for their passion, dedication, and tremendous commitment to the Keystone project. Without the code developers, code reviewers, and operators contributing to the project over the years, Keystone would not have the rich feature set and large adoption it has today.

We would also like to thank our OpenStack Keystone colleagues, Dolph Mathews, Morgan Fainberg, Brant Knudson, Lance Bragstad, Jamie Lennox, David Stanek, Adam Young, Joe Heck, Marek Denis, Nathan Kinder, and Lin Hua Cheng for the wonderful collaboration over the years.

A very special thanks to Dr. Angel Luis Diaz, Todd Moore, and Vince Brunssen for all of their support and encouragement during this endeavor.

—Steve, Henry, and Brad

Introduction

Identity, Authentication, and Access Management Capabilities of Keystone

Cloud environments at the Infrastructure-as-a-Service layer (IaaS) provide users access to key resources such as virtual machine instances, large amounts of block and object storage, and network bandwidth. A critical feature of any cloud environment is how it provides secure, controlled access to these valuable resources. In OpenStack environments, the Keystone service is responsible for providing secure controlled access to all of the cloud's resources. Keystone has proven itself to be a vital component for a secure cloud. As shown in [Figure P-1](#), 90% to 95% of OpenStack users report that they utilize Keystone in their deployments.

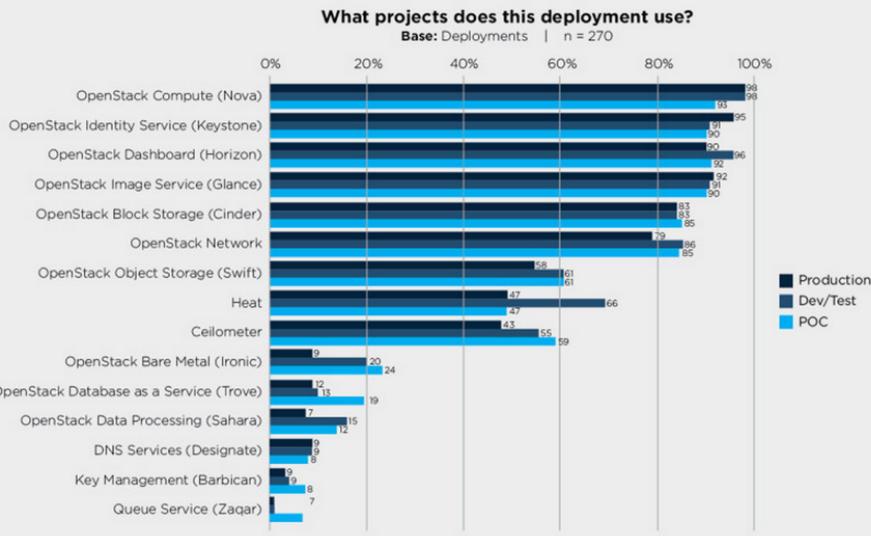


Figure P-1. Percentage of deployments using Keystone and other OpenStack services ([source](#))

In order to understand how Keystone provides secure and controlled access to OpenStack Cloud resources we need to examine Keystone's fundamental capabilities for providing Identity, Authentication, and Access Management. The following sections provide short overviews of these core Keystone capabilities.

Identity

Identity refers to the identification of who is trying to access cloud resources. In OpenStack Keystone, identity is typically represented as a user. In simple deployments, the identity of a user can be stored in Keystone's own database. In production or enterprise environments, an external Identity Provider is commonly used. An example of this is IBM's Tivoli Federated Identity Manager. Keystone should be able to retrieve the user's identity information from these external Identity Providers.

Authentication

Authentication is the process of validating a user's identity. In many cases, authentication is initially performed by a user performing a login with their user identity and a password. In a very rudimentary OpenStack environment, Keystone is capable of performing all the authentication steps itself. This is not recommended for production or enterprise environments. For real environments where passwords need to be prop-

erly protected and managed, Keystone is pluggable such that it easily integrates with an existing hardened production authentication service, such as LDAP or Active Directory.

While a user identity is typically initially authenticated with a password, it is very common as part of this initial authentication to create a token for subsequent authentications. This reduces the amount of visibility and exposure of the password which needs to be kept hidden and protected as much as possible. Tokens also have a limited lifespan and expire so that their usefulness is limited if they are stolen. OpenStack relies heavily on tokens for authentication and other purposes—and Keystone is the one and only OpenStack service that can issue them. Currently, Keystone uses a form of token called a *bearer token*. This means that whomever has obtained ownership of the token, whether it be properly or improperly (i.e., stolen), is capable of using the token to authenticate and access resources. As a result, when using Keystone it is very important to protect tokens and their contents.

Access Management (Authorization)

Once a user identity has been authenticated and a token has been created and allocated, things start to get interesting. This is because we now have enough foundation in place to start performing Access Management. *Access Management*, also referred to as *Authorization*, is the process of determining what resources a user is permitted to access. Cloud environments such as OpenStack provide users with access to large amounts of resources. For example, there needs to be a mechanism for determining which users are allowed to create new instances of a particular virtual machine, which users are allowed to attach or delete a volume of block storage, which users are allowed to create virtual networks, etc. In OpenStack, Keystone maps Users to Projects or Domains by associating a Role for the User for that Project or Domain. Other OpenStack subprojects such as Nova, Cinder, and Neutron examine the User's Project and Role associations and evaluate this information using a policy engine. The policy engine examines this information (particularly the Role value) and makes a determination about what actions the user is allowed to perform.

Keystone's Primary Benefits

While Keystone is mostly focused on providing Identity, Authentication, and Access Management it does provide a large number of benefits for an OpenStack environment. Key benefits include the following:

- Single Authentication and Access Management interface for other OpenStack services. Keystone handles the complex tasks of integrating with external Authentication systems and also provides uniform Access Management for all the other OpenStack services, such as Nova, Glance, Cinder, Neutron, etc., and

thus Keystone isolates all the other services from knowing how to talk to different identity and authorization providers.

- Keystone provides a registry of containers (“Projects”) that other OpenStack services can use to segregate resources (e.g., servers, images, etc.).
- Keystone provides a registry of Domains that are used to define separate namespaces for users, groups, and projects to allow segregation between customers.
- A registry of Roles that will be used for authorization between Keystone and the policy files of each of the OpenStack services.
- An assignment store allowing users and groups to be assigned roles on projects and domains.
- A catalog storing OpenStack services, endpoints, and regions, allowing clients to discover the service or endpoint they need.

The rest of the book is organized as follows. In [Chapter 1](#), we provide an overview of Keystone’s fundamental constructs. [Chapter 2](#) provides the reader with an instructional tutorial for interacting with Keystone to invoke its core operational capabilities. In [Chapter 3](#) we provide extensive coverage of the variety of token formats supported by Keystone and highlight the advantages and disadvantages of each format as well describing how each can be configured and utilized. [Chapter 4](#) provides in-depth coverage of how to integrate Keystone with an LDAP Identity Provider. [Chapter 5](#) describes Keystone’s approach to federation and illustrates how to integrate Keystone with a federated identity provider. Finally, in [Chapter 6](#) we conclude with a discussion of areas for future improvement of Keystone.

Fundamental Keystone Topics

In this chapter we provide an introduction to the basic foundations of Keystone. We start with an overview of Keystone Projects and Domains, which are abstractions used to group and isolate resources. We then discuss how Keystone supports Users and User Groups and how Roles can be assigned to Users and User Groups on both a Project and Domain basis. We then introduce how Keystone utilizes Tokens and provides Service Catalogs. Next, we describe Keystone's Identity service and the types of Identity backends that can be leveraged by Keystone. We then conclude this chapter with in-depth descriptions of Keystone's Authentication and Access Management (Authorization) capabilities.

1.1 Keystone Concepts

Keystone itself has several concepts that are specific to its model and how it relates to OpenStack as a whole. These are Identity and Authorization related concepts, but their focus is on how Keystone implements Authorization, Access Management, and Discovery.

1.1.1 What's a Project?

In Keystone, a Project is an abstraction used by other OpenStack services to group and isolate resources (e.g., servers, images, etc.). In the early days of OpenStack, Keystone Projects were originally referred to as Tenants but this was changed to Projects, a more intuitive name for this concept. It is probably fair to say that the most fundamental purpose of Keystone is to be the registry of Projects and to be able to articulate who should have access to those Projects. Projects themselves don't own Users, but Users or User Groups are given access to a Project using the concept of Role Assignments. Having a Role assigned to a User or User Group on a Project denotes that the User or User Group has some kind of access to resources in the Project, and

the specific Role selected determines the type of access and capabilities the User or User Group is entitled to have. Assigning a Role to a User is sometimes referred to as a “grant” in OpenStack documentation. Roles, Users, and User Groups are more thoroughly described in later sections of this chapter.

1.1.2 What’s a Domain?

In the early days of OpenStack, there was no mechanism to limit the visibility of Projects to different user organizations. This could result in unintentional collisions on Project names by different organizations. User names also had global visibility and could also result in undesired collisions on user names if two different organizations both used the same user name. In order for OpenStack clouds to more cleanly support multiple user organizations at the same time, Keystone added a new abstraction, called a *Domain*, that could provide the ability to isolate the visibility of a set of Projects and Users (and User Groups) to a specific organization. A Domain is formally defined as a collection of users, groups, and projects. Domains enable you to divide the resources in your cloud into silos that are for use by specific organizations. A domain can serve as a logical division between different portions of an enterprise, or each domain can represent completely separate enterprises.

For example, a cloud could have two domains, IBM and Acme Inc. IBM has their own collection of groups, users, and projects and so does Acme Inc.

1.1.3 Users and User Groups (Actors)

In the Keystone realm, Users and User Groups are the entities given access to resources that are isolated in Domains and Projects. Groups are a collection of Users. Users are individuals who will end up using your cloud. We refer to Users and Groups as *Actors* since, when assigning a role, these are the entities to which the role is “assigned to.”

1.1.3.1 Graphical representation

The relationship between domains, projects, users, and groups is shown below. Users, groups, and projects will always be “domain scoped,” meaning that a name for a user, group, and project *may* be common across domains. Think of this as having a user “Jim” or group “administrators” at both IBM and Acme Inc. However, each entity is guaranteed to have a *universally unique identifier* (UUID). This implies that trying to find a resource with just the ID will work. However, since names are not guaranteed to be unique, providing the domain is essential to find the resource. As of Liberty, roles are not domain scoped, but this could change in the future.

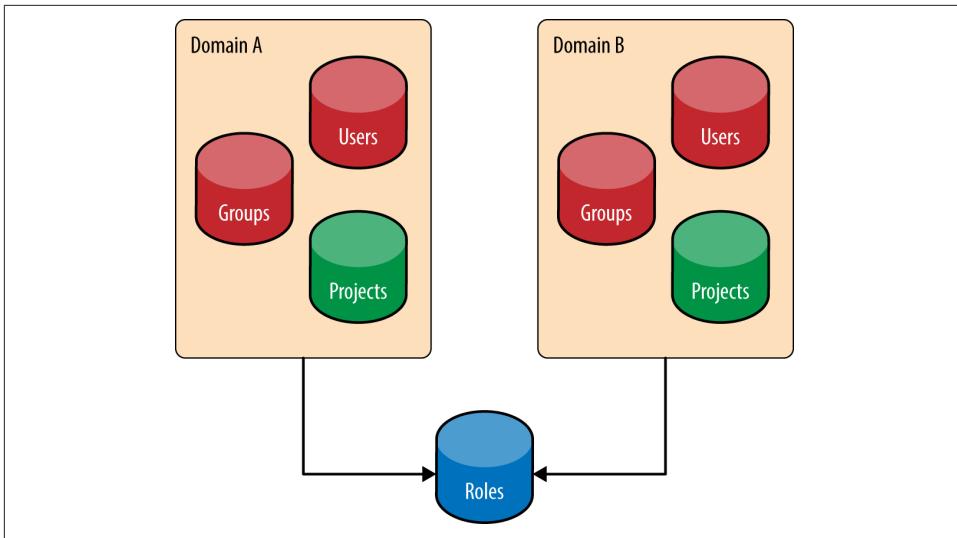


Figure 1-1. Domains are a collection of Users, Groups, and Projects. Roles are globally unique. Users may have membership in many Groups.

1.1.4 Roles

Roles are used in Keystone to convey a sense of Authorization. An actor may have numerous roles on a target. How the roles are used to determine authorization is covered in “[1.4 Access Management and Authorization](#)” on page 12. For example, the role of admin is “assigned to” the user “bob” and it is “assigned on” the project “development.”

1.1.5 Assignment

A role *assignment* is a ternary (or triple): the combination of an actor, a target, and a role. Role assignments are granted and revoked, and may be inherited between groups and users and domains and projects.

1.1.6 Targets

Projects and Domains are very similar in that both are entities where the role is “assigned on.” In other words, a User or User Group is given access to either a Project or Domain by assigning a particular Role value for that User or User Group for a specific Project or Domain. Because Projects and Domains have such similar characteristics, when we need to refer to both entities collectively we refer to them as *Targets*.

1.1.7 What's a Token?

In order for a user to call any OpenStack API they need to (a) prove who they are, and (b) that they should be allowed to call the API in question. The way they achieve that is by passing an OpenStack token into the API call—and Keystone is the OpenStack service responsible for generating these tokens. A user receives this token upon successful authentication against Keystone. The token also carries with it authorization. It contains the authorization a user has on the cloud. A token has both an ID and a payload. The ID of a token is guaranteed to be unique per cloud, and the payload contains data about the user.

The payload can be seen below:

Example 1-1. An Identity V3 Scoped Token consists of many fields that indicate Identity and Authorization attributes about the user on a project

```
{  
    "token": {  
        "issued_at": "201406-10T20:55:16.806027Z",  
        "expires_at": "2014-06-10T2:55:16.806001Z",  
        "roles": [  
            {"id": "c703057be878458588961ce9a0ce686b",  
             "name": "admin"}  
        ],  
        "project": {  
            "domain": { "id": "default",  
                       "name": "Default" },  
            "id": "8538a3f13f9541b28c2620eb19065e45",  
            "name": "admin"  
        },  
        "user": {  
            "domain": { "id": "default",  
                       "name": "Default" },  
            "id": "3ec3164f750146be97f21559ee4d9c51",  
            "name": "admin"  
        },  
        "catalog": [  
            {  
                "endpoints": [...],  
                "type": "identity",  
                "id": "bd73972c0e14fb69bae8ff76e112a90",  
                "name": "keystone"  
            }  
        ]  
    }  
}
```

As you can tell, the token payload contains information about when it was created, when it will expire, which user authenticated—and thus is allowed to use the token—

which project the token is valid on, and finally, the catalog, which brings us to our next point. Tokens will be discussed in detail in [Chapter 3](#).

1.1.8 What's a Catalog?

The service catalog is essential for an OpenStack cloud. It contains the URLs and endpoints of the different Cloud services. Without the catalog, users and applications would not know where to route requests to create VMs or store objects. The service catalog is broken up into a list of endpoints, and each endpoint is broken down into an admin URL, internal URL, and public URL, which may be the same.

Example 1-2. A simple Catalog of just two services, Identity and Object Storage. Endpoints are essential to allow the user to discover where different services are hosted.

```
{
    "serviceCatalog": [
        {
            "endpoints": [
                {
                    "adminURL": "http://swift.admin-nets.local:8080/",
                    "region": "RegionOne",
                    "internalURL": "http://127.0.0.1:8080/v1/AUTH_1",
                    "publicURL": "http://swift.publicinternets.com/v1/AUTH_1"
                }
            ],
            "type": "object-store",
            "name": "swift"
        },
        {
            "endpoints": [
                {
                    "adminURL": "http://cdn.admin-nets.local:35357/v2.0",
                    "region": "RegionOne",
                    "internalURL": "http://cdn.admin-nets.local:5000/v2.0",
                    "publicURL": "http://cdn.admin-nets.local:5000/v2.0"
                }
            ],
            "type": "identity",
            "name": "keystone"
        }
    ]
}
```

1.2 Identity

The Identity Service in Keystone provides the Actors. Identities in the Cloud may come from various locations, including but not limited to SQL, LDAP, and Federated Identity Providers.

1.2.1 SQL

Keystone includes the option to store your actors (Users and Groups) in SQL; supported databases include MySQL, PostgreSQL, and DB2. Keystone will store information such as name, password, and description. The settings for the database must be specified in Keystone's configuration file. Essentially, Keystone is acting as an Identity Provider, which may not be the best case for everyone, and certainly not the best case for enterprise customers. The following provides a brief summary of the advantages and disadvantages of using the SQL Identity option.

Pros:

- Easy to set up
- Management of users and groups through OpenStack APIs

Cons:

- Keystone should not be an Identity Provider
- Weak password support
 - No password rotation
 - No password recovery
- Most enterprises have an LDAP server they want to use
- Identity silo: yet another username and password users must remember

1.2.2 LDAP

Keystone also has the option to retrieve and store your actors (Users and Groups) in Lightweight Directory Access Protocol (LDAP). Keystone will access the LDAP just like any other application that uses the LDAP (System Login, Email, Web Application, etc.). The settings for connecting to the LDAP are specified in Keystone's configuration file. These options also include whether Keystone is able write to LDAP or simply read the LDAP data. Ideally, the LDAP should perform only read operations, such as user and group lookup (via search) and authentication (via bind). If using LDAP as a read-only Identity backend, Keystone should need a minimal amount of privilege to use the LDAP. For instance, it needs read access to the user and group attributes defined in `keystone.conf`, an unprivileged account (anonymous access is preferred), and it does *not* require access to password hashes. The advantages and disadvantages of the LDAP Identity option are shown below.

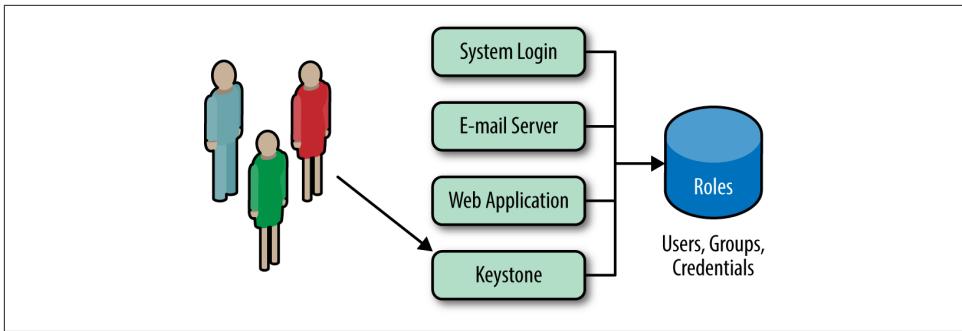


Figure 1-2. Keystone should use an internal LDAP just like any other application

Pros:

- No longer need to maintain copies of user accounts.
- Keystone does not act as an Identity Provider.

Cons:

- Service accounts still need to be stored somewhere, and the LDAP admin may not want these accounts in LDAP.
- Keystone is still “seeing” user passwords, since the passwords are in the authentication request. Keystone simply forwards these requests, but ideally Keystone does not want to see a user’s password, ever!

1.2.3 Multiple Backends

As of the Juno release, Keystone supports multiple Identity backends for the V3 Identity API. The impact of this is that a deployment may have one identity source (backend) per Keystone domain. The default domain is usually an SQL backend, as it is used to host service accounts. Service accounts are the accounts that the different OpenStack services use to interact with Keystone. Any additional LDAP backends may be hosted in their own domain. The motivation for supporting multiple Identity backends is that in an enterprise setting, LDAP administrators may not be the same organization as the OpenStack deployment team, so creating service accounts in LDAP is highly unlikely. LDAP is typically restricted to be used only for employee information. Another benefit of a logical split between Identity backends and domains is that now multiple LDAPs can be used. So, in the case of a company merger or different departments having different LDAPs, the same enterprise can still be represented. The advantages and disadvantages of multiple backends are as follows.

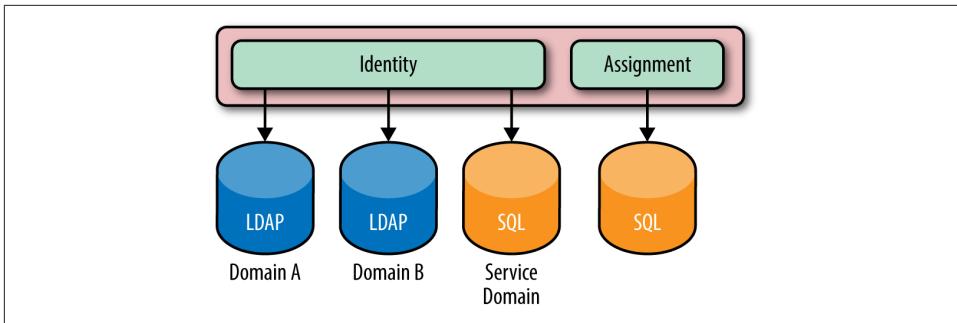


Figure 1-3. The Identity service may have multiple backends per domain. LDAPs for both Domains A and B, and an SQL-based backend for the service accounts is usual. The Assignment service is also shown to remind readers that the assignment, resource, and identity service may all be backed by various stores.

Pros:

- Able to simultaneously support multiple LDAPs for various user accounts and SQL backends for service accounts
- Leverage existing identity LDAP while not impacting it

Cons:

- Slightly more complex to set up
- Authentication for user accounts must be domain scoped

1.2.4 Identity Providers

As of the Icehouse release, Keystone is able to consume federated authentication via Apache modules for multiple trusted Identity Providers. These users are not stored in Keystone, and are treated as ephemeral. The federated users will have their attributes mapped into group-based role assignments. From a Keystone perspective, an identity provider is a source for identities; it may refer to software that is backed by various backends (LDAP, AD, MongoDB) or Social Logins (Google, Facebook, Twitter). Essentially, it is software (IBM's Tivoli Federated Identity Manager, for instance) that abstracts out the backend and translates user attributes to a standard federated identity protocol format (SAML, OpenID Connect). This fits well in Keystone's greater plan to offload the authentication and identity related portions to a service that already does this in an enterprise. The Identity Provider approach has lots of advantages and few disadvantages.

Pros:

- Able to leverage existing infrastructure and software to authenticate users and retrieve information about users
- Further separation between Keystone and handling identity information
- Opens the door for new possibilities in the federation realm, such as single sign-on and hybrid cloud
- Keystone does not see any user passwords
- Identity provider handles authentication completely, so whether it is password, certificate, or two-factor based is irrelevant to Keystone

Cons:

- Most complex set up of the identity sources

1.2.5 Use Cases for Identity Backends

The use cases for the different identity backends are organized in [Table 1-1](#).

Table 1-1. List of identity sources and their most common use case

Identity Source	Use Cases
SQL	<ul style="list-style-type: none"> • Use this if you're testing or developing with OpenStack • Small set of users • OpenStack-specific accounts (service users)
LDAP	<ul style="list-style-type: none"> • Use this if it's already in place in your business • Use <i>only</i> LDAP if you are able to create the service accounts necessary in the LDAP
Multiple Backends	<ul style="list-style-type: none"> • Preferred approach for most enterprises • Use if service users are not allowed in LDAP
Identity Provider	<ul style="list-style-type: none"> • You want to take advantage of the new Federated Identity mechanisms • Use this if an identity provider already exists (for single sign-on) • Keystone is unable to access LDAP • Non-LDAP identity source • Use if LDAP interaction is offloaded to underlying platform and Web server (SSSD)

1.3 Authentication

There are various ways to authenticate with the Keystone service—by far the two most common are by supplying a password or by using a token. In this section we will be highlighting those two methods of authentication by showing the data required in a POST request to Keystone and their usual flow between the User, Keystone, and other OpenStack services.

1.3.1 Password

The most common way for a user or service to authenticate is to supply a password. The payload shown below is a sample POST request to Keystone. It is helpful to show the entire payload so the reader recognizes the information that is necessary to authenticate.

Example 1-3. An example of an authentication payload request that contains a scope.

```
{  
    "auth": {  
        "identity": {  
            "methods": [  
                "password"  
            ],  
            "password": {  
                "user": {  
                    "domain": {  
                        "name": "example.com"  
                    },  
                    "name": "Joe",  
                    "password": "secretsecret"  
                }  
            }  
        }  
    },  
    "scope": {  
        "project": {  
            "domain": {  
                "name": "example.com"  
            },  
            "name": "project-x"  
        }  
    }  
}
```

About the payload, and a note about domains

The payload of the request must contain enough information to find where the user exists, authenticate the user, and optionally, retrieve a service catalog based on the user's permissions on a scope (project).

The `user` section that identifies the incoming user should have domain information (either the domain name or ID), unless a user's globally unique ID is used, in which case that is sufficient to identify the user. This is because in a multi-domain deployment, there may be multiple users with the same name, so proper scoping is necessary to determine which user is authenticating.

The scope section is optional but is often used, since without a scope a user will not be able to retrieve a service catalog. The scope is used to indicate which project the user wishes to work against. If a user does not have a role on that project, then the request will be rejected. Similar to the user section, the scope section must have enough information about the project to find it, so the owning domain must be specified. As in the case of users and groups, project names may also conflict across domains. The project ID, however, is guaranteed to be unique and if specified, no domain information is necessary.

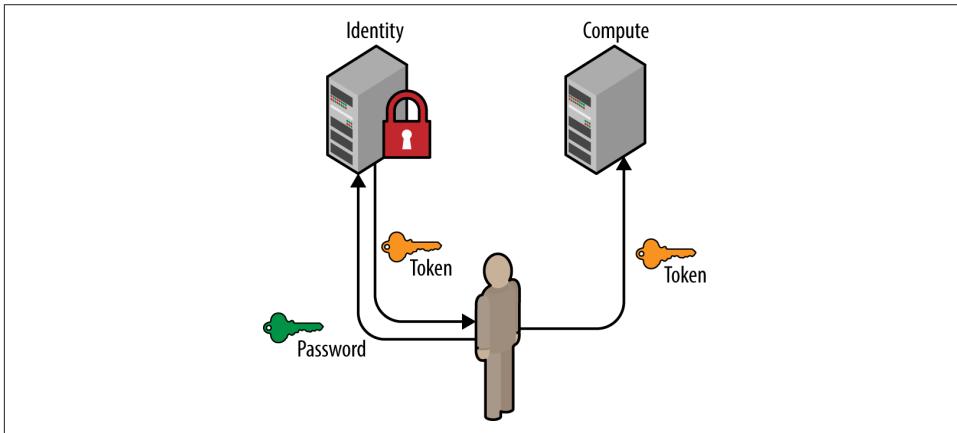


Figure 1-4. A user requests a token by using their username, password, and project scope. The resultant token may then be used at other OpenStack services (such as Compute or Object Storage).

1.3.2 Token

Similar to above, a user may also request a new token by providing a current token. The payload of this POST request is significantly less code than its password counterpart. There are many reasons why a token will be used to retrieve another, such as refreshing a token that will soon expire or to change an unscoped token to a scoped token.

```
{  
    "auth": {  
        "identity": {  
            "methods": [  
                "token"  
            ],  
            "token": {  
                "id": "e80b74"  
            }  
        }  
    }  
}
```

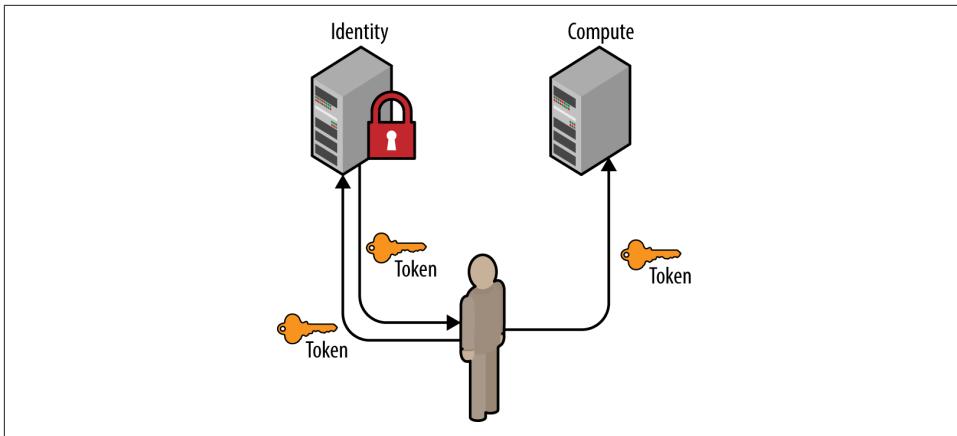


Figure 1-5. A user requests a token by using an existing token. The resultant token will have the same scope and roles as the initial token.

1.4 Access Management and Authorization

Managing access and authorizing what APIs a user can use is one of the key reasons Keystone is essential to OpenStack. Keystone's approach to this problem is to create a Role-Based Access Control (RBAC) policy that is enforced on each public API endpoint. These policies are stored in a file on disk, commonly named *policy.json*. To better understand this concept, it's best to jump right into an example.

The following is a brief example of Keystone's *policy.json* file, which is comprised of targets and rules.

Targets refer to the left-hand key, and rules refer to the right-hand value. At the top of the file, targets are established that can be used for evaluation of other targets. It is here that we define what it means to be an admin, an owner, or either.

Example 1-4. Snippet of Keystone's policy.json file

```
{
    "admin_required": "role:admin or is_admin:1",
    "owner" : "user_id:(%user_id)s",
    "admin_or_owner": "rule:admin_required or rule:owner",

    "identity:list_projects": "rule:admin_required",
    "identity:create_project": "rule:admin_required",
    "identity:delete_project": "rule:admin_required",

    "identity:list_user_projects": "rule:admin_or_owner"
}
```

Each rule that begins with `identity:` and specifies a protected controller that manages an API. We use the already established targets to protect these new targets. The table below outlines which target protects which API. The full 1:1 mapping for this information can be found in Keystone's documentation.

Table 1-2. A brief mapping of policy targets and APIs

Policy Target	API
<code>identity:list_projects</code>	GET /v3/projects
<code>identity:create_project</code>	POST /v3/projects
<code>identity:delete_project</code>	DELETE /v3/projects/{project_id}
<code>identity:list_user_projects</code>	GET /v3/users/{user_id}/projects

It's important to note the subtle difference between listing all projects and listing a user's projects. Listing all projects should be an admin-only operation, and its policy is written as such. Listing all projects a user has access to should not only be restricted to admins, but also to the user that has a role on that project.

1.5 Backends and Services

It's worth summarizing backends and services into a handy picture. The green portions indicate backends that are usually SQL, the pink portions indicate backends that are usually LDAP or SQL, the blue indicates SQL or Memcache, and finally, policy is serviced from a file. There are other Keystone services; however, these are by far the most commonly used.

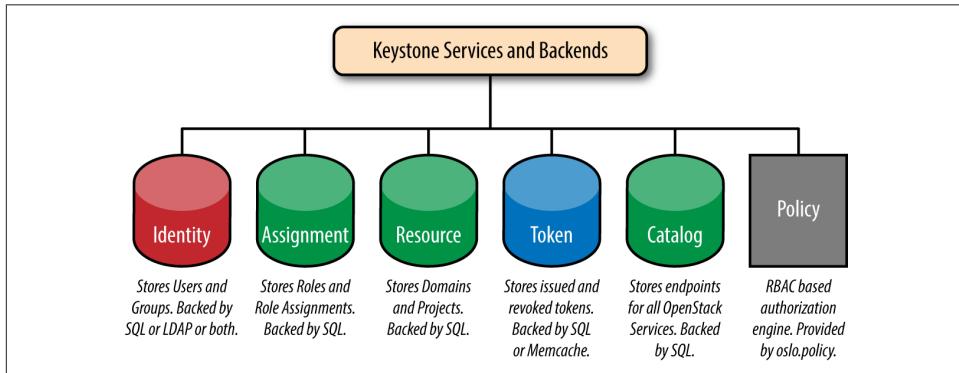


Figure 1-6. An overview of the different services and backends Keystone supports

1.6 FAQs

What's the difference between a Domain and a Region?

These terms are often used interchangeably in the OpenStack world, but are very different concepts. A region is intended to be a geographical representation: an enterprise may have a US-West region and a US-East region. A domain is a logical split between ownership of projects and identity sources.

Can users exist in multiple domains?

In short, no. Each user is owned by a domain. A major point of confusion is that user names may be the same across domains. Meaning there can be a *stevemar* at the IBM domain and a *stevemar* at the Acme Inc. domain. Each would have their own unique ID because each *stevemar* is owned by a different domain, and is considered by Keystone to be a different user.

Why does OpenStack documentation sometimes mention tenants and other times mention projects? Why mention both? Is there a difference?

In the early days of OpenStack, the term *tenant* was used by OpenStack services to be the entity for grouping and isolating resources. Over time it was decided that *project* was a more intuitive term for this. Unfortunately, not all of the projects have fully migrated over from tenant to project. Therefore, in some OpenStack services and documentation you may still run into the term *tenant*.

What is the difference between a scoped and unscoped token?

An *unscoped* token is one where the user is authenticated but not for a specific project or domain. This type of token is useful for making queries such as determining what projects a user has access to. A *scoped* token is created when the user is authenticated for a specific project or domain. Scoped tokens have role information associated with them and are the types of tokens used by the other OpenStack services to determine what types of operations are permitted.

I just authenticated and received a token back but the token doesn't have any roles in it. Is this a bug? Why are there no roles in the token?

If this happens it is because a user has not submitted a scope in their authentication request. As such, the user gets back an unscoped token, which will not contain any roles, since it does not carry any authorization on a project. So until you authenticate and specify a scope (project or domain) you will not see role information in the token.

What about Active Directory support?

It's there! Keystone's LDAP support extends to Active Directory. In its earlier releases, there were bugs in the LDAP backend that were related to AD, but since then multiple enterprises have been able to configure Keystone to communicate with their AD.

What are the other authentication methods?

Keystone also provides support for authenticating through a certificate. The only caveat here is that the user name associated with the certificate must exist in an Identity backend.

Let's Use Keystone!

In this chapter we will explain how to use Keystone in a development environment. This involves a few steps. First, we deploy OpenStack with DevStack, then try basic Keystone operations with OpenStackClient (a command line interface), then we perform the same Keystone operations with Horizon (a Web interface). We will also be providing cURL alternatives to the OpenStackClient commands to illustrate the fact that the CLI is simply just a wrapper for a REST call.

2.1 Getting DevStack

The examples shown below were performed on a new Ubuntu 64-bit virtual machine (VM). There are several options available for quickly getting an Ubuntu VM up and running, such as VMWare Fusion or Oracle's VirtualBox.

To begin, we need to install git and curl, then clone the DevStack repository. It is best to perform these steps from your user's home directory.

```
$ sudo apt-get install git curl  
$ git clone https://github.com/openstack-dev/devstack/  
$ cd devstack
```

Within the *devstack* folder, we create a file called *local.conf*. The settings in this file tell DevStack which components of OpenStack to install and provide a few minor configuration details. We'll be installing Keystone, Nova, Glance, Cinder, and Horizon. We'll also be using the latest development branch of OpenStack.

Example 2-1. The local.conf file contents

```
[[local|localrc]]  
RECLONE=yes
```

```

# Credentials
DATABASE_PASSWORD=openstack
ADMIN_PASSWORD=openstack
SERVICE_PASSWORD=openstack
SERVICE_TOKEN=openstack
RABBIT_PASSWORD=openstack

# Services
ENABLED_SERVICES=rabbit,mysql,key
ENABLED_SERVICES+=,horizon
ENABLED_SERVICES+=,n-api,n-crt,n-obj,n-cpu,n-cond,n-sch,n-novnc,n-cauth
ENABLED_SERVICES+=,n-net
ENABLED_SERVICES+=,g-api,g-reg
ENABLED_SERVICES+=,cinder,c-api,c-vol,c-sch,c-bak

# Enable Logging
LOGFILE=/opt/stack/logs/stack.sh.log
VERBOSE=True
LOG_COLOR=True
SCREEN_LOGDIR=/opt/stack/logs

```

The last step in the process is to kick off the `stack.sh` script and grab a coffee—the whole process will take around 25 minutes for the initial setup.

```
$ ./stack.sh
```

Once completed, the output looks like:

```

This is your host IP address: 10.0.2.15
This is your host IPv6 address: ::1
Horizon is now available at http://10.0.2.15/
Keystone is serving at http://10.0.2.15:5000/
The default users are: admin and demo
The password: openstack

```

Congratulations, you've installed OpenStack!

2.2 Basic Keystone Operations Using OpenStackClient

In this section we will use OpenStackClient to perform basic Keystone operations such as authenticating, listing, and creating users, projects, domains, groups, and roles. For illustrative purposes we also include the cURL alternative to the specific command to show the reader that OpenStackClient performs the equivalent REST call.

Before we start executing commands, we should create a few environment variables to make authenticating with our Keystone service easier. Note that most of these values should be the same for your deployment, but the hostname or IP address of your VM may be different.

```
$ export OS_IDENTITY_API_VERSION=3
$ export OS_AUTH_URL=http://10.0.2.15:5000/v3
$ export OS_USERNAME=admin
$ export OS_PROJECT_NAME=admin
$ export OS_USER_DOMAIN_NAME=Default
$ export OS_PASSWORD=openstack
$ export OS_PROJECT_DOMAIN_NAME=Default
```

To check that the environment variables are set, perform the following and analyze the output:

```
$ env | grep OS
OS_IDENTITY_API_VERSION=3
OS_AUTH_URL=http://10.0.2.15:5000/v3
OS_USERNAME=admin
OS_PROJECT_NAME=admin
OS_USER_DOMAIN_NAME=Default
OS_PASSWORD=openstack
OS_PROJECT_DOMAIN_NAME=Default
```

2.2.1 Getting a Token

Using OpenStackClient

Since we already set authentication and authorization data as environment variables we simply perform the following to issue a token:

```
$ openstack token issue
+-----+-----+
| Field      | Value           |
+-----+-----+
| expires    | 2015-08-27T21:45:41.712853Z |
| id         | d219ca63fd2548f685fea48623b22a10 |
| project_id | 92841d1c386643a08c697c833ed840af |
| user_id    | 82d7e61f128b4e398bb165f278f45569 |
+-----+-----+
```

Using cURL

When using cURL to obtain a token, the payload for the authentication request must include information about the user and the project.

```
$ curl -i -H "Content-Type: application/json" -d '
{ "auth": {
    "identity": {
        "methods": ["password"],
        "password": {
            "user": {
                "name": "admin",
                "domain": { "name": "Default" },
                "password": "openstack"
            }
        }
    }
}}
```

```

        }
    },
    "scope": {
        "project": {
            "name": "admin",
            "domain": { "name": "Default" }
        }
    }
}
}' http://localhost:5000/v3/auth/tokens
```

HTTP/1.1 201 Created
Date: Thu, 27 Aug 2015 21:57:56 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Subject-Token: 2511aaa898ff42158addea8c90ba2622
Vary: X-Auth-Token
x-openstack-request-id: req-ab469b0e-6990-4cca-9b1a-98780ae4108a
Content-Length: 5276
Content-Type: application/json

```
{"token": {"methods": ["password"], "roles": [{"id": "2d357899bd5c430988772eb861fb7a68", "name": "admin"}], "expires_at": "2015-08-27T22:57:58.249918Z", "project": {"domain": {"id": "default", "name": "Default"}, "id": "ed50269971bc41ef9f25ce2c7e9b9d11", "name": "admin"}, "catalog": [{"endpoints": [{"region_id": "RegionOne", "url": "http://10.0.2.15:35357/v2.0", "region": "RegionOne"}, {"interface": "admin", "id": "19dfab5f51b24dbe8fc0e7d5f4677a7"}, {"region_id": "RegionOne", "url": "http://10.0.2.15:5000/v2.0", "region": "RegionOne"}, {"interface": "internal", "id": "763fef84e9bd43b58733813d5b1f29bb"}, {"region_id": "RegionOne", "url": "http://10.0.2.15:5000/v2.0", "region": "RegionOne"}, {"interface": "public", "id": "d210b05177b24733b40146f7dcadb90"}]}, "type": "identity", "id": "8ae94fc3bb534ef2ae5bc16979e28eaf", "name": "keystone"}], "extras": {}, "user": {"domain": {"id": "default", "name": "Default"}, "id": "9644330bc62542c491179895c6a6d228", "name": "admin"}, "audit_ids": ["2h1Ass48TvKnOa4u3LR3HA"], "issued_at": "2015-08-27T21:57:58.249944Z"}}
```



The actual response was trimmed down, since it included the full catalog with all endpoints. For readability, we opted to remove most endpoints in the catalog.

The Token value is set in the X-Subject-Token response header. For the next few examples we will show both CLI and cURL examples. To make the cURL examples easier, set OS_TOKEN to the value seen in X-Subject-Token.

```
$ OS_TOKEN=2511aaa898ff42158addea8c90ba2622
```

2.2.2 Listing Users

Using OpenStackClient

After running DevStack, several users will be created automatically. Generally, these are service accounts for other OpenStack services (Cinder, Glance, and Nova), an administrator account (admin), and a non-administrator account (demo).

```
$ openstack user list
+-----+-----+
| ID      | Name   |
+-----+-----+
| 05a77e13219949c59368b99047b6be4b | cinder |
| 4da7bd7e25f34ea4aca792055f715fb8 | admin   |
| 8e30d2d495eb467b8b673fdbfb8be6c7 | glance  |
| ae760d23927b467e910146e4e9f400c0 | nova    |
| f6ddb1f6568942cbb85fdbf441c45c05 | demo    |
+-----+-----+
```

Using cURL

The alternative would be to use a token and reach the correct API.

```
$ curl -s -H "X-Auth-Token: $OS_TOKEN" \
       http://localhost:5000/v3/users | python -mjson.tool
{
    "links": {
        "next": null,
        "previous": null,
        "self": "http://localhost:5000/v3/users"
    },
    "users": [
        {
            "domain_id": "default",
            "enabled": true,
            "id": "0aa473010af04acb86018125ef71a65e",
            "links": {
                "self": "http://localhost:5000/v3/users/0aa473010a...125ef71a65e"
            },
            "name": "glance"
        },
        {
            "domain_id": "default",
            "enabled": true,
            "id": "9644330bc62542c491179895c6a6d228",
            "links": {
                "self": "http://localhost:5000/v3/users/96443...91179895c6a6d228"
            },
            "name": "admin"
        },
        {
            "domain_id": "default",
            "enabled": true,
```

```

        "enabled": true,
        "id": "ab7ab1f3fe2745fd91b3d62fd3b7309b",
        "links": {
            "self": "http://localhost:5000/v3/users/ab7a...d91b3d62fd3b7309b"
        },
        "name": "cinder"
    },
    {
        "domain_id": "default",
        "email": "demo@example.com",
        "enabled": true,
        "id": "c88b8fa1414e44be918e9e129f04147d",
        "links": {
            "self": "http://localhost:5000/v3/users/c88b8fa...8e9e129f04147d"
        },
        "name": "demo"
    },
    {
        "domain_id": "default",
        "enabled": true,
        "id": "cf15631f360c4db18fc95fe95da827f2",
        "links": {
            "self": "http://localhost:5000/v3/users/cf1563...fc95fe95da827f2"
        },
        "name": "nova"
    }
]
}

```

2.2.3 Listing Projects

Using OpenStackClient

Similar to users, DevStack also creates several projects by default. They can be seen by issuing the following command:

```
$ openstack project list
+-----+-----+
| ID      | Name   |
+-----+-----+
| 10bc96bf62c14d44b02bb5ad8aef57d3 | admin  |
| 241d8b116a164bcb9c63d75117ed3894 | demo   |
| 81dc6de893924fbfff16f272bdfba38d | invisible_to_admin |
| ebad3ef327c143bb8e79a52b9b0324e8 | service |
+-----+-----+
```

Using cURL

Again, similar to the users API, a token is required and we need to reach the proper endpoint.

```

$ curl -s -H "X-Auth-Token: $OS_TOKEN" \
      http://localhost:5000/v3/projects | python -mjson.tool
{
    "links": {
        "next": null,
        "previous": null,
        "self": "http://localhost:5000/v3/projects"
    },
    "projects": [
        {
            "description": "",
            "domain_id": "default",
            "enabled": true,
            "id": "5f198c74b0984566b0a6da26440c5f85",
            "is_domain": false,
            "links": {
                "self": "http://localhost:5000/v3/projects/5f1...0a6da26440c5f85"
            },
            "name": "demo",
            "parent_id": null
        },
        {
            "description": "",
            "domain_id": "default",
            "enabled": true,
            "id": "9457bf6cb940453ab92a88219a40a3f7",
            "is_domain": false,
            "links": {
                "self": "http://localhost:5000/v3/projects/9457bf6cb9...9a40a3f7"
            },
            "name": "service",
            "parent_id": null
        },
        {
            "description": "",
            "domain_id": "default",
            "enabled": true,
            "id": "dcd882c878bd43188f9758831676559a",
            "is_domain": false,
            "links": {
                "self": "http://localhost:5000/v3/projects/dcd882c878...1676559a"
            },
            "name": "invisible_to_admin",
            "parent_id": null
        },
        {
            "description": "",
            "domain_id": "default",
            "enabled": true,
            "id": "ed50269971bc41ef9f25ce2c7e9b9d11",
            "is_domain": false,
            "links": {

```

```

        "self": "http://localhost:5000/v3/projects/ed5026997...c7e9b9d11"
    },
    "name": "admin",
    "parent_id": null
}
]
}

```

2.2.4 Listing Groups

Using OpenStackClient

Similar to users, DevStack also creates two groups by default. Note that the users are not part of either group yet.

```
$ openstack group list
+-----+-----+
| ID      | Name   |
+-----+-----+
| a95873d6a7f54dae90e53de3044b0964 | nonadmins |
| e27d8f1c492441888ea4b0a7e836a835 | admins    |
+-----+-----+
```

Using cURL

As with the other commands, simply reuse the token and reach the proper endpoint.

```
$ curl -s -H "X-Auth-Token: $OS_TOKEN" \
    http://localhost:5000/v3/groups | python -mjson.tool
{
    "groups": [
        {
            "description": "openstack admin group",
            "domain_id": "default",
            "id": "5553cf58254746c8ba615b671dc5b2",
            "links": {
                "self": "http://localhost:5000/v3/groups/5553cf582...1dc5b2"
            },
            "name": "admins"
        },
        {
            "description": "non-admin group",
            "domain_id": "default",
            "id": "8a2e550b2edc4f76bae608e578becf5c",
            "links": {
                "self": "http://localhost:5000/v3/groups/8a2e550b2...8e578becf5c"
            },
            "name": "nonadmins"
        }
    ],
    "links": {
        "next": null,
    }
}
```

```

        "previous": null,
        "self": "http://localhost:5000/v3/groups"
    }
}

```

2.2.5 Listing Roles

Using OpenStackClient

In a fashion similar to the previous commands, DevStack also creates a number of roles by default and assigns these roles to various users across the projects.

```
$ openstack role list
+-----+-----+
| ID      | Name   |
+-----+-----+
| 03bdd6d46c2c4f99818bc855872a909e | service |
| 4684cf02622f49dd82458852c62f4135 | Member  |
| 8a5dbcef99274f688effa338db6bf928 | anotherrole |
| a22b737940ed4e7588639f1bcb3e3afa | admin   |
| b5e5224549574edf9ff2caebbd9431d | ResellerAdmin |
+-----+-----+
```

Using cURL

Again, use a token and hit the correct endpoint.

```
$ curl -s -H "X-Auth-Token: $OS_TOKEN" \
      http://localhost:5000/v3/roles | python -mjson.tool
{
    "links": {
        "next": null,
        "previous": null,
        "self": "http://localhost:5000/v3/roles"
    },
    "roles": [
        {
            "id": "20cd9e77e2d44d3490d8c044f508c8f3",
            "links": {
                "self": "http://localhost:5000/v3/roles/20cd9e77e...044f508c8f3"
            },
            "name": "service"
        },
        {
            "id": "2d357899bd5c430988772eb861fb7a68",
            "links": {
                "self": "http://localhost:5000/v3/roles/2d357899b...eb861fb7a68"
            },
            "name": "admin"
        },
        {
            "id": "40df936c87824fd98b7e39da44f2dbde",
            "links": {
                "self": "http://localhost:5000/v3/roles/40df936c...da44f2dbde"
            },
            "name": "ResellerAdmin"
        }
    ]
}
```

```

    "links": {
        "self": "http://localhost:5000/v3/roles/40df936c8...9da44f2dbde"
    },
    "name": "Member"
},
{
    "id": "d1e7fed5471941b9aeebf15a341edef9",
    "links": {
        "self": "http://localhost:5000/v3/roles/d1e7fed54...15a341edef9"
    },
    "name": "ResellerAdmin"
},
{
    "id": "face2e488b1c4562bc9f43f8a5e143db",
    "links": {
        "self": "http://localhost:5000/v3/roles/face2e488...3f8a5e143db"
    },
    "name": "anotherrole"
}
]
}

```

2.2.6 Listing Domains

Using OpenStackClient

Keystone automatically has a single domain when it is brought up; this is to handle backwards compatibility between Identity API versions.

```
$ openstack domain list
+-----+-----+-----+
| ID      | Name     | Enabled | Description
+-----+-----+-----+
| default | Default  | True    | Owns users and projects for Identity API v2. |
+-----+-----+-----+
```

Using cURL

To list domains, use a valid token and hit the correct endpoint.

```
$ curl -s -H "X-Auth-Token: $OS_TOKEN" \
    http://localhost:5000/v3/domains | python -mjson.tool
{
    "domains": [
        {
            "description": "Owns users and projects for Identity API v2.",
            "enabled": true,
            "id": "default",
            "links": {
                "self": "http://localhost:5000/v3/domains/default"
            },
            "name": "Default"
        }
    ]
}
```

```

        }
    ],
    "links": {
        "next": null,
        "previous": null,
        "self": "http://localhost:5000/v3/domains"
    }
}

```

2.2.7 Creating Another Domain

Using OpenStackClient

To get a better sense of v3 of the Identity API, let's create a domain, and a project and user to be owned by that domain.

```
$ openstack domain create acme
+-----+
| Field | Value |
+-----+
| enabled | True |
| id | adf547d21ae148aa81c77b36b611d1c3 |
| name | acme |
+-----+
```

Using cURL

Here, we use a POST request with the minimal amount of information in the payload, just the domain name, and by default it'll be enabled.

```
$ curl -s -H "X-Auth-Token: $OS_TOKEN" \
-H "Content-Type: application/json" -d '{ "domain": { "name": "acme"} }'
http://localhost:5000/v3/domains | python -mjson.tool
{
    "domain": {
        "enabled": true,
        "id": "05a0fcfb796142bfbeb30d0fd3dfa67a",
        "links": [
            "self": "http://localhost:5000/v3/domains/05a0fcfb796142b...d3dfa67a"
        ],
        "name": "acme"
    }
}
```

2.2.8 Create a Project within the Domain

Using OpenStackClient

To create a project within the domain, we need to specify the domain we created in the previous step, and preferably, give it a fun description.

```
$ openstack project create tims_project \
--domain acme \
--description "tims dev project"
+-----+
| Field      | Value           |
+-----+
| description | tims dev project |
| domain_id   | adf547d21ae148aa81c77b36b611d1c3 |
| enabled     | True            |
| id          | 77941c6a73ed452eb65d36d7c962201c |
| is_domain   | False           |
| name        | tims_project    |
| parent_id   | None            |
+-----+
```

Using cURL

Along with the name of the project, the payload of the POST request must also specify the domain ID, which can be found in the previous step. The description of the project is optional.

```
$ curl -s -H "X-Auth-Token: $OS_TOKEN" \
-H "Content-Type: application/json" \
-d '{ "project": { "name": "tims_project", \
"domain_id": "05a0fcfb796142bfbeb30d0fd3dfa67a", \
"description": "tims dev project"} }' \
http://localhost:5000/v3/projects | python -mjson.tool
{
  "project": {
    "description": "tims dev project",
    "domain_id": "05a0fcfb796142bfbeb30d0fd3dfa67a",
    "enabled": true,
    "id": "26436cea94ac4a40b890f05434f15aa4",
    "is_domain": false,
    "links": [
      "self": "http://localhost:5000/v3/projects/26436cea94ac4...434f15aa4"
    ],
    "name": "tims_project",
    "parent_id": null
  }
}
```

2.2.9 Create a User within the Domain

Using OpenStackClient

To create a user within the domain, we need to specify the domain we created in the previous step. Setting a password and email for the user is optional, but encouraged.

```
$ openstack user create tim --email tim@tim.ca \
--domain acme --description "tims openstack user account" \
--password s3cr3t
```

Field	Value
description	tims openstack user account
domain_id	adf547d21ae148aa81c77b36b611d1c3
email	tim@tim.ca
enabled	True
id	a5db8e8e6a994e05afe94aa21c8c4ec8
name	tim

Using cURL

Similar to the previous cURL commands, the payload has to be structured correctly.

```
$ curl -s -H "X-Auth-Token: $OS_TOKEN" \
      -H "Content-Type: application/json" \
      -d '{ "user": { "name": "tim", "password": "s3cr3t", \
      "email": "tim@tim.ca", "domain_id": \
      "05a0fcbf796142bfbeb30d0fd3dfa67a", "description": \
      "tims openstack user account"} }' http://localhost:5000/v3/users \
      | python -mjson.tool
{
  "user": {
    "description": "tims openstack user account",
    "domain_id": "05a0fcbf796142bfbeb30d0fd3dfa67a",
    "email": "tim@tim.ca",
    "enabled": true,
    "id": "6eced805da4649198dd1dd3bdddadd68",
    "links": [
      "self": "http://localhost:5000/v3/users/6eced805da4649198dd...dadd68"
    ],
    "name": "tim"
  }
}
```

2.2.10 Assigning a Role to a User for a Project

Using OpenStackClient

To assign a role to the new user on the new project, we can use the CLI, but both the user and the project must be qualified with the right domain, or OpenStackClient will use the default domain.

```
$ openstack role add member --project tims_project --project-domain acme \
      --user tim --user-domain acme
```

Using cURL

The API for assigning a role is different than the previous commands; it uses PUT instead of POST, and only accepts IDs for the user, project and role.

```
$ curl -s -X PUT -H "X-Auth-Token: $OS_TOKEN" \
      http://localhost:5000/v3/projects/26436cea94ac4a40b890f05434f15aa4/users/6ece
      d805da4649198dd1dd3bddadd68/roles/40df936c87824fd98b7e39da44f2dbde
```

2.2.11 Authenticating as the New User

Using OpenStackClient

To authenticate as the new user, it's best to start a new terminal session and create new environment variables. In this case, the user name, password, project, and domain information must all be set accordingly.

```
$ export OS_PASSWORD=s3cr3t
$ export OS_IDENTITY_API_VERSION=3
$ export OS_AUTH_URL=http://10.0.2.15:5000/v3
$ export OS_USERNAME=tim
$ export OS_PROJECT_NAME=tims_project
$ export OS_USER_DOMAIN_NAME=acme
$ export OS_PROJECT_DOMAIN_NAME=acme
```

Once that is set, we can try to retrieve a token to ensure our user is able to authenticate.

```
$ openstack token issue
+-----+-----+
| Field | Value |
+-----+-----+
| expires | 2015-08-22T05:26:02.385430Z |
| id | 59c60758f9f947a58289900c1f87b700 |
| project_id | 77941c6a73ed452eb65d36d7c962201c |
| user_id | 26898b5d27114f90a1f57c5333e3e9b4 |
+-----+-----+
```

Using cURL

Similar to the authentication request of “[2.2.1 Getting a Token](#)” on page 19, the structure of the payload remains the same, but the values must be updated. Again, for readability, we remove several of the endpoints from the service catalog.

```
$ curl -i -H "Content-Type: application/json" -d '
  "identity": {
    "methods": ["password"],
    "password": {
      "user": {
        "name": "tim",
        "domain": { "name": "acme" },
        "password": "s3cr3t"
      }
    }
  },
  "scope": {
    "project": {
```

```

        "name": "tims_project",
        "domain": { "name": "acme" }
    }
}
}
}' http://localhost:5000/v3/auth/tokens
HTTP/1.1 201 Created
Date: Thu, 27 Aug 2015 22:47:01 GMT
Server: Apache/2.4.7 (Ubuntu)
X-Subject-Token: 1bce8850bc6c4b9e8e90b6822885d93c
Vary: X-Auth-Token
x-openstack-request-id: req-a71ec134-60d8-4f86-8546-00877a48151f
Content-Length: 5326
Content-Type: application/json

{"token": {"methods": ["password"], "roles": [{"id": "40df936c87824fd98b7e39da44f2dbde", "name": "Member"}], "expires_at": "2015-08-27T23:47:01.322257Z", "project": {"domain": {"id": "05a0fcfb796142bfbeb30d0fd3dfa67a", "name": "acme"}, "id": "26436cea94ac4a40b890f05434f15aa4", "name": "tims_project"}, "catalog": [{"endpoints": [{"region_id": "RegionOne", "url": "http://10.0.2.15:35357/v2.0", "region": "RegionOne", "interface": "admin", "id": "19dfab5f51b24dbe8cfcc0e7d5f4677a7"}, {"region_id": "RegionOne", "url": "http://10.0.2.15:5000/v2.0", "region": "RegionOne", "interface": "internal", "id": "763fef84e9bd43b58733813d5b1f29bb"}, {"region_id": "RegionOne", "url": "http://10.0.2.15:5000/v2.0", "region": "RegionOne", "interface": "public", "id": "d210b05177b24733b40146f7dcfad90"}], "type": "identity", "id": "8ae94fc3bb534ef2ae5bc16979e28eaf", "name": "keystone"}, {"extras": {}, "user": {"domain": {"id": "05a0fcfb796142bfbeb30d0fd3dfa67a", "name": "acme"}, "id": "6eced805da4649198dd1dd3bddadd68", "name": "tim"}, "audit_ids": ["rWvgwqBWSjq3sZgGWFNmVg"], "issued_at": "2015-08-27T22:47:01.322285Z"}]}

```

There are many, many more commands a user or administrator can perform; we suggest using `openstack --help` to find additional commands, or looking at OpenStack-Client's documentation.

2.3 Basic Keystone Operations Using Horizon

2.3.1 What Keystone Operations Are Available through Horizon?

There are various Keystone operations that are supported through Horizon, OpenStack's Dashboard. However, depending on the version enabled in Horizon's configuration file, there will be some differences. If v2 of the Identity API is enabled, then only User and Project CRUD support will be available, and users will only be able to authenticate against a single domain. If v3 of the Identity API is enabled, then User, Group, Project, Domain, and Role CRUD support will be made available, and users will be able to authenticate against multiple domains.

2.3.2 Accessing the Identity Operations

The Identity operations will appear in their own menu on the left-hand accordion.

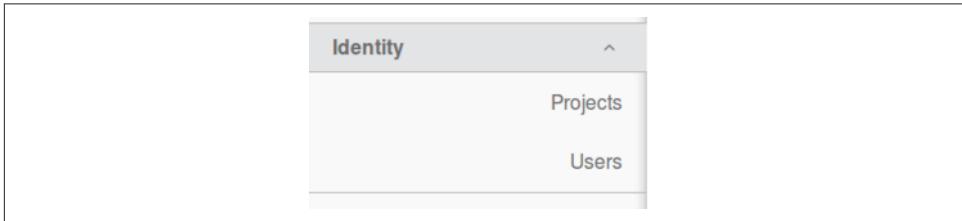


Figure 2-1. A screenshot of the Identity menu in Horizon, with v2 of the Identity API enabled

2.3.3 List, Set, Delete, Create, and View a Project

From Horizon, a user will be able to perform the same project-based command line operations.

Projects					
<input type="text" value="Filter"/> Q + Create Project x Delete Projects					
<input type="checkbox"/>	Name	Description	Project ID	Enabled	Actions
<input type="checkbox"/>	invisible_to_admin	-	3f04daa2aea54c8fb3489dbc88fc1408	Yes	Manage Members ▾
<input type="checkbox"/>	alt_demo	-	5663c22939834088b582c9e3ba0ef718	Yes	Manage Members ▾
<input type="checkbox"/>	service	-	7a34eb60461d487b9401d79af81b1fa6	Yes	Manage Members ▾
<input type="checkbox"/>	admin	-	a194c87013374986bc当地1012c870a1a0	Yes	Manage Members ▾
<input type="checkbox"/>	demo	-	cd17ac420e434609ad17d3c3df964783	Yes	Manage Members ▾

Figure 2-2. A screenshot of the Project pane in Horizon

2.3.4 List, Set, Delete, Create, and View a User

From Horizon, a user will be able to perform the same user-based command line operations.

Users					
	User Name	Email	User ID	Enabled	Actions
<input type="checkbox"/>	glance		2940afb2148149b49bc8999dfb358321	Yes	<button>Edit</button> <button>▼</button>
<input type="checkbox"/>	alt_demo	alt_demo@example.com	2a043ce582fb4320a9c6f5321e3b5c3b	Yes	<button>Edit</button> <button>▼</button>
<input type="checkbox"/>	demo	demo@example.com	2dbd498e7f4340058d578b97d0b78ecf	Yes	<button>Edit</button> <button>▼</button>
<input type="checkbox"/>	admin		a22aa9bac3454b45a23fab6efb950bb3	Yes	<button>Edit</button> <button>▼</button>
<input type="checkbox"/>	cinder		c4df212cb7554df6b3220d3cd641f652	Yes	<button>Edit</button> <button>▼</button>
<input type="checkbox"/>	nova		d3c7403c17544f51b2485fe09ee732fd	Yes	<button>Edit</button> <button>▼</button>

Displaying 6 items

Figure 2-3. A screenshot of the User pane in Horizon

2.4 Tips, Common Pitfalls, and Troubleshooting

Check Your Scope: A Common Authentication Problem

Continuing from our previous example, with the non-admin account, suppose we incorrectly set the project’s domain. Let’s try to authenticate with that information and see what happens:

```
$ export OS_PROJECT_DOMAIN_NAME=Default
$ openstack token issue
ERROR: openstack Could not find project: tims_project (Disable debug mode to suppress these details.) (HTTP 401) (Request-ID: req-3aa6c248-325f-4f10-bc27-b4d85a2c338b)
```

This error indicates that Keystone could not find the project in the specified domain. That’s because we created the project in the “acme” domain, and no project named “acme” exists in the default domain, yet. Using our admin account, let’s create a project with the same name in the default domain.

```
$ openstack project create tims_project
+-----+
| Field      | Value           |
+-----+
| description |                |
| domain_id   | default         |
| enabled     | True            |
| id          | 07dd8005d0d94be7ac6cb1ccb181f436 |
| is_domain   | False           |
| name        | tims_project    |
```

```
| parent_id | None |  
+-----+-----+
```

Now, going back to the non-admin account, let's try to authenticate again.

```
$ openstack token issue  
ERROR: openstack User 26898b5d27114f90a1f57c5333e3e9b4 has no access to project  
07dd8005d0d94be7ac6cb1ccb181f436 (Disable debug mode to suppress these details.)  
(HTTP 401) (Request-ID: req-b4b3048a-cf7a-453e-a5af-8ae8120e8fbf)
```

As you can see, the message is very different. The output indicates the project was found, but the user does not have a role on the project. That's because we never added a role for the user on that project.

Check Your Policy and Role: A Common Authorization Problem

Let's return to the non-admin account and set the project domain back to the correct value and ensure we can still authenticate to get a token.

```
$ export OS_PROJECT_DOMAIN_NAME=acme  
$ openstack token issue  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| expires | 2015-08-22T06:16:14.191211Z |  
| id | 4be37a084ca145e187c22e58cd619a44 |  
| project_id | 77941c6a73ed452eb65d36d7c962201c |  
| user_id | 26898b5d27114f90a1f57c5333e3e9b4 |  
+-----+-----+
```

Let's try to perform an action a non-admin user can't perform, like listing all the users in a deployment.

```
$ openstack user list  
ERROR: openstack You are not authorized to perform the requested action: identity  
:list_users (Disable debug mode to suppress these details.) (HTTP 403) (Request-I  
D: req-48f059cb-58fb-44d5-b015-05223b2af50e)
```

As the error message suggests, this occurs because the policy for listing all users (by default) is set to the following (you may find this in *policy.json*):

```
"identity:list_users": "rule:admin_required"
```

Getting Additional Information

The `--debug` option for OpenStackClient is very handy; it allows the user to see a lot of information about the request and response. Simply add `--debug` to any OpenStackClient command to enable this option.

Token Formats

3.1 History of Keystone Token Formats

Keystone offers several token formats, and users may wonder why there are so many. To help with understanding why, we provide a brief history of how the Keystone token formats have evolved. In the early days, Keystone supported a UUID token. This token was a 32-character string bearer token used for authentication and authorization. The advantage of this token format was that the token was small and very easy to use, and it was simple enough to add to a cURL command. The disadvantage of this token is it did not carry with it enough information to locally do authorization. OpenStack services would always have to send this token back to the Keystone server to determine if an operation was authorized. This resulted in Keystone being pinged for any OpenStack action, and becoming a bottleneck for all of OpenStack.

In an attempt to address the issues encountered with UUID tokens, the Keystone team created a new token format called the PKI token. This token contained enough information to perform local authorization and also contained the service catalog in the token as well. In addition, the token was signed and services could cache the token and use it until it expired or was revoked. The PKI token resulted in less traffic to the Keystone server, but the disadvantage of this token was that they were huge. PKI tokens could easily be 8K in size, and this made them difficult to fit in HTTP headers. Many web servers could not handle an 8K entry in an HTTP header without reconfiguration. In addition, these tokens were more difficult to use in cURL commands and offered poor user experience. Each of these drawbacks could be mitigated, but each took extra effort. The Keystone team tried a variant of this token that was compressed. This token format was called PKIz. Unfortunately, the feedback from the OpenStack community was that this token format was still too big.

The Keystone team went back to the drawing board and came up with a new token format known as the Fernet token. The Fernet token was small (around 255 characters) but contained enough information in it to be used for local authorization. The token had another critical advantage. It contained enough information that Keystone did not have to store token data in a token database. One of the most annoying aspects of Keystone's early token formats was that they needed to be persisted in a database. The database would fill up and Keystone's performance would suffer. OpenStack operators were constantly having to flush the Keystone token database to keep their OpenStack environments running. The Fernet token did have the drawback that the symmetric keys used to sign the tokens needed to be distributed and rotated. The OpenStack operators needed to come up with a way to handle these issues. Nonetheless, the operators we have worked with have said that they'd much rather deal with the key distribution associated Fernet tokens than have to use the other token formats and be responsible for flushing the Keystone token database.

With so many token choices, you may be wondering which tokens are used by OpenStack operators. Fortunately, the OpenStack community performs [user surveys](#) to gain these types of insights. As shown in the chart below, the UUID token is currently the most popular format. Please note, however, that this survey was performed in the OpenStack Juno release time frame, and does not include Fernet tokens because they were not added until the Kilo release.

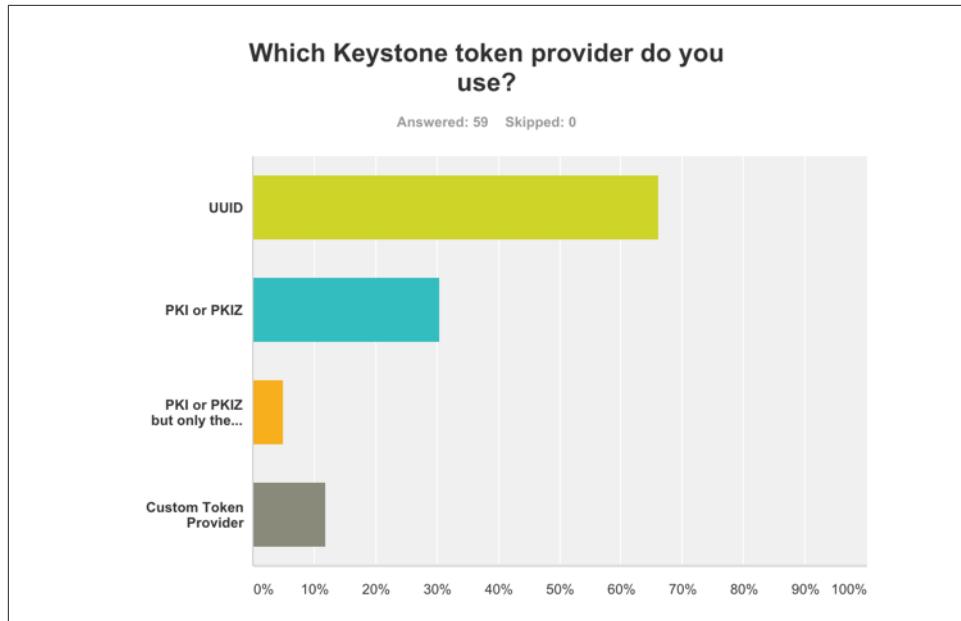


Figure 3-1. A survey conducted by the OpenStack User committee indicates that UUID tokens are still the preferred token format (as of Juno)

In this chapter, we describe Keystone’s UUID, PKI, and Fernet tokens in greater detail to provide the reader with a much better understanding of the token formats that are available to them in Keystone.

For all three token formats, it is important to note that all are bearer tokens. This means all three tokens need to be protected from unnecessary disclosure to prevent unauthorized access.

3.2 UUID Tokens

Keystone’s first token format was the UUID token format. The UUID token is simply a randomly generated UUID 32-character string. These are issued and validated by the identity service. A `hexdigest()` method is used, which ensures the tokens are made up of solely hexadecimal digits. This makes the tokens URL friendly and safe for transfer in any non-binary environment. A UUID token must be saved in a persistent backend (typically a database) in order to be available for subsequent validation. A UUID token can be revoked by simply issuing a DELETE request with the token ID. Note that the token is not actually removed from the backend, but rather marked as revoked. Since the token is only 32 characters, its size in an HTTP header is 32 bytes.

A typical UUID token would look like the following:

```
468da447bd1c4821bbc5def0498fd441.
```

The token is extremely small and easy to use when accessing Keystone through a cURL command. As previously mentioned, the disadvantage with this token format is that Keystone can become a bottleneck due to the tremendous amount of communication that occurs when Keystone is needed to validate the token.

The method for returning a UUID token is relatively simple. It is shown below and can be found [on GitHub](#).

```
def _get_token_id(self, token_data):
    return uuid.uuid4().hex
```

3.3 PKI Tokens

Keystone’s second supported format is the PKI token format. In the PKI format, the token contains the entire validation response that would be received from Keystone. What this means is that the token has a large amount of information in it, such as: when it was issued, when it expires, user identification, project, domain and role information for this user, a service catalog, and other information as well. All of this information is represented in a JSON document payload, and the payload is then signed using cryptographic message syntax (CMS). With the PKIz format, the signed payload is then compressed using zlib compression.

An example of what the JSON token payload would look like can be found in Keystone's documentation and is repeated here:

```
{  
    "token": {  
        "domain": {  
            "id": "default",  
            "name": "Default"  
        },  
        "methods": [  
            "password"  
        ],  
        "roles": [  
            {  
                "id": "c703057be878458588961ce9a0ce686b",  
                "name": "admin"  
            }  
        ],  
        "expires_at": "2014-06-10T21:52:58.852167Z",  
        "catalog": [  
            {  
                "endpoints": [  
                    {  
                        "url": "http://localhost:35357/v2.0",  
                        "region": "RegionOne",  
                        "interface": "admin",  
                        "id": "29beb2f1567642eb810b042b6719ea88"  
                    },  
                    {  
                        "url": "http://localhost:5000/v2.0",  
                        "region": "RegionOne",  
                        "interface": "internal",  
                        "id": "87057e3735d4415c97ae231b4841eb1c"  
                    },  
                    {  
                        "url": "http://localhost:5000/v2.0",  
                        "region": "RegionOne",  
                        "interface": "public",  
                        "id": "ef303187fc8d41668f25199c298396a5"  
                    }  
                ],  
                "type": "identity",  
                "id": "bd7397d2c0e14fb69bae8ff76e112a90",  
                "name": "keystone"  
            }  
        ],  
        "extras": {},  
        "user": {  
            "domain": {  
                "id": "default",  
                "name": "Default"  
            },  
            "meta": {}  
        }  
    }  
}
```

```

        "id": "3ec3164f750146be97f21559ee4d9c51",
        "name": "admin"
    },
    "audit_ids": [
        "Xpa6Uyn-T9S6mTREudUH3w"
    ],
    "issued_at": "2014-06-10T20:52:58.852194Z"
}
}

```

As shown in the JSON above, the token has everything a service needs with respect to user, domain, project, and role information to determine if the user is authorized for a particular OpenStack service operation. The service is thus able to cache this token and make authorization decisions locally without having to contact the Keystone server on every authorization request. In order to transport the token via HTTP, the signed JSON document is base64 encoded with some minor modifications. An example of what the token would look like in transport is shown here in an abbreviated form:

```
MIDsAYCCAokGCSqGSIB3DQEHAaCCAnoEggJ2ew0KICAgICJhY2QogICAgICAgI...EBMFwwVzELMAKGA
1UEBhMCVVMxDjAMBgNVBAgTBVVuc2V0MCoIIDoTCCA50CAQExCTAHBgUrDgMQ4wDAYDVQQHEwVVbnNldD
E0MAwGA1UEChM7r0iosFscpnfCuc8jGMobyfApz/dZqJnsk4lt1ahlNtpXQeVFxNK/ydKL+tzEjg
```

Note that even a basic token with a single endpoint in the catalog approaches a size of approximately 1,700 bytes. With larger deployments with several endpoints and services, the size of a PKI token quickly exceeds that of a default HTTP header limit for most web servers, which is 8KB. Even when compressed, the size of PKI tokens does not decrease enough to mitigate these problems. In practice, the resultant token is approximately 10% smaller.

Although PKI and PKIz tokens can be cached, they have several disadvantages. It can be difficult to configure Keystone to use this type of token, as they really should use certificates issued by a trusted certificate authority. Moreover, they are extremely large and their size may cause other OpenStack services to have issues. Their size can also break web performance tools, and they are quite difficult to use in a cURL command. In addition, in practice the Keystone service typically must persist these tokens in its persistence backend for purposes such as creating revocation lists. As a result the user must continue to worry about flushing the Keystone token database frequently or performance will suffer.

Once the difficult part of having all the certificates created is complete, generating PKI tokens is not that complex. As an illustration, the key aspects of the method for returning a PKI token are shown below.¹

¹ <https://github.com/openstack/keystone/blob/master/keystone/token/providers/pki.py>

```

def _get_token_id(self, token_data):
    try:
        token_json = jsonutils.dumps(token_data, cls=utils.PKIEncoder)
        token_id = str(cms.cms_sign_token(token_json,
                                         CONF.signing.certfile,
                                         CONF.signing.keyfile))
    return token_id
    except environment.subprocess.CalledProcessError:
        LOG.exception(_LE('Unable to sign token'))
        raise exception.UnexpectedError(_('Unable to sign token.'))

```

3.4 Fernet Tokens

The newest Keystone token format is the Fernet token format. The Fernet token attempts to improve on previous token formats in a variety of ways. First, it is quite small—it typically comes in around 255 characters, and as such are larger than UUID tokens, but significantly smaller than PKI.¹ The token also contains just enough information to enable the token to not have to be stored in a persistent Keystone token database. Instead, the token has enough information that the rest of the needed information such as the roles a user has on a project can be generated. In large-scale OpenStack deployments, having to store massive amounts of token data has been identified as a key contributor to performance degradation, not to mention the need to constantly flush the persistent token database.²

Fernet tokens contain a small amount of information, such as a user identifier, a project identifier, token expiration information, and other auditing information. Fernet tokens are signed using a symmetric key to prevent tampering. The use of a Fernet token follows the same workflow as a UUID token. Thus, in contrast to a PKI token, they must be validated by Keystone similar to how this must be done with UUID tokens.

One complicating factor is that Fernet tokens use symmetric keys to sign the token, and these keys need to be distributed to the various OpenStack regions. Additionally, these keys need to be rotated. Because Fernet tokens are being revised in the current release, we are hesitant to provide a large amount of detail on these tokens at this point in time. Instead, we refer the reader to some online sources we anticipate will be updated based on real-world experience using Fernet tokens and symmetric key rotation.³

¹ <http://dolphm.com/the-anatomy-of-openstack-keystone-token-formats/>

² See <https://review.openstack.org/#/c/130050/33/specs/kilo/klwt.rst>, <http://dolphm.com/openstack-keystone-fernet-tokens/>, and <http://dolphm.com/benchmarking-openstack-keystone-token-formats/>.

³ See <http://lbragstad.com/?p=133>, <http://www.mattfischer.com/blog/?p=648>, and <http://www.mattfischer.com/blog/?p=665>.

Keystone provides a configuration tool for setting up symmetric encryption keys. This can be performed by invoking the following command:

```
$ keystone-manage fernet_setup
```

In addition, the keys should be periodically rotated using the following command:

```
$ keystone-manage fernet_rotate
```

As an illustration, the key aspects of creating a project-scoped Fernet token is shown below. The example deviates from the source code to improve readability. The fundamental step to creating a Fernet token is to safely convert any data to bytes, then MessagePack the token contents, sign it, and make it URL safe.¹

```
def create_token(self, user_id, expires_at, audit_ids, methods=None,
                 domain_id=None, project_id=None, trust_id=None,
                 federated_info=None):

    b_user_id = cls.attempt_convert_uuid_hex_to_bytes(user_id)
    methods = auth_plugins.convert_method_list_to_integer(methods)
    b_project_id = cls.attempt_convert_uuid_hex_to_bytes(project_id)
    expires_at_int = cls._convert_time_string_to_int(expires_at)
    b_audit_ids = list(map(provider.random_urlsafe_str_to_bytes, audit_ids))

    payload = (b_user_id, methods, b_project_id, expires_at_int, b_audit_ids)
    serialized_payload = msgpack.packb(payload)
    token = urllib.parse.quote(self.crypto.encrypt(serialized_payload))
    return token
```

A typical Fernet token looks like:

```
gAAAAABU7roWGiCu0vgFcckec-0ytpGnMZDBLG9hA7Hr9qfvZDHjsak39YN98HXxoYLIqVm19Egku5YR
3wyI7heVr0mPNETmr-fIM1rtahudEdEAPM4HCiMrBmiA1Lw6SU8jc2rPLC7FK7nBCia_BGhG17NVHuQu0
S7waA306jyKNhHwUnpsBQ%3D
```

Fernet tokens are validated by simply performing the same steps to create the token, but in reverse. We begin by unquoting the URL-safe bits, decrypting the value, and unpacking the payload. From there, it's simply a matter of checking the different sections of the payload, specifically the `expires_at` field.

```
def validate_token(self, token):

    token = urllib.parse.unquote(six.binary_type(token))
    serialized_payload = self.crypto.decrypt(token)
    payload = msgpack.unpackb(serialized_payload)

    user_id = cls.attempt_convert_uuid_bytes_to_hex(payload[0])
    methods = auth_plugins.convert_integer_to_method_list(payload[1])
    project_id = cls.attempt_convert_uuid_bytes_to_hex(payload[2])
```

¹ https://github.com/openstack/keystone/blob/master/keystone/token/providers/fernet/token_formatters.py

```
expires_at_str = cls._convert_int_to_time_string(payload[3])
audit_ids = list(map(provider.base64_encode, payload[4]))

return (user_id, methods, project_id, expires_at_str, audit_ids)
```

3.5 Tips, Common Pitfalls, and Troubleshooting

In this section, we describe a few commonly experienced pitfalls that occur when using the different types of Keystone token formats and provide troubleshooting tips to address these issues.

3.5.1 UUID Token Performance Degradation for Authentication Operations

When using UUID tokens over long periods of time, the operator may begin to notice that the performance of Keystone authentication begins to degrade severely. CPU utilization will increase and response times will lag. This typically results from Keystone's persistent token backend, which has likely become extremely large due to storing many tokens. As a result, authentication requests, which require looking up the token ID in persistent storage, can take an extremely long time. If you experience this behavior, try running Keystone's utility for purging its database of old tokens:

```
$ keystone-manage token_flush
```

It is highly recommended that you run this command periodically. Best practices typically involve running it as a cron job.

3.5.2 Using PKI Token and Swift or Horizon Not Working?

Since the entire catalog is included in a PKI token, a large-scale deployment with many endpoints in its catalog could inflate the token size to a value greater than HTTP request header limits. This can be quite problematic for some OpenStack projects such as OpenStack Swift and Horizon. If this situation is encountered, there are several possible options to mitigate this issue.

First, you can try switching from PKI to PKIz. The compression that results from using PKIz may result in enough size reduction to get you back under the limit. This is accomplished by changing from the PKI token provider to the PKIz token provider. This configuration value can be found in the [token] section of the configuration file and can be set to PKIz by setting

```
[token]
provider = keystone.token.providers.pkiz.Provider
```

Another possible option is to increase the maximum header size of the web server, or Eventlet, or both, depending on the project being impacted. It should be noted that

changing this value is simply masking the poor performance, and raising a threshold is not a final solution.

Finally, Keystone has added an option that allows the catalog to not be included in the PKI token and some of the OpenStack projects such as Swift have configuration options that allow them to utilize this feature. Within the Swift configuration file in the `keystone_auth` section, the `include_service_catalog` option can be set to `False`. The Swift project is perfectly capable of running with Keystone without the need of a service catalog. More information on configuring Swift to work with Keystone in this fashion can be found [here](#).

CHAPTER 4

LDAP

As OpenStack grew and started to gain attention from enterprises, one of the most common initial requests around the Identity space was for LDAP support. This was first introduced, in a basic form, in the Essex release. In this chapter, we will discuss the approach to LDAP integration, the new, comprehensive, configuration options for LDAP, using multiple domains for identity, and, finally, tips, common pitfalls, and troubleshooting.

4.1 Approach to LDAP Integration

LDAP and Microsoft’s Active Directory (AD) are two of the most common existing corporate directories in the enterprise and are used by countless corporations. Given that enterprises already use these as their identity store, it is neither desirable nor practical for this to be replicated in Keystone. Instead, we want Keystone’s Identity APIs to be a front to these corporate enterprise directories, allowing the information they hold to be accessed in-place. Keystone was designed to provide this kind of support by the very nature of its architecture. In essence, it has drivers for different back-end technologies that can be loaded to allow components of Keystone to be backed by external data stores (such as LDAP). As an aside, although we will talk predominantly about LDAP in this chapter, since AD supports the LDAP protocol, everything here applies equally to AD. Where there are differences, we will point them out.

As can be seen in “[1.5 Backends and Services](#)” on page 13, the Keystone architecture is composed of a number of backends and services. Although conceptually they all could have LDAP drivers, in reality only three have such support:

- **Identity:** Storing users and groups
- **Resource:** Storing domains and projects
- **Assignment:** Storing roles and their assignments

In the vast majority of installations, only Identity is mapped to LDAP and usually in read-only mode, since most enterprises have separate, well-established management systems for their corporate directory. The mapping of the Resource and Assignment backends to LDAP is usually problematic; for example, there is no obvious LDAP schema that matches that needed by Keystone, and this needs to be fully read-write from a Keystone perspective. In fact, given these problems, and that so few companies attempted to use these drivers, both the Resource and Assignment support for LDAP is now deprecated in Keystone, and support is planned to be removed in the Mitaka release of OpenStack. Given this, it is highly recommended that LDAP not be used for these drivers and that enterprises only map the Keystone Identity backend to LDAP.

4.2 Configuring Keystone to Integrate with LDAP

In its basic form, Keystone LDAP integration has been included in OpenStack since the Essex release. By providing your specific LDAP options in the Keystone configuration file (*keystone.conf*), Keystone will translate any user and group API calls into something your LDAP server will understand. There is one huge restriction in this classic LDAP support: you can only have one domain in Keystone, which is the default domain. We'll discuss how we use multi-domains in the newer LDAP support (introduced in Juno) later in this chapter, but it is still worthwhile to look through the classic support, since this is the underpinning on which such follow-on enhancements are built. Further, there are some installations for which a single domain is sufficient.

In looking at the task of mapping Keystone users and groups to LDAP, let's first examine what Keystone (and the rest of OpenStack) needs. A user entity in Keystone (that is returned by a `GET /users` call) looks like this:

```
{  
    "domain_id": "default",  
    "enabled": true,  
    "id": "ef7ab1f3fe2745fd91b3d62fd3b7309b",  
    "links": {  
        "self": "http://localhost:5000/v3/users/ef7ab1f3fe2745fd91b3d62fd3b7309b"  
    },  
    "name": "GeorgeSmiley"  
}
```

Now, the `links` attribute is actually filled in by Keystone as part of formatting the entity, and, as described above, in classic LDAP we only support the default domain, so that bit's easy. However, the other attributes need to come from the identity store (be it SQL, LDAP, or whatever)—and in fact, there is one more—password! For security reasons, this is not returned by the Keystone API—but it does need to be accessible to Keystone so that it can check an authentication request. So that's the job of our

classic LDAP support to somehow map `enabled`, `id`, `name`, and `password` to some attributes already stored in an existing LDAP database (but whose schema and exact attribute naming will vary from customer to customer). Further, since customers will already have processes for disabling a user in the corporate directory, how is it that we can ensure such a user is disabled in Keystone? And, of course, there are a similar set of attributes for the Keystone group entity as well, which also need to be mapped:

```
{  
    "description": "A group of spies",  
    "domain_id": "default",  
    "id": "a2ffab1b3fe3745fdaab3d62fd3b75093",  
    "links": {  
        "self": "http://localhost:5000/v3/users/a2ffab1b3fe3745fd...d62fd3b75093"  
    },  
    "name": "Undercover"  
}
```

To solve these issues, Keystone provides a set of configuration options where you specify for each of the needed Keystone entity attributes where in the LDAP schema you will find the equivalent. For example, for the user entity, the configuration file might include the following:

```
user_id_attribute = mail  
user_name_attribute = mail  
user_enabled_attribute = enabled  
user_pass_attribute = userPassword
```

The left-hand side specifies the Keystone entity attribute; the right-hand side the name of the LDAP attribute to use. Similarly for groups:

```
group_id_attribute = cn  
group_name_attribute = cn  
group_desc_attribute = description
```

There are other things Keystone also needs to know, of course. For example, the URL of the LDAP server, the location in the LDAP tree where the user and group objects are stored, etc. So let's look at a complete set of LDAP options in a typical configuration file:

```
[identity]  
driver = ldap  
  
[ldap]  
url = ldap://myservice.acme.com  
query_scope = sub  
  
user_tree_dn = "ou=users,o=acme.com"  
user_objectclass = person  
user_id_attribute = mail  
user_name_attribute = mail  
user_mail_attribute = mail
```

```
user_pass_attribute = userPassword
user_enabled_attribute = enabled
group_tree_dn = "ou=memberlist,ou=groups,o=acme.com"
group_objectclass = groupOfUniqueNames
group_id_attribute = cn
group_name_attribute = cn
group_member_attribute = uniquemember
group_desc_attribute = description

user_allow_create = false
user_allow_update = false
user_allow_delete = false
group_allow_create = false
group_allow_update = false
group_allow_delete = false
```

Let's look, one by one, at the key options in the above configuration settings to see what's being done here:

```
[identity]
driver = ldap
```

This tells Keystone to use the LDAP driver for the [identity] backend (the default is SQL).

The next section, [ldap], adds the specific LDAP settings:

```
url = ldap://myservice.acme.com
```

This tells Keystone the URL of your server.

```
query_scope = sub
```

This tells Keystone what type of depth of search it should do when looking for objects in LDAP. The default is `one` meaning just search at the top level of the specified tree, whereas `sub` means search for any objects within the sub-tree.

```
user_tree_dn = "ou=users,o=acme.com"
```

This specifies the root of the tree in the LDAP server in which Keystone will search for users.

```
user_objectclass = person
```

This specifies the LDAP object class that Keystone will filter on within `user_tree_dn` to find user objects. Any objects of other classes will be ignored.

```
user_id_attribute = mail
user_name_attribute = mail
user_pass_attribute = userPassword
```

This set of options define the mapping to LDAP attributes for the three key user attributes supported by Keystone. The LDAP attribute chosen for `user_id` must be something that is immutable for a user and no more than 64 characters in length.

Although one might think that the Distinguished Name (DN) would be a good choice here, there are problems with this (e.g., it can be longer than 64 characters). Often, an LDAP schema will include a `uid` of some type, which is a good substitute, or alternatively `mail` is a popular choice. The user name must also be unique within the user objects that will be accessed by Keystone.

```
user_mail_attribute = mail
```

Similar to the attributes above, this gives the mapping to the LDAP email attribute. In fact, `email` is not a standard attribute supported by Keystone (i.e., you will not find it defined in the OpenStack Identity API specification), since it can be considered Personally Identifiable Information (PII). As such, Keystone does not automatically provide this for storage in its default SQL database. However, if your LDAP already provides this, then setting the mapping above will cause this to appear in the Keystone user entity as the `email` attribute.

```
user_enabled_attribute = enabled
```

In Keystone, a user entity can be either enabled or disabled. Setting the above option will give a mapping to an equivalent attribute in LDAP, allowing your LDAP management tools to disable a user and for that fact to be reflected in OpenStack (since a disabled user is prevented from authenticating by Keystone).

```
group_tree_dn = "ou=memberlist,ou=groups,o=acme.com"
```

Similar to `user_tree_dn`, this specifies the root of the tree in LDAP in which to search for group objects.

```
group_objectclass = groupOfUniqueNames
```

Similar to `user_objectclass`, this defines the LDAP object class of a group object.

```
group_member_attribute = uniquemember
```

The LDAP attribute within the group object that holds the users who are members.

```
group_id_attribute = cn
group_name_attribute = cn
group_desc_attribute = description
```

This set of options specify the LDAP attributes mapped to the Keystone group entity attributes. Again, as in the case for users, the group id must be unique for the group objects that will be accessed by Keystone, as must the group name.

```
user_allow_create = false
user_allow_update = false
user_allow_delete = false
group_allow_create = false
group_allow_update = false
group_allow_delete = false
```

The above set of options dictate whether any of the Keystone writeable operations should be supported against LDAP (by default they are set to `true`).

With the above configuration (once you have restarted Keystone), you should be able to use Keystone's user and group APIs to read the information in your corporate LDAP, for example:

```
$ openstack user show henryn@uk.acme.com --domain default
+-----+-----+
| Field | Value
+-----+-----+
| domain_id | default
| email | henryn@uk.acme.com
| id | henryn@uk.acme.com
| name | henryn@uk.acme.com
+-----+-----+
```

Issuing the above command will cause Keystone to:

1. Search for the objects of class `person` within the tree rooted by `ou=users, o=acme.com` that have the `mail` attribute equal to `henryn@uk.acme.com` (since we defined mail as the user name)
2. Having found such an object, return a Keystone user entity with the attributes of `id`, `email`, and `name` all set to the value of the `mail` attribute (since that is what we specified in the configuration options)

Note that for the above to work, the user accessing Keystone needs to have anonymous query rights to the `dn_tree` specified in the configuration options. If this is not possible, then you can provide a special user and password account that will be used for that purpose in the configuration file in the `[ldap]` section:

```
user = cn=Manager,ou=users,o=acme.com
password = ga9s86d3das98dd
```

The above account is the only user to bind and search for users/groups.

4.2.1 Other Keystone Configuration Options in Classic LDAP Support

The Keystone configuration supports a number of other additional options and shortcuts that allow integration with other common LDAP schemas and configurations:

```
use_tls = True
tls_cacertfile = /etc/keystone/ssl/certs/cacert.pem
tls_cacertdir = /etc/keystone/ssl/certs/
tls_req_cert = demand
```

The above four options all relate to enabling Transport Layer Security (TLS) for providing a secure connection from Keystone to LDAP—and we highly recommend this. Setting `use_tls` to `true` will cause Keystone to upgrade the connection to SSL. Note

that an alternative to specifying `use_tls` is to specify an alternate port in the URL by using the form “`ldaps://myserver.com:636`”. Whichever method you choose (and you should use one or the other, but not both), the subsequent SSL connection is governed by the other three `tls` options. `tls_cacertfile` gives the name of the file on the Keystone server where the certificate(s) is/are to be found—if you specify this option, then all certificates must be placed inside this one PEM file. Alternatively, the certifications can be stored as individual files in a directory specified in `tls_cacertdir`. Finally, you can control how the certificates are checked for validity in the client (i.e., Keystone end) of the secure connection (this doesn’t affect what level of checking the server is doing on the certificates it receives from Keystone). The client checking is really designed to spot man-in-the-middle (MITM) attacks—and the options for `tls_req_cert` (which corresponds to the standard TLS option `TLS_REQCERT`) are `demand`, `never`, and `allow`. The default of `demand` means the client always checks the certificate and will drop the connection if it is not provided or invalid. `never` is the opposite—it never checks it, nor requires it to be provided. `allow` means that if it is not provided then the connection is allowed to continue, but if it is provided it will be checked—and if invalid, the connection will be dropped.

```
suffix = "o=acme.com"
```

By default, if you don’t specify `user_tree_dn` or `group_tree_dn`, Keystone will construct these by adding `ou=Users` and `ou=UserGroups`, respectively, on the front of the suffix (e.g., `ou=Users,o=acme.com`).

```
use_dumb_member = false
dumb_member = "cn=dumb,dc=nonexistent"
```

`dumb_member` is only used when LDAP groups are writeable and the LDAP class chosen requires the `member` attribute to be defined. If `use_dumb_member` is true, the LDAP `member` attribute will be set to the value defined in `dumb_member` upon creation of a group.

```
user_filter = "objectCategory=person"
group_filter = "objectCategory=group"
```

The two filter options allow additional filters (over and above `user_objectclass` and `group_objectclass`) to be included (i.e., ANDed) into the search of user and group objects, respectively. One common use of this is to provide more efficient searching within AD installations, where the recommended search for user objects is `(&(objectCategory=person)(objectClass=user))`. By specifying `user_objectclass` as `user` and `user_filter` as `objectCategory=person` in the Keystone configuration file, this can be achieved. Similarly, for group searching, the recommended search is `(objectCategory=group)`, which can be achieved by specifying `group_objectclass` as `*` and `group_filter` as `objectCategory=group`.

```
user_enabled_mask = 2
```

Some LDAP schemas, rather than having a dedicated attribute for user enablement, use a bit within a general control attribute (such as `userAccountControl`) to indicate this. Setting `user_enabled_mask` will cause Keystone to look at only the status of this bit in the attribute specified by `user_enabled_attribute`, with the bit set indicating the user is enabled.

```
user_enabled_default = 512
```

Most LDAP servers use a boolean or bit in a control field to indicate enablement. However, some schemas might use an integer value in an attribute. In this situation, set `user_enabled_default` to the integer value that represents a user being enabled.

```
user_enabled_invert = true
```

Some LDAP schemas have an “account locked” attribute, which is the equivalent to account being “disabled.” In order to map this to the Keystone enabled attribute, you can utilize the `user_enabled_invert` setting in conjunction with `user_enabled_attribute` to map the lock status to disabled in Keystone. Note that this option cannot be used in conjunction with either `user_enabled_emulation` or `user_enabled_mask` (see below):

```
user_enabled_emulation = true
user_enabled_emulation_dn = "ou=enabledusers,o=acme.com"
```

Finally, some LDAP schemas do not hold a suitable attribute that can be mapped directly to the Keystone user-enabled attribute, but nonetheless want to allow their LDAP management tools to cause Keystone users to be enabled/disabled. If `user_enabled_emulation` is `true` then Keystone treats the DN specified by `user_enabled_emulation_dn` as a group object containing the list of enabled users. Adding/removing users to/from this group using your LDAP management tools will cause those users to be enabled/disabled in OpenStack.

```
allow_subtree_delete = false
```

Some of the data Keystone stores in LDAP is in tree form, e.g., user and group roles on projects in assignments (and is relevant to Identity if you are also using the LDAP assignment driver). When it comes to deleting groups or projects, by default Keystone will essentially walk the tree deleting the objects. Some LDAP servers support the ability to delete a tree in a single command, and setting `allow_subtree_delete` to `true` will cause Keystone to try and use this facility. If you set this to `true` and your LDAP server does not support tree deletion, then it will respond with an error—in which case Keystone will fall back to tree walking.

```
page_size = 0
```

By default, when searching data, Keystone will talk to the LDAP server in a synchronous fashion. Setting `page_size` to something other than zero will result in Keystone asking the LDAP server to send results in chunks of `page_size` number of entries.

For this to work, your LDAP server must support paging (or Keystone will log an error). Note that although Keystone is now talking to the LDAP server in an asynchronous fashion, Keystone will only return from its API call once all the data has been returned by the LDAP server. In other words, this level of paging does not enable paging at the API level.

```
alias_dereferencing = default
```

Aliasing in an LDAP server is a way of linking objects (i.e., part of one object is an alias to another object). When querying a server, an LDAP client can specify whether these alias links are followed or “dereferenced,” so that the aliased object is returned in the query. The default for Keystone is just to use the underlying configuration of your LDAP client. However, that can be overridden by specifying any of the standard options for LDAP alias dereferencing, which are:

- `never` (which means never dereference)
- `always` (which means always dereference)
- `finding` (which means dereference in *finding* the base object, but not when returning objects from the subsequent *search* of subordinate objects)
- `searching` (which means dereference the objects returned from the *search* of subordinate objects, but not in *finding* the initially specified base object)

```
debug_level = 0
```

A non-zero value of `debug_level` will enable debugging at the LDAP library level, with the value being passed directly into the LDAP calls themselves. You will need to consult your LDAP library and server documentation for the appropriate settings depending on what level of debugging you want to set.

```
chase_referrals = true
```

Most LDAP and AD servers support the concept of referrals. This is where you ask the LDAP server for the value of some object, and it replies “Well, I don’t have that, but I think I know who does, try that server over there.” The LDAP client library can automatically “chase the referral” for us—and your LDAP client library will most likely be configured as to whether it should do that or not. Setting `chase_referrals` in the Keystone configuration file will override the setting in your LDAP client. Note that if referrals are not being chased, and Keystone receives a referral response when querying your LDAP server, it will discard this response (and log a debug error message indicating that you should enable referral chasing). Also note that *referral chasing* is an LDAP client-side activity, as opposed to *chaining*, which is where the LDAP server does this for you. If you are using chaining, you do not need to set `chase_referrals`.

```
user_attribute_ignore = ["default_project_id"]
```

This specifies a list of Keystone user entity attributes that should not be mapped to LDAP, and should be ignored for both read and write operations. The default list for the user object contains `default_project_id`, since this rarely maps to the typical LDAP schema. With this being ignored, in order to obtain a project-scoped token, you must always explicitly provide the `project_id` or name when authenticating to Keystone.

```
user_default_project_id_attribute = None
```

This defines the LDAP attribute that should be mapped to the `default_project_id` attribute in the Keystone user entity. In reality, as described above, this is rarely used since it does not match most LDAP schemas—and it is entirely optional for Keystone, since with the v3 API it simply represents a shortcut from having to define a project scope when authenticating.

```
user_additional_attribute_mapping = []
```

Keystone supports the dynamic adding of attributes to its entities, allowing cloud deployers and developers to, for instance, extend the user object. When the backend is SQL, these are stored in a hidden (at the API level) column called `extra`. For LDAP, since an additional attribute needs to be mapped to the schema, you can set `user_additional_attribute_mapping` to indicate the name of the Keystone attribute and the LDAP attribute to which it is to be mapped. This option is a list of pairs of mappings for the form "`keystone_attrib`": "`LDAP attribute`", for example:

```
user_additional_attribute_mapping = ["cell_number": "mobileTelephoneNumber"]
```

With the above specified in the configuration, the user entity returned by Keystone from a `GET /users` API call would look like something like this:

```
{
    "cell_number": "925-482-7336",
    "domain_id": "default",
    "enabled": true,
    "id": "ef7ab1f3fe2745fd91b3d62fd3b7309b",
    "links": [
        "self": "http://localhost:5000/v3/users/ef7ab1f3fe2745fd91b3d62fd3b7309b"
    ],
    "name": "GeorgeSmiley"
}
group_attribute_ignore = []
```

This specifies a list of Keystone group entity attributes that should not be mapped to LDAP, and should be ignored for both read and write operations.

```
group_additional_attribute_mapping = []
```

Similar to `user_additional_attribute_mapping`, this allows you to define additional attributes to be mapped from your LDAP group entity into the one Keystone will return via its API.

```
use_pool = false
pool_size = 10
pool_retry_max = 3
pool_retry_delay = 0.1
pool_connection_timeout = -1
pool_connection_lifetime = 600
```

This batch of options relates to performance optimizations by instructing Keystone to maintain a pool of connections to the LDAP server, and hands these out to be used by incoming request processing rather than having to build and tear down a connection for each API request. The main pool is used for all requests except authentication requests (see below). Use of the pool is disabled by default, and setting `use_pool` to true will enable it. Set `pool_size` to the maximum likely number of concurrent requests.

```
use_auth_pool = false
auth_pool_size = 100
auth_pool_connection_lifetime = 60
```

The above allows you to further refine performance by having a separate pool of connections for authentication requests, as opposed to those used for other API processing.

4.3 Multiple Domains and LDAP

4.3.1 Requirements for Multi-Domain Corporate Directory Support

There are two main use cases for multi-domain LDAP with Keystone:

- A cloud provider who wants to support multiple customers on their cloud, each of which wants to integrate with their own corporate directory.
- An enterprise wants to model their internal company structure around domains—for example, each division of the company might be a domain. Each of these divisions might have their own LDAP server, or perhaps a separate tree of users within a common company LDAP server.

In both cases, the first key issue for Keystone is how to hook different LDAP URLs (and other configuration options) to different domains. The domain definitions are stored in the Keystone Resource backend, but somehow the Identity backend needs to know on a domain-by-domain basis to which LDAP server to send the request. Easy, you might think: just have an index by domain, right? Unfortunately, most user and group API calls do not include a domain identifier, so that's not so easy after all. One of Keystone's functions for Identity is to be the source (for other OpenStack services) of a globally unique ID (for each user and group). Once a user is created, this unique ID is used in most Keystone calls to identify the user or group in question; for

example, to get a user entity you would issue a Keystone API call specifying just the user ID:

```
GET /v3/users/d19cac8f7ef3e6de36a47b85f306f137
```

How will Keystone know which domain/LDAP server to talk to in order to find this user? For performance reasons, searching them all is not a viable option. This problem was the main reason it took until Juno to support multi-domain LDAP. Further, as we saw from the classic (mono-domain) LDAP support in the previous section, the Keystone configuration file specifies the LDAP attribute to map to the `user_id` or `group_id`. Given the (OpenStack) constraint that this ID must be globally unique (i.e., across all the LDAPs being interfaced by one Keystone), a second issue is whether we would trust any given LDAP to be providing such a globally unique attribute. Imagine if we just let whatever ID the underlying LDAP administrator chose for a user to become the ID that had to be unique across the LDAP directories of all the other customers (who may even be their competitors) who were hosted on the same cloud? Sounds like a recipe for disaster! The solution to both issues was to get Keystone to provide an intermediate ID mapping, which creates a “public” unique ID that is published to the rest of OpenStack, and builds a table that will map this public ID back to its Domain, LDAP server, and local ID within that LDAP server. This directory-mapping layer is essentially invisible to callers of the Keystone API, and the mapping is built on the fly as Keystone encounters entities in its identity drivers. The only thing a caller would notice is that the IDs for users and groups have gotten bigger. (With multi-domain enabled, they are, by default, 64-byte SHA256 hashes as opposed to the classic 32-byte UUIDs.) It should be noted that this directory mapping is not related to the Keystone federation mapping used to handle the mapping of identity provider assertion attributes into Keystone attributes.

So let’s take a look at an example of using domain-specific directories in practice. In a service provider that offered cloud services to its customers, it would be common to represent a customer as a domain in Keystone. This allows customers to have their own users, groups, and projects. What the provider would like to do is to be able to offer customers the ability to use their existing corporate directory for authentication to their domain—enabling each customer domain to point to that customer’s own LDAP/AD service. The first thing to do is to enable domain-specific configuration support in Keystone (it is disabled by default) by setting the following option in the main Keystone configuration file:

```
[identity]
domain_specific_drivers_enabled=true
```

The next step is to decide how you will provide the domain-specific LDAP configuration options (e.g., URL to the LDAP server). Keystone supports two methods for doing this:

- By providing a configuration override file for each domain (this method has been supported since Juno).
- By using some new additions to the Keystone API to specify the LDAP overrides for each domain. (This was introduced in Kilo, and is currently marked as experimental. Despite being marked as experimental, it is fully functional and is likely to become the primary supported mechanism in the Mitaka release.)

Both methods achieve the same goal: telling Keystone how to handle LDAP for a given domain. You can't mix the two approaches—it's either/or—so you need to decide which one to use. The API approach has the benefit that you can onboard a domain (including setting its LDAP parameters) purely via the Keystone API. The configuration file approach, while being familiar to anyone setting up an OpenStack service, means you must ensure the file is created on your Keystone server for each LDAP-backed domain you set up.

4.3.2 Setting Up Multi-Domain Using the Configuration File-Based Approach

To use this approach, you specify to Keystone a directory on the Keystone server in which to locate the domain-specific configuration files (one per domain).

```
[identity]
domain_config_dir=/etc/keystone/domains
```

When domain-specific configuration files are enabled, each time Keystone is started it will scan the contents of *domain_config_dir* for any domain-specific configuration files. For each domain that requires its unique corporate directory, a domain-specific configuration file is required that contains the specific details of the corporate directory for that domain. For example, if the service provider creates a domain called “customerA” to represent that customer, then they create a domain-specific configuration file called *keystone.customerA.conf*. (The name is important and is always of the form: *keystone.{domain-name}.conf*—the domain name must match the name of the domain created in Keystone.) The contents of this configuration file might look something like this:

```
[ldap]
url = ldap://ldapservice.thecustomer.com
query_scope = sub

user_tree_dn = ou=Users,dc=openstack,dc=org
user_objectclass = MyOrgPerson
user_id_attribute = uid
user_name_attribute = mail
user_mail_attribute = mail
user_pass_attribute = userPassword
user_enabled_attribute = enabled
group_tree_dn = ou=UserGroups,dc=openstack,dc=org
```

```
group_objectclass = groupOfUniqueNames
group_id_attribute = cn
group_name_attribute = cn
group_member_attribute = uniqueMember
```

As you can see, these are exactly the same types of configuration options that are normally set in the main Keystone configuration file—except in this case they are specified in the specific configuration file for this domain and act as overrides to any options specified in the main Keystone configuration file. Note that only the options that are specific to the domain need to be specified here; common options can stay in the main Keystone configuration file.

Once a domain has its own configuration file, Keystone will build a mapping for the user and group entities in that backend as it encounters them, and route any requests issued for those entities to the appropriate corporate directory.

While it is primarily designed to support different corporate directories, you can also specify a configuration file for a domain to use an SQL backend. This is particularly useful if you want to have some local users (such as service users or local admin users) that you don't want to place in a corporate directory. This is done in exactly the same way as in the LDAP/AD case: Create a domain-specific configuration file for a domain using the same naming convention, but in this case it will probably just contain the name of the driver. For example, if you want domain CloudAdmin to be backed by SQL while all your other domains perhaps use LDAP, then create a domain-specific configuration file called *keystone.CloudAdmin.conf* that contains:

```
[identity]
driver = sql
```

The big restriction today, however, is that Keystone currently allows only one SQL driver to be loaded at a time—so either this can be the catch-all driver (by specifying the Identity driver as SQL in the main Keystone configuration file), or it can be assigned to a specific domain (using a domain-specific configuration file). If you try specifying more than one domain with its own SQL configuration, Keystone will raise an exception when starting up. Future versions of Keystone may lift this limitation.

As was stated at the start of this subsection, the configuration files are only read when starting Keystone—so every time you add or change any of the LDAP configuration parameters, you need to restart Keystone. The alternative approach is to use an extension that is part of the Keystone API, which modifies the operation of a running Keystone—this is described in the next subsection.

4.3.3 Setting Up Multi-Domain Using the Keystone API-Based Approach

The Keystone API approach is similar to the configuration file approach, except that rather than specify the LDAP override options in a file, you can specify them by iss-

ing an API call against the domain in question. This support is (in Liberty) marked as experimental, although it is fully functional, and is targeted to be a part of the core API in the Mitaka release. It is enabled by the (rather poorly named) option in the main Keystone configuration file:

```
[identity]
domain_configurations_from_database=true
```

Having enabled the capability, you can now use the domain configuration management part of the Identity API, e.g., to get or modify the overrides for a given domain:

```
[GET | PUT | PATCH | DELETE] /domains/{domain_id}/config
```

As of the Liberty release, these APIs are not yet supported from OpenStackClient (the changes to add them are currently under review), so using a cURL request to create the configuration overrides for a given domain might look like:

```
$ curl -s -X PUT -H "X-Auth-Token: $OS_TOKEN" \
-H "Content-Type: application/json" -d '
{
    "config": {
        "identity": {
            "driver": "ldap"
        },
        "ldap": {
            "url" = "ldap://ldapservice.thecustomer.com"
            "query_scope" = "sub"
            "user_tree_dn" = "ou=Users,dc=openstack,dc=org"
            "user_objectclass" = "MyOrgPerson"
            "user_id_attribute" = "uid"
            "user_name_attribute" = "mail"
            "user_mail_attribute" = "mail"
            "user_pass_attribute" = "userPassword"
            "user_enabled_attribute" = "enabled"
            "group_tree_dn" = "ou=UserGroups,dc=openstack,dc=org"
            "group_objectclass" = "groupOfUniqueNames"
            "group_id_attribute" = "cn"
            "group_name_attribute" = "cn"
            "group_member_attribute" = "uniquemember"
        }
    }
}' http://localhost:5000/v3/domains/421370cd78234413bbeb5d7ce1c73077/config
```

The above is the equivalent to the configuration file described in “[4.3.2 Setting Up Multi-Domain Using the Configuration File-Based Approach](#)” on page 57.

In fact, the domain configuration management API extensions are quite powerful, and can be used to read and modify individual options, not just to bulk load/read the entire block. For instance, let’s say that the LDAP server URL for the domain in the above example was changed, you could update just that option using the following API command (via cURL):

```
$ curl -s -X PATCH -H "X-Auth-Token: $OS_TOKEN" \
      -H "Content-Type: application/json" -d \
{
    "url" = "ldap://ldapservice2.thecustomer.com"
}' \
http://localhost:5000/v3/domains/421370cd78234413bbeb5d7ce1c73077/config/ldap/url
```

Similarly, you can just get the value of an individual option:

```
$ curl -s -H "X-Auth-Token: $OS_TOKEN" -H "Content-Type: application/json" \
      http://localhost:5000/v3/domains/421370cd78234413bbeb5d7ce1c73077/config/ldap/u
rl | python -mjson.tool
{
    "url" = "ldap://ldapservice2.thecustomer.com"
}
```

It is likely that over time this API-based method of domain configuration management will become the primary method, and that the configuration file-based method will be deprecated. However, as stated above, currently this API method is still marked as experimental. This is mainly due to the fact that there are some corner cases that are not yet well handled within its implementation—primarily to do with what happens in a multiprocess installation of Keystone (i.e., one where multiple Keystone processes are used to improve the response time and throughput). The challenge for Keystone in this case is to ensure that all processes see the update (and any cache copies are updated), and further that they see the update in an atomic fashion (i.e., if you are updating multiple configuration options for a domain, that all processes see all the updates or none of the updates). The implementation for Liberty solves the first of these challenges, but it is possible that if you try and access the LDAP users or groups at the same time you are updating the domain configuration options that for a short period that other process could see a partial update (until the caches timeout). So if you choose to use this capability, then either ensure creating/updating the LDAP options are done when the domain is not in use, or worst case, ensure that you wait for at least a period of `cache_time` before using the domain (by default this is 300 seconds, i.e., 5 minutes). The `cache_time` is itself configurable in the main Keystone file using the option:

```
[domain_config]
cache_time=300
```

As stated, these restrictions are planned to be resolved in the Mitaka release, and the API management method will be declared as *stable*.

In summary, the API method of setting up the domain LDAP configuration options has several advantages over the file-based method:

- It is all done via the Keystone API, so there's no need to separately distribute the domain-specific configuration file. Given that the API can also be used in a fine-grained fashion, this is also easier to integrate into your management systems.

- Does not require a restart of the Keystone server, ensuring no interruption of service when onboarding a domain.

4.3.4 Restrictions When Using Multi-Domain Identity

The main restriction relates to one of the common queries people want to make when first starting to test Identity, which is “list me all the users in the system,” using the `GET /users` API call. Now, for small systems when all the users are in an SQL database, such a query doesn’t seem unreasonable. However, imagine a setup where there are many tens (or even hundreds) of customers supported in a single cloud, each in their own domain, each using their own corporate LDAP. In such situations, satisfying “list me all the users in the system” would require a scan of all the separate corporate LDAP servers and merging the results. But, more importantly, does asking such a question really make any sense? The primary use of multi-domain is to enable each customer to have their own LDAP support, so the use of such a “list everything” command would have to be restricted to administrators of the cloud provider, who, in reality, simply do not need to have a list of all the users of all their customers. As such, if you enable multi-domain support, Keystone will require you to specify a domain as part of a list all users request in one of two ways:

1. Add a domain filter to the request:
`GET /users?domain_id=421370cd78234413bbeb5d7ce1c73077`
2. Use a domain-scoped token when issuing the request, and the scope of the token will be used as the domain to search

Hence, with multi-domain enabled, you can no longer ask to “list me all the users in the system,” but you can ask to “list me all the users in this domain.”

A second restriction, which is probably already apparent, is that multi-domain is only supported when using the Keystone v3 API. The v2 API does not support the concept of domains, so there is no way to separate the different namespaces.

4.3.5 Choosing the Right Mix of SQL and LDAP Domains

As discussed already, the main use of multi-domain LDAP is to allow the users and groups of newly created domains to be drawn from specific LDAP corporate directories. In setting up multi-domain to achieve this, there are a couple of decisions you need to make:

1. What backing store will be used for the *default* domain? Before trying to answer this, a little background is needed. Originally, before Keystone v3, there were no domains—users were “global” across a Keystone installation. With the introduction of domains, users and groups are owned by a given domain; however, that begged the question as to what happens to the users created before the v3 API

was introduced? Also, a modern Keystone supports both the v3 and v2 APIs (and although v2 will be deprecated soon, you can still call it), so what happens to users created via the v2 API today (since you can't specify what domain they are in)? To solve these issues, Keystone automatically creates a "default" domain (which has an ID of `default` and a name of `Default`). Any users created using the v2 API will be placed in this domain.

2. Where are the service users going to be created? Service users are the user accounts that other services (e.g., Nova, Glance, etc.) utilize to make calls to Keystone. Since these already existed pre-v3, these typically live in the default domain, particularly since until recently some services still made v2 API calls to Keystone (and a v2 API call is only allowed to see users in the default domain). With the latest keystone middleware (the Keystone client code that is used by services to call Keystone), services can now specify the domain ID for the service user, enabling cloud providers to move their service users into a separate service domain.

So given the above, you have three basic choices relating to the default domain, as discussed in the following.

Use SQL for the Default Domain

The most obvious solution for new installations is that the default domain is backed by a local Keystone SQL database, and all other domains use LDAP. One important advantage to this approach is that service users can be created locally, without the need for them to be added to a corporate directory.

In fact, there are two ways of configuring Keystone so that the default domain is SQL. Depending on which you choose there will be implications for any other SQL domains you want to support:

1. Set the Identity driver to `sql` in the main Keystone configuration file, and then add domain-specific LDAP configurations (using the file- or API-based mechanisms) for those domains that have their own LDAP. What this configuration allows is for the SQL driver to handle not just the default domain, but any other domain that doesn't have explicit LDAP configurations defined. This can be quite a common multi-customer cloud scenario—some smaller customers just want their users and groups managed by the cloud provider, some larger customers want to link to their existing corporate directory. Using this SQL configuration, this gives a cloud provider the option to choose to use SQL for a set of smaller customer domains if that was more cost effective (these customers' users and groups are still in their own domains, so their namespaces are maintained). Of course, there's no reason the cloud provider could not also use an internal LDAP server with different tree attach points for these smaller customers, if that suited their requirements better.

2. Create a specific domain configuration for the `default` domain and in it set the Identity driver to `sql`. Using the API-based method you would issue the following call:

```
$ curl -s --request PUT -H "X-Auth-Token: $OS_TOKEN" \
-H "Content-Type: application/json" -d '
{
    "config": {
        "identity": {
            "driver": "sql"
        }
    }
}' http://localhost:5000/v3/domains/default/config
```

Alternatively, if you are using the file-based configuration method you would create a domain-specific configuration file of the name `keystone.Default.conf` (note how the name of the default domain is used in the filename) that includes just the `[identity]` section and set the `driver` to `sql`. Also set the identity driver in the main Keystone configuration file to `ldap` (but don't worry about any of the other LDAP configuration options in the main configuration file). This will tell Keystone that *only* the default domain is backed by SQL and all other domains must be given their own LDAP configuration—and access to a domain won't work without creating a specific LDAP configuration for it.

This method is probably the simplest to set up. But one thing to keep in mind, however, is that if you still need to support users accessing Keystone via the v2 API, then they will access the default domain, which is SQL. There is no way of allowing v2 clients to access any of the LDAP domains.

Use LDAP for All Domains, Except an SQL Service Domain

This is a new capability using kestonemiddleware plus enhancements to multi-domain support that was added to Keystone in Liberty. To configure this you need to:

- Set the identity driver to `ldap` in the main Keystone configuration file.
- Create a domain in Keystone that will hold your service users, and create a domain-specific configuration for this service domain, setting the identity driver to `sql`.
- Create your service users in this new domain (which will be stored in the Keystone SQL database).
- Modify the configuration file for each of your other services to specify the `domain_id` for the service users, e.g., in `nova.conf` you would set the following:

```
[keystone_auth_token]
username = nova
user_domain_id = 7c0df68c6b734735a22b2365f241b9b8
```

For any additional domains, create LDAP domain-specific configurations for each one, using either the file- or API-based mechanisms.

This SQL/LDAP mix requires that all services be configured to use the domain for service users as described above. Provided you can do this, then you kind of get the best of both worlds—LDAP for both the default and additional domains, with your service users stored locally in a separate service domain. In addition, you preserve the ability (if you need it) for any user access via the v2 API to successfully access the LDAP-backed default domain.

Use LDAP for All Domains

The third approach is LDAP for all domains, including the default domain. This is often used in installations where LDAP was already in place prior to multiple domains being supported (with the Identity driver set to `ldap` in the main keystone configuration file). In these scenarios, the service users will have already been created in the LDAP directory that is backing the default domain. Under this circumstance, multi-domain can be enabled and additional domains provided with their own configurations. Note that in this setup, all domains (except the default domain) will require their LDAP configuration to be defined using either the file- or API-based mechanisms.

Finally, as an aside, remember that in all of the above methods, we are talking about whether to use SQL or LDAP for the store of *Identities*. Whichever mix of SQL and LDAP you chose for your domains, you can always use SQL for the storage of resources and assignments (and indeed this is recommended; we will discuss this more in “[4.5 Projects, Roles, and Assignments from LDAP \(Just Say NO!\)](#)” on page 72).

4.4 A Practical Guide to Using Multi-Domains and Keystone

In this section, we show how to use Keystone against IBM’s internal corporate directory (“bluepages”), which is a standard LDAP server. Although the particular LDAP attributes might be different in your setup, the principles will be the same.

4.4.1 Setting Up LDAP

Like the early examples from [Chapter 2](#), we start from a fresh DevStack installation of OpenStack, which will have set up the default domain as a service domain containing the service accounts OpenStack needs. Hence, the first task is to add a new domain that will subsequently be configured so its users and groups are mapped to the bluepages directory, where the user accounts for all IBMers are stored.

```
$ openstack domain create ibm
+-----+
| Field | Value |
+-----+
| enabled | True |
| id | 421370cd78234413bbeb5d7ce1c73077 |
| name | ibm |
+-----+
```

Let's also create a project within that domain (both the domain and project records are actually being stored in SQL in Keystone since the Resource backend is backed by SQL).

```
$ openstack project create ibmcloud --domain ibm
+-----+
| Field | Value |
+-----+
| description | |
| domain_id | 421370cd78234413bbeb5d7ce1c73077 |
| enabled | True |
| id | 95ae2a9c259b4012b8b7e8ad7dc9a939 |
| name | ibmcloud |
| parent_id | None |
+-----+
```

Now we need to tell Keystone to use multi-domains and that for this specific IBM domain, to go and talk to the IBM bluepages server. First, we update the main Keystone configuration file (*keystone.conf*) to enable the domain-specific driver option:

```
[identity]
domain_specific_drivers_enabled = true
```

Then we need to set the LDAP options for the specific IBM domain. As described earlier, there are two ways to do this: using the Keystone API or by creating a separate configuration file for this domain. We'll show both ways here.

To use the API method, we first need to enable this option (in addition to setting `domain_specific_drivers_enabled` to `true`):

```
[identity]
domain_configurations_from_database=true
```

Then we simply call the domain configuration management PUT API to create the options all in one go:

```
$ curl -s --request PUT -H "X-Auth-Token: $OS_TOKEN" \
-H "Content-Type: application/json" - d '
{
    "config": {
        "identity": {
            "driver": "ldap"
        },
        "ldap": {
            "url": "ldaps://bluepages"
        }
    }
}'
```

```

"url" = "ldap://bluepages.ibm.com",
"query_scope" = "sub",
"user_tree_dn" = "ou=bluepages,o=ibm.com",
"user_objectclass" = "ibmPerson",
"user_id_attribute" = "uid",
"user_name_attribute" = "mail",
"user_mail_attribute" = "mail",
"user_pass_attribute" = "userPassword",
"user_enabled_attribute" = "enabled",
"group_tree_dn" = "ou=memberlist,ou=ibmgroups,o=ibm.com",
"group_objectclass" = "groupOfUniqueNames",
"group_id_attribute" = "cn",
"group_name_attribute" = "cn",
"group_member_attribute" = "uniqueMember",
"group_desc_attribute" = "description",
"user_allow_create" = false,
"user_allow_update" = false,
"user_allow_delete" = false,
"group_allow_create" = false,
"group_allow_update" = false,
"group_allow_delete" = false
}
}
}' http://localhost:5000/v3/domains/421370cd78234413bbeb5d7ce1c73077/config

```

That's it for the API method—we're done. Skip to “[4.4.2 Running Admin Commands](#)” on page 67 to try some commands to test our new setup.

The alternative to the above is to create a domain-specific configuration file. To enable this, we need to tell Keystone where to find this file:

```
[identity]
domain_config_dir = /etc/keystone/domains
```

Now we create the IBM domain-specific configuration file (*keystone.ibm.conf*) in the */etc/keystone/domains* directory, and use it to specify the bluepages-specific settings:

```
[identity]
driver = ldap

[ldap]
url = ldap://bluepages.ibm.com
query_scope = sub

user_tree_dn = "ou=bluepages,o=ibm.com"
user_objectclass = ibmPerson
user_id_attribute = uid
user_name_attribute = mail
user_mail_attribute = mail
user_pass_attribute = userPassword
user_enabled_attribute = enabled
group_tree_dn = "ou=memberlist,ou=ibmgroups,o=ibm.com"
```

```

group_objectclass = groupOfUniqueNames
group_id_attribute = cn
group_name_attribute = cn
group_member_attribute = uniquemember
group_desc_attribute = description

user_allow_create = false
user_allow_update = false
user_allow_delete = false
group_allow_create = false
group_allow_update = false
group_allow_delete = false

```

Before continuing, we need to restart Keystone for the changes to take effect:

```
$ sudo service apache2 restart
```

4.4.2 Running Admin Commands

Finding a user

In this example, we attempt to find an LDAP user within the domain that is backed by LDAP.

```

$ openstack user show stevemar@ca.ibm.com --domain ibm
+-----+-----+
| Field | Value |
+-----+-----+
| domain_id | c3c24e60d57e4461ad64b372c14128b7 |
| email | stevemar@ca.ibm.com |
| id | c29cac8f7003e6de36a47b85f306f137bea8026e3d55c35aed27dfbf09c8fb28 |
| name | stevemar@ca.ibm.com |
+-----+-----+

```

Finding groups a user is a member of

In this example, we list all LDAP groups a user is a member of. Similar to the above command, we need to specify the domain the user can be found in. Note that if a user is not specified as a filter, then Keystone will attempt to list all LDAP users, an operation that will likely result in a timeout.

```

$ openstack group list --user stevemar@ca.ibm.com --user-domain ibm
+-----+-----+
| ID | Name |
+-----+-----+
| 178e5df37393dd6695c4d93382...60b21b393373ff8360a191b74bb | SWG_Canada |
| 5f620ab5abe2f0b14e3112357d...3ac637e7416b7d562152f73e938 | Toronto_Lab_VPN |
| ed9f9cb5dbe7d6a6e5808ca979...b5d160e0a7308c199680c595a96 | HiPODS-OpenCloud |
+-----+-----+

```

List all members of a group

In this example, we attempt to list all users in a group. We can group by ID or by using the group's name and owning domain. Similar to the previous command, if a group is not specified as a filter, then Keystone will attempt to list all LDAP groups, an operation that will likely result in a timeout. Note that the ability to list users in a group by group and domain name will be available in the Mikata release of OpenStackClient.

```
$ openstack user list \
    --group ed9f9cb5dbef6a6e5808ca97921734da3b90b5d160e0a7308c199680c595a96
+-----+
| ID | Name |
+-----+
| c29cac8f7003e6de36a47b...a8026e3d55c35aed27dfbf09c8fb28 | stevemar@ca.ibm.com |
| f3ca1de06fbe100e71577a...d792686ae75812becfe43d5b4aa09c | topol@us.ibm.com |
+-----+
```

Assigning a group a role on a project

In this example, we want to provide authorization to all members of a group. All users within the group will now have the role on the project specified.

```
$ openstack role add member --group "HiPODS-OpenCloud" --group-domain ibm \
    --project ibmcloud --project-domain ibm
```

4.4.3 Running LDAP User Commands

Setting up LDAP credentials

In this example, we attempt to authenticate as an LDAP user. To do this, we should set our environment variables to our LDAP credentials, and update the project domain and user domain values as necessary. The resultant settings should look like:

```
$ env | grep OS
OS_IDENTITY_API_VERSION=3
OS_PASSWORD=My5uper5ecretPa$$word
OS_AUTH_URL=http://172.16.240.134:5000/v3
OS_USERNAME=stevemar@ca.ibm.com
OS_USER_DOMAIN_NAME=ibm
OS_PROJECT_NAME=ibmcloud
OS_PROJECT_DOMAIN_NAME=ibm
```

Getting a token

In this example, we attempt to authenticate and retrieve a token using our newly set credentials.

```
$ openstack token issue
+-----+
| Field | Value |
+-----+
```

```

+-----+
| expires | 2015-08-13T21:54:15.411376Z |
| id      | 7c0df68c6b734735a22b2365f241b9b8 |
| project_id | e68bd223fb8045f6b7b2b7aa926433c8 |
| user_id | c29cac8f7003e6de36a47b85f306f137bea8026e3d55c35aed27dfbf09c8fb28 |
+-----+

```

Listing images

In this example, we attempt to list all public images that are available to the user. This highlights the fact that LDAP users are able to communicate with other OpenStack services.

```

$ openstack image list
+-----+-----+
| ID           | Name          |
+-----+-----+
| 1f8a0809-5aa5-454a-916b-6077c00e20e1 | cirros-0.3.4-x86_64-uec |
| 1d181a59-7ab7-4b5c-a351-118383d0b58b | cirros-0.3.4-x86_64-uec-ramdisk |
| b8aa23ee-1791-4a0e-b42a-cac6f7607b21 | cirros-0.3.4-x86_64-uec-kernel |
+-----+-----+

```

Creating a VM

In this example, we create a VM using the Nova service, once again illustrating that we are able to use other OpenStack services.

```

$ openstack server create myVM --image cirros-0.3.4-x86_64-uec --flavor 1
+-----+-----+
| Field          | Value        |
+-----+-----+
| OS-EXT-AZ:availability_zone | nova |
| OS-EXT-STS:task_state       | scheduling |
| OS-EXT-STS:vm_state         | building |
| adminPass           | PQDNiбоGaNQ2 |
| created             | 2015-08-14T07:35:36Z |
| flavor              | m1.tiny (1) |
| id                  | fe2b4ab0-80cf-48fe-ad5b-3e8b9624edd1 |
| image               | cirros-0.3.4-x86_64-uec (1f8a0809-... |
| name                | myVM |
| project_id          | e68bd223fb8045f6b7b2b7aa926433c8 |
| security_groups     | [{u'name': u'default'}] |
| status               | BUILD |
| updated              | 2015-08-14T07:35:36Z |
| user_id              | c29cac8f7003e6de36a47b85f306f137be... |
+-----+-----+

```

4.4.4 Authenticating with Horizon

Updating the Horizon configuration file

In order for LDAP users to authenticate through Horizon, certain options need to be enabled. In the *horizon/openstack_dashboard/local/local_settings.py* file, make the following changes:

```
OPENSTACK_API_VERSIONS = {  
    "data-processing": 1.1,  
    "identity": 3,  
    "volume": 2,  
}  
OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT = True  
OPENSTACK_KEYSTONE_URL="http://172.16.240.134:5000/v3"
```

Log in with LDAP credentials and specify the domain name

In the screenshots below we can see the modified Horizon landing page, which now expects a user to specify the domain they want to authenticate against.

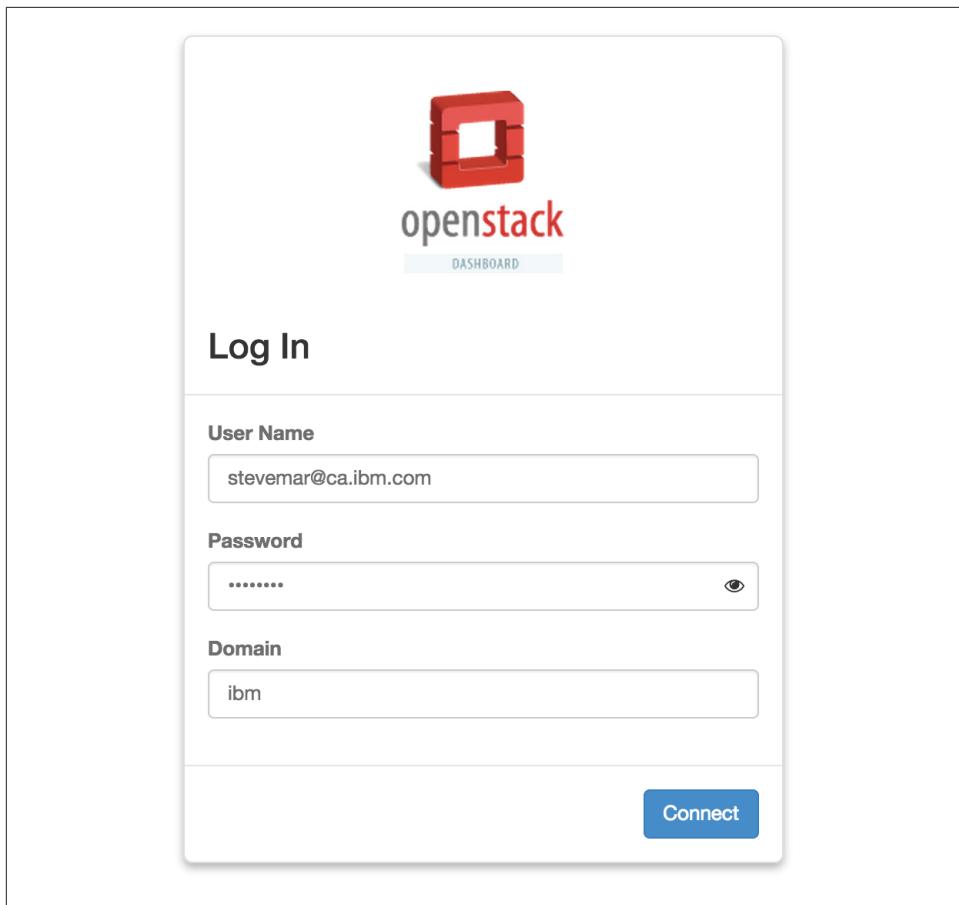


Figure 4-1. Horizon landing page with an additional text box for the domain name

Once logged in, a user is able to see their username and domain along the top of the dashboard.

The screenshot shows the OpenStack dashboard after logging in. The top right corner displays the user's email address: 'stevemar@ca.ibm.com'. The left sidebar shows navigation options like 'Project', 'Compute', 'Overview', 'Instances', 'Volumes', 'Images' (which is currently selected), and 'Identity'. The main content area is titled 'Images' and lists three items in a table:

Figure 4-2. Once logged in, the user's email address should be visible at the top right.

4.5 Projects, Roles, and Assignments from LDAP (Just Say NO!)

Identities in LDAP work well. They are typically managed by an LDAP administrator, using existing enterprise management tools (that most likely ensure an enterprise's other systems are also maintained; e.g., human resources, etc.). When employees are hired or fired from the enterprise company, they are added and removed from LDAP accordingly. Similarly, when users are added and removed from groups in the corporate LDAP, Keystone's view of the world is also updated (since, of course, Keystone is just using the corporate LDAP as its Identity store). This is a great fit for Identity and a common approach used by many applications.

Initially, when the LDAP driver was written in the Essex release, there was no separation between the Project, Role, and User services—they were all lumped together in the same backend together. This resulted in the first few iterations of LDAP support in Keystone forcing administrators to keep this data in their LDAP, a very unpopular restriction. Since then, the Keystone team has worked hard to separate the services and allow each to use their own backends (as described earlier, called, Identity, Assignment, and Resource). While Keystone still supports an LDAP driver for each of these three backends, we overwhelming suggest that you not store Assignment and Resource data in LDAP, but rather use a database such as SQL. The reasons for this are as follows:

- While there is a pretty good schema match for Identity with LDAP, the same is not true for Assignment and Resource. You will most likely have to modify your schema significantly to support OpenStack data.
- In general, Keystone needs the backends for Assignment and Resource to be read-write, something most corporate directory groups will feel uneasy about.
- Due to the lack of usage, the actual backend support for LDAP in Assignment and Resource is not as up to date as SQL. In particular, you can only use the default domain (the multi-domain support implemented in Identity is not provided with Assignment and Resource). In addition, domain role assignments are not supported on the default domain.

Given the above, as stated earlier, the LDAP backends for Assignment and Resource are likely to marked as deprecated in Liberty, for removal in the Mitaka release. For clarity, let us emphasize that this does not mean any reduction in support or commitment from Keystone to LDAP for Identity. This remains a fundamental part of Keystone and represents the most common backend for Identity in use in larger enterprise deployments.

4.6 Tips, Common Pitfalls, and Troubleshooting

In this section, we describe some commonly experienced pitfalls that occur when using LDAP and the multi-backend approach to identity in Keystone and provide troubleshooting tips to address these issues.

4.6.1 General LDAP Issues

Missing Python LDAP libraries

If a user is seeing either of the following errors, then they may be missing LDAP Python libraries (these libraries are not installed by default).

```
$ openstack user show stevemar@ca.ibm.com --domain ibm  
ERROR: openstack An unexpected error prevented the server from fulfilling your  
request: No module named ldap.filter (Disable debug mode to suppress these  
details.) (HTTP 500) (Request-ID: req-74bb843e-cc52-482e-9a25-4366a276ba3d)
```

or

```
$ openstack user show stevemar@ca.ibm.com --domain ibm  
ERROR: openstack An unexpected error prevented the server from fulfilling your  
request: No module named ldappool (Disable debug mode to suppress these  
details.) (HTTP 500) (Request-ID: req-74529f55-079b-4afd-b71d-6270fe4c1b2d)
```

To resolve either of these issues, install the required libraries:

```
$ sudo pip install python-ldap  
$ sudo pip install ldappool
```

Or (as of the Liberty release of Keystone):

```
$ sudo pip install keystone[ldap]
```

Use tools to help you determine LDAP attributes

If you are unaware of the values for LDAP attributes that are necessary to configure Keystone, we suggest downloading software to help you determine these values. There are several free tools available, such as Xplorer, which have a graphical user interface that are easy to use. Using a simple value like name or email to find a single entry in LDAP is usually sufficient, since this will allow you to see all the fields available to every user.

4.6.2 Tips for Using Multi-Domain LDAP

When using the configuration file-based method, make sure you set up things in the right order

If you are starting with the classic configuration and you only have the default domain that points at a single LDAP/AD corporate directory, then the order of steps should be:

1. Set `domain_specific_drivers_enabled` to “true” in the main Keystone configuration file and restart Keystone.
2. Create the domain for which you want to have specific configurations.
3. Create the domain-specific configuration files for each of these domains.
4. Restart Keystone again.

With LDAP as the default driver, if you don’t first set `domain_specific_drivers_enabled` to `true`, Keystone will prevent you from creating new domains (since it knows the default LDAP driver can only handle the default domain).

If you have set up your domain and the configuration file but you still can’t access the data in the corporate directory, then it might be worth checking the Keystone log file. At startup, as Keystone is scanning the `domain_config_dir`, it will examine each file. If it doesn’t match the form `keystone.{domain-name}.conf`, then it will be ignored. If Keystone can’t find the domain of the name specified in the file name, then it will also ignore it. In both situations, Keystone will log a warning with the name of the problematic file. For example:

```
Invalid domain name (CustomerB) found in config file name
```

Remember, you can’t list all the users

As described earlier, with multi-domain, Keystone requires you to specify a domain when listing “all users.” If you fail to do this (and are not using a domain-scoped token, in which case this will imply the domain), then Keystone will return an error. This error can sometimes be a bit confusing—“Request Unauthorized” (HTTP error code 401), when one might expect it to return a “Bad Request” (HTTP error code 400).

You can’t move users between domains

As described earlier, with multi-domain LDAP, when Keystone encounters a user it will assign it a unique external ID it will publish to the rest of OpenStack (and itself, for assigning roles). This unique ID is effectively a hash of the domain and the local ID within the respective LDAP server). Hence, if you change the local ID or “move” that user to an LDAP server that is mapped to a different Keystone domain, as far as

Keystone (and the rest of OpenStack) is concerned, this is a different user. None of the old assignments or project relationships will be carried across.

Occasional maintenance of the directory-mapping table

As we mentioned earlier, with multi-domain LDAP, Keystone dynamically builds the directory mapping of the external published user and group IDs to the underlying local entities in the appropriate corporate directory service. Since these corporate directories are usually read-only from a Keystone perspective, any user management will happen out-of-band via the tooling used to maintain the corporate directory in question. In practice, this means that over time as user and group entities are deleted out of the corporate directory, old mapping entries will remain in Keystone's directory-mapping table. While this is benign, over time it could bloat the Keystone table. To manage this, the `keystone-manage` tool has been enhanced to support several options for purging entries out of the Keystone directory-mapping table. The most specific option is to purge the mapping for a given local identifier in a given domain. For example:

```
$ keystone-manage mapping_purge --domain-name CustomerA --local-id abc@de.com
```

If this is a relatively frequent occurrence, then you might want to automate such individual purging.

An alternative, however, is to simply purge all the mappings periodically for a given domain. A concern might be whether this will cause you to lose the externally published ID for local entities. In fact, that's not an issue, since the algorithm for the mapping is designed such that Keystone can regenerate the mapping and guarantee to reassign the *same* externally published user/group ID as before. Note that this assumes a user has not moved between corporate directories; if they have, then they are considered a different user. To purge all the mappings for a given domain:

```
$ keystone-manage mapping_purge --domain-name CustomerA
```


CHAPTER 5

Federated Identity

As Keystone progressed with new features and functionality, one of the most sought-after features was support for federated identity: giving administrators the ability to leverage their existing federated identity solution, allowing users to use their own credentials and to leverage existing single sign-on capabilities. One of the key benefits of this approach is that users can exist in an identity store that is not owned or accessible by Keystone, reducing the logic of maintaining user credentials, or supporting a single type of corporate backend (LDAP and not AD). It also allows Keystone to concentrate on authorization and how to better serve OpenStack services.

A brief overview of the terminology used in this chapter:

Identity Provider (IdP)

A trusted provider of identity information. This may be backed by LDAP, SQL, AD, or Mongo. The identity provider is able to construct user attributes into Assertions or Claims.

Service Provider (SP)

A service that consumes identity information. In our case, Keystone will be the service provider. Many web applications that utilize single sign-on are examples of service providers.

SAML

An XML-based standard approach to federated identity. The resultant document that contains user attributes is called an Assertion.

OpenID Connect

A newer standard approach to federated identity. Leverages OAuth 2.0, and ditches XML in favor of JSON. The resultant information about the user is called a Claim.

Assertions and Claims

A standard method of representing information and attributes about a user. An identity provider may issue Assertions or Claims, based on the standard they use, for service providers to consume.

5.1 Approach to Federation

In this section we discuss how existing technology can be leveraged and Keystone-specific resources that need to be created.

5.1.1 Leveraging Existing Technology

Keystone's approach to Federation was based on its decision to run Keystone under the Apache HTTP Server (httpd). Since Apache (and other HTTP servers) is the recommended way to run Keystone, it made sense to architect a solution that allowed Keystone to leverage existing Apache plugins and modules that handle different federation standards for it. For instance, if you intend to use a SAML identity provider, then `mod_shib` or `mod_auth_mellon` are available plugins; if you intend to use an OpenID Connect identity provider, then `mod_auth_openidc` is available. This approach makes Keystone a service provider that is protocol agnostic, since the web server plugins will handle the logic of interacting with the identity provider. The result of a correct setup is that once a user successfully authenticates with their identity provider, Keystone is able to see user attributes as HTTP request environment variables.

One of the difficulties with this approach, and so many of the federated identity standards, is that they are primarily designed to be used in an Internet browser. The reason for this is that any form of authentication can be used, password, two-factor, certificate, etc., so long as the browser can support it and handle the inherent redirects that come with browser-based authentication, then all is good. The issue with using this approach for OpenStack is that so many users of OpenStack interact with their cloud through a command line interface (CLI). Within the standards, there are various levels of support for a non-browser-based flow, but these are often not as user friendly, and may even be disabled or unsupported by the identity provider.

5.1.2 Keystone-Specific Federation Concepts

In order to have a full federated identity solution, the Keystone team needed to create a new set of APIs, and resources stored in their own backends. These APIs are intended to allow cloud administrators to add trusted identity providers, specify which protocols the identity providers will support, and create mappings to allow translating user attributes to keystone concepts that model the external identity providers and protocols they will be consuming.

Identity Provider

Keystone now has an Identity Provider resource, which is served through the following API: `/OS-FEDERATION/identity_providers`. This endpoint allows a cloud administrator to create, delete, update, retrieve, and list identity providers. Each entry is meant to map to an external identity provider that will be trusted to send assertions or claims about an external user. Note that identity provider names are guaranteed to be globally unique in an OpenStack deployment.

Protocol

Keystone now has a Protocol resource, which is served at the following API: `/OS-FEDERATION/identity_providers/{idp_id}/protocols`. This endpoint allows a cloud administrator to create, delete, update, retrieve, and list protocols *per* identity provider. A protocol has only a name and reference to the mapping that will be used (more on that later). The reasoning behind setting the protocol per identity provider is that an external identity provider may support more than one type of federated authentication, so we must allow an identity provider to have multiple protocols as well. The protocols are meant to mirror the federated identity standard that is being used. Common values for protocol names are `saml2` (for SAML) and `oidc` (for OpenID Connect).

Mapping

Mappings are the lynchpin to federated identity in Keystone. A mapping resource has now been created and served at the following API: `/OS-FEDERATION/mappings`. This endpoint allows a cloud administrator to create, delete, update, retrieve, and list mappings. A mapping is specified when creating a protocol that an identity provider supports. The actual contents of a mapping is a set of rules that will be evaluated when a Claim or Assertion is seen by Keystone. These rules will vary greatly between deployments, and depend on the attributes issued by the identity provider and how much access the cloud administrator wants to give to federated users.

5.2 Translating User Attributes to Keystone Concepts

Now that identity providers, protocols, and mappings have been briefly covered, in this section we discuss what assertions about the incoming user look like, how the mapping engine evaluates these assertions, and the rules used by the mapping engine. Let's start by showing an OpenID Connect Claim and a SAML assertion, and how each will have its content transformed into HTTP headers by the web server plugins.

5.2.1 OpenID Connect Claims

An OpenID Connect claim represents information about a user in JSON. There are a number of required fields and values, such as the issuer, the subject, when the claim was issued, and when it will expire. The following is an example OpenID Connect claim after a user has authenticated with an OpenID Connect Identity Provider.

```
{  
    "nonce": "9c2b51b0ef3849828d6fedc8019eae2b",  
    "sub": "stevemar@ca.ibm.com",  
    "at_hash": "RC3QNVOT1h31ofNyFlQskQ",  
    "iss": "https://localhost:8020/oidc/v10/providers/OP",  
    "uniqueSecurityName": "emailaddress=stevemar@ca.ibm.com,c=ca...es,o=ibm.com",  
    "realmName": "w3",  
    "groupIds": [  
        "cn=SWG_Canada,ou=memberlist,ou=ibmgroups,o=ibm.com",  
        "cn=Toronto_Lab_VPN,ou=memberlist,ou=ibmgroups,o=ibm.com",  
        "cn=IBM Regular Employees Canada,ou=memberlist,ou=ibmgroups,o=ibm.com"  
    ],  
    "exp": 1395358515,  
    "iat": 1395354915,  
    "aud": "rp"  
}
```

Looking at the claim above, we can see various fields specific to the authenticated user and the identity provider. The Issuer (`iss`) represents the Identity Provider that issued the claim, the Subject (`sub`) represents the user that authenticated, and most importantly we have the `groupIds`. Note that `sub` and `iss` (and `nonce`, `exp`, `iat`, and `aud`) are required to exist, as per the OpenID Connect specification. The specification also allows for identity providers to add extra fields; in this case, our Identity Provider was backed by our internal bluepages LDAP, and added `groupIds`; this field represents a list of group CNs the user is a member of. How does this look to Keystone?

```
HTTP_OIDC_ISS="https://localhost:8020/oidc/v10/providers/OP"  
HTTP_OIDC_SUB="stevemar@ca.ibm.com"  
HTTP_GROUP_IDS="cn=SWG_Canada,ou=memberlist,ou=ibmgroups,o=ibm.com;  
                cn=Toronto_Lab_VPN,ou=memberlist,ou=ibmgroups,o=ibm.com;  
                cn=IBM Regular Employees Canada,ou=memberlist,ou=ibmgroups,  
                o=ibm.com"
```

The above is a representation of how the claims are seen by Keystone. Each claim will be transformed into an HTTP header and placed in the request back to Keystone. Most of these are straight translations, but in the case of a list of groups, it is common to see either commas or semicolons used to separate the items in the list.

5.2.2 SAML Assertions

A SAML assertion is an XML representation of an authenticated user. Similar to OpenID Connect, there are a number of required fields pertaining to the issuer, the subject, their authorization on the identity provider, and when the assertion was made. The following is a significantly simplified example SAML assertion that was generated after a user has authenticated with an SAML Identity Provider.

```
<saml2:Assertion IssueInstant="2014-06-11T13:05:28.891Z">
    <saml2:Issuer>
        https://bluepages.ibm.com
    </saml2:Issuer>
    <saml:AttributeStatement>
        <saml2:Attribute Name="UserName">
            <saml2:AttributeValue>stevemar@ca.ibm.com</saml2:AttributeValue>
        </saml2:Attribute>
        <saml:Attribute Name="Role">
            <saml:AttributeValue>IBM Regular Employ...anada</saml:AttributeValue>
        </saml:Attribute>
        <saml:Attribute Name="Role">
            <saml:AttributeValue>SWG Canada</saml:AttributeValue>
        </saml:Attribute>
        <saml:Attribute Name="Role">
            <saml:AttributeValue>Toronto Lab VPN</saml:AttributeValue>
        </saml:Attribute>
    </saml:AttributeStatement>
</saml2:Assertion>
```

Looking at the assertion above, we can see various fields specific to the authenticated user and identity provider. The main portion of the assertion we are interested in are the **Attributes** in the **AttributeStatement** section. This indicates the level of authorization the user has on that identity provider. These field values will be used to construct an effective mapping. How does this look to Keystone?

```
HTTP_ISSUSER="https://bluepages.ibm.com"
HTTP_USERNAME="stevemar@ca.ibm.com"
HTTP_ROLE="IBM Regular Employees Canada;SWG Canada;Toronto Lab VPN"
```

Similar to the example above, the information that is returned to Keystone should have a very similar structure when SAML is used as opposed to OpenID Connect; this demonstrates a Keystone protocol-agnostic approach to federating identities.

5.2.3 The Mapping Engine

Another detail about Keystone's approach to federation is that users coming from identity providers are ephemeral, meaning they will not exist or be stored anywhere on a Keystone database. This decision was made because, if users were to be added or removed, then the databases would need to be updated accordingly, rendering our goal (to eliminate user management in Keystone) moot. The impact of this decision is that the individual ephemeral users will not have a sense of authorization on the cloud, since we would not be able to assign the user a role or a project. To address this issue, mappings were created to convey a sense of where these external entities belong in a Keystone cloud.

The goal of a mapping is to give the ephemeral user just enough authorization in the cloud to perform a task, and also to ensure we have enough data about the user to populate token details, and create auditable information. To accomplish this task, the Keystone team decided to leverage User Groups that are already present in the Identity backend, since groups may have role assignments. Using the information in the claim or assertion, a Keystone administrator should be able to create a mapping that will give the ephemeral user access to a specific part of the cloud.

The overall concept of the input and output of the mapping rules can be seen below.

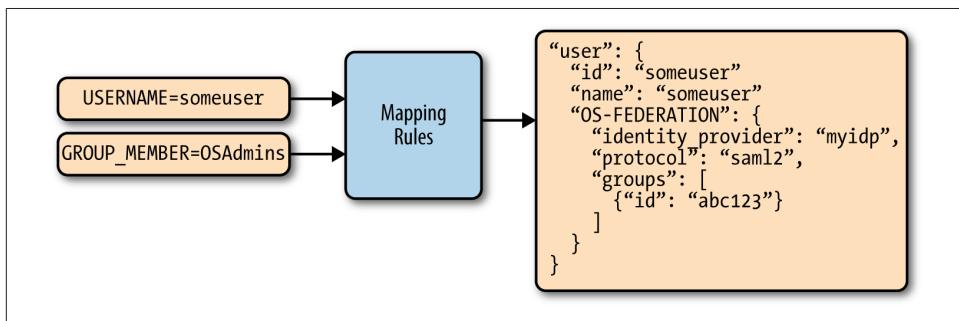


Figure 5-1. Assertions as HTTP headers are sent to the mapping engine with the mapping rules for the identity provider and protocol, and the result is enough information to include in a token.

On the left-hand side of the figure are the HTTP request environment variables, as seen by Keystone; additional examples of these environment variables can be seen in the previous section. Using the mapping rules associated with a specific identity provider and protocol, we are able to determine suitable groups to give the user membership to, and also determine a suitable username and user ID to populate the resulting token and any auditable events.

5.2.4 Mapping Rules

In this section, we will discuss how the user attributes, represented as HTTP requests headers, can be translated into Keystone concepts. To accomplish this, we select a mapping that contains a set of rules that will be invoked when requests come from a specific identity provider.

So let's say we are an administrator at Acme Inc., and we want to use cloud resources at IBM. When authenticated with our external identity provider, the following user attributes are returned, represented by HTTP headers:

```
HTTP_ISSUSER="https://acmepages.acme.com"
HTTP_USERNAME="topol"
HTTP_FULL_NAME="Brad Topol"
HTTP_EMAIL="brad@acme.com"
HTTP_ROLE="Acme IT Dept; IT Administrators; Regular Employees; Non-Contractors"
```

So, when the cloud administrator at IBM is creating a mapping, what should it look like for users authenticating from Acme Inc.? The answer depends on just how much access the cloud administrator wants to give! Mapping rules are made up of a `local` and `remote` section; the `local` section represents the Keystone resource the users will be mapped to, and the `remote` section represents the user attributes in the HTTP header. Below is a sample mapping that can be used to federate user identities from Acme to Keystone.

```
{
  "rules": [
    {
      "local": [
        "user": {
          "name": "{0}"
        }
      ],
      "remote": [
        {
          "type": "HTTP_USERNAME"
        }
      ]
    },
    {
      "local": [
        {
          "group": {
            "name": "acme_federated_users",
            "domain": {
              "name": "acme_private_cloud"
            }
          }
        }
      ],
      "remote": [
        {
          "type": "HTTP_EMAIL"
        }
      ]
    }
  ]
}
```

```

    "remote": [
      {
        "type": "HTTP_ROLE",
        "any_one_of": [
          "Acme IT Dept",
          "IT Administrators"
        ]
      }
    ]
  }
}

```

Different mapping rules can be used to accomplish the same goal. Below is another example of how we can federate user identities from Acme to Keystone.

```

{
  "rules": [
    {
      "local": [
        {
          "user": {
            "name": "{0}"
          },
          "groups": {
            "name": "{1}",
            "domain": {
              "id": "0cd5e9"
            }
          }
        }
      ],
      "remote": [
        {
          "type": "HTTP_USERNAME"
        },
        {
          "type": "HTTP_ROLES",
          "blacklist": [
            "Regular Employees",
            "Contractors"
          ]
        }
      ]
    }
  ]
}

```

Assuming Keystone receives the user attributes in HTTP headers (like the example above), what would happen if the first mapping was used? In this case, a Keystone token will be generated with Brad's username (topol); if no ID is specified, then it'll use a URL-safe version of the username. Furthermore, the Keystone token will

include a list of groups Brad is successfully matched with, which includes the group ID for the Keystone group “`acme_federated_users`” in the domain “`acme_private_cloud`”. Brad matched the criteria for mapping into that group since he has any of the roles: “Acme IT Dept” and “IT Administrator.” Note that the term *any_one_of* is important here, since it means that as long as a single match occurs, the user has met the criteria.

In the second example, we reuse the same rule for choosing the user’s name and ID. The changes are in how the groups are processed. In this example, the local section attempts to use a direct lookup (using the `{0}` value initiates a direct lookup) for all incoming values in the `HTTP_ROLES` list. The only catch is that the group names listed in the blacklist are not looked up.

5.3 Authentication Flow: What’s It Look Like?

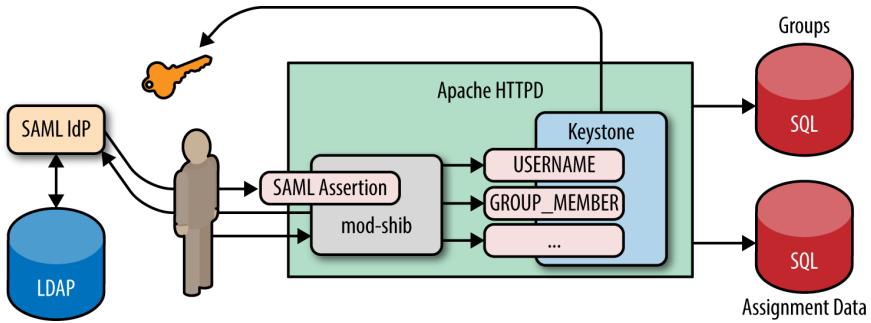
Now that we’ve got all the background, what does the full authentication flow look like? We will go through all the steps that are performed in a federated authentication request in this section. Note that to initiate an authentication request, the Apache module used must protect a specific path. We recommend the path be in the following style:

```
GET/POST /OS-FEDERATION/identity_providers/{identity_provider}  
/protocols/{protocol}/auth
```

This allows each identity provider to have a separate authentication flow per supported protocol, and gives Keystone enough information to find a mapping to be used for the combination.

Federated authentication flow:

1. A user accesses the protected URL above, which triggers the apache module (`mod_shib` in this case) to redirect the user to authenticate with the external identity provider (through a web page or otherwise).
2. Once the user has authenticated with their identity provider, the identity provider returns user attributes (as a SAML assertion or OpenID Connect claim) to the Apache plugin. The Apache module then transforms the user attributes into HTTP request headers and passes them along to Keystone.
3. Keystone then determines which mapping to use, based on the identity provider and protocol, invokes the mapping engine, and, if there is enough information to construct a token (user information and group IDs), then an unscoped token is issued!
4. (Not included in the diagram) The user can then use the unscoped token to look up the projects they are allowed to access and request a scoped token for that project, using their federated unscoped token.



5.4 Single Sign-On

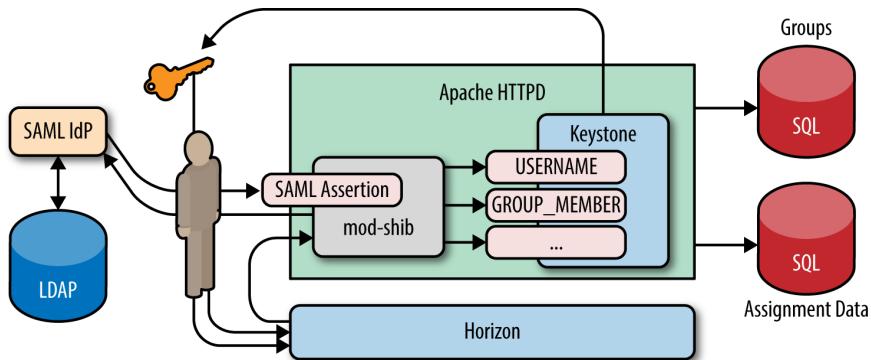
One of the most demanded features in OpenStack was to support single sign-on access. Configuring Keystone for federating identities does not imply that Keystone will have single sign-on support automatically. This is problematic because Horizon maintains the “login” page for OpenStack and runs as a separate process. To solve this issue, the Keystone and Horizon teams worked together to create a solution whereby Keystone and Horizon are able to communicate and trust each other. The result is that end users will see a familiar login page, which is important for branding. But most importantly, Keystone does not see any sensitive user information (passwords), as was the case in the SQL and LDAP-based backends.

To enable this option in Horizon, certain options need to be enabled in Horizon’s configuration file. These options and more details about this work will be further highlighted in [“5.6 A Practical Guide to Setting Up SSO with Google”](#) on page 93, including screenshots. For this section, we will simply outline the steps pointing out where the differences happen when compared to the previous flow.

Single Sign-On Flow

1. A user hits the Horizon landing page. Upon loading the page, the user is given a list of methods to authenticate. This list will include a method to authenticate by username and password for service accounts, but also includes methods for authenticating by their protocol, or identity provider and protocol, depending on how Horizon is set up.
2. Once the user has selected a federation flow, a redirect to the external identity provider is made, and the user is redirected to their single sign-on page.
3. At this point, the steps remain the same. The user attributes are sent from the identity provider to Keystone, and the mapping engine is invoked and returns a token.
4. Where the process differs is what Keystone does with the token: it must post back the token ID to Horizon. This is done using JavaScript.

- Once Horizon has the token, it creates an unscoped client session and the user is logged in.



5.5 A Practical Guide to Federating Identities for IBM WebSphere Liberty and Bluepages

In this guide, we will reuse the Bluepages LDAP server as our identity store. However, we will also use WebSphere Liberty as our Identity provider, since it has capabilities to act as an OpenID Connect provider.

5.5.1 Download, Install, and Configure IBM WebSphere Liberty

To proceed with this section, adding WebSphere Liberty to your environment is necessary. IBM's WebSphere Liberty runtime and its extensions are available for free. To download the runtime, navigate to <https://developer.ibm.com/wasdev/downloads/liberty-profile-using-non-eclipse-environments/>.

Once the jar files have been installed, we need to use the built-in feature manager to install the OpenID Connect Server feature.

```
$ ./wlp/bin/featureManager install openidConnectServer-1.0
```

Before we configure WebSphere Liberty to run as an OpenID Connect provider, we need to create a server.

```
$ ./wlp/bin/server create oauthServer
```

We can now configure the server to point to the Bluepages LDAP backend. The following is an example configuration in `./wlp/usr/servers/oauthServer/server.xml`.

```
<server>
  <featureManager>
    <feature>openidConnectServer-1.0</feature>
    <feature>ssl-1.0</feature>
```

```

<feature>appSecurity-2.0</feature>
<feature>servlet-3.0</feature>
</featureManager>

<ldapRegistry id="bluepages" realm="w3" host="bluepages.ibm.com" port="389"
    ignoreCase="true" baseDN="o=ibm.com"
    ldapType="IBM Tivoli Directory Server" >
    <idsFilters
        userFilter="(&(emailAddress=%v)(objectclass=person))"
        groupFilter="(&(cn=%v)(objectclass=groupOfUniqueNames))"
        userIdMap="*:emailAddress"
        groupIdMap="*:cn"
        groupMemberIdMap="groupOfUniqueNames:uniqueMember" />
</ldapRegistry>

<keyStore id="defaultKeyStore" password="insecurePass"/>
<httpEndpoint host="localhost" httpPort="9080" httpsPort="9443"
    id="defaultHttpEndpoint"/>

<oauth-roles>
    <authenticated>
        <special-subject type="ALL_AUTHENTICATED_USERS" />
    </authenticated>
</oauth-roles>

<openidConnectProvider id="OP" oauthProviderRef="Oauth" />

<oauthProvider id="Oauth" >
    <localStore>
        <client name="rp" secret="LDo8LTor"
            displayName="rp"
            introspectTokens="true"
            redirect="https://localhost:9443/oauthclient/redirect.jsp"
            scope="openid profile email phone address"
            preAuthorizedScope="openid profile"
            enabled="true"/>
    </localStore>
</oauthProvider>
</server>

```

The configuration above does the following:

1. Using the `ldapRegistry` option, we point it to our Bluepages LDAP with a user and group filter, not unlike the ones we would provide for Keystone.
2. The `oauthProvider` option specifies the OAuth client ID and client secret. These are `rp` and `LDo8LTor`, respectively. It also specifies which port the process is run on. In this case it's 9443.
3. We're disabling SSL for now in the `keyStore` option to make things easy to test.

Endpoints are specified in Liberty's documentation, but we'll outline them here briefly:

- The token endpoint is: /oidc/endpoint/OP/token
- The introspection endpoint is: /oidc/endpoint/OP/introspect

Now let's start the server to test if the configuration is valid.

```
$ ./wlp/bin/server start oauthServer
```

We can quickly check to see if we can authenticate with the OpenID Connect Provider by providing our own username and password and the client ID and secret. The output should be an access token.

```
$ curl -k -H "Content-Type: application/x-www-form-urlencoded; charset=UTF-8" -d \
"grant_type=password&client_id=rp&client_secret=LDo8LTor&username=stevemar%40ca
.ibm.com&password=mySecretPassword&scope=profile" https://localhost:9443/oidc/
endpoint/OP/token

{
  "access_token": "kjDVDz7vyUp4maswQbeoaHjN42WFXNSMdBkos1VJ",
  "token_type": "Bearer",
  "expires_in": 7200,
  "scope": "profile",
  "refresh_token": "1tqvETSuEo9oiBBCbsabjfY0W1fyvbpH5rTTf3i6979BEsD0p4"
}
```

Optionally, we can immediately retrieve an OpenID Connect. We can exchange the access token for an OpenID Connect claim, which contains information about the authenticated user, via the introspection endpoint. To do this we supply the client ID and secret and our newly acquired access token.

```
$ curl -k --user rp:LDo8LTor \
https://localhost:9443/oidc/endpoint/OP/introspect?token=kjDVDz7vyUp4maswQbeoaH
jN42WFXNSMdBkos1VJ

{
  "exp": "1433979091514600",
  "realmName": "w3",
  "sub": "stevemar@ca.ibm.com",
  "scope": "profile",
  "grant_type": "resource_owner",
  "uniqueSecurityName": "uid=123456789,c=ca,ou=bluepages,o=ibm.com",
  "active": "true",
  "client_id": "rp",
  "token_type": "Bearer",
  "iat": "1433979091511000",
  "groupIds": "[cn=IBM_Regular_Empl...Canada,ou=memberlist,ou=ibmgroups,o=ibm.com,
               cn=SWG_Canada,ou=memberlist,ou=ibmgroups,o=ibm.com,
               cn=Toronto_Lab_VPN,ou=memberlist,ou=ibmgroups,o=ibm.com]"
}
```

5.5.2 Configuring Keystone to Use OpenID Connect

Now we must configure Keystone to be able to use the new OpenID Connect provider we created. We have to modify Keystone's configuration file to enable federated authentication, create the identity provider, protocol, and mapping in Keystone, and lastly we must protect a specific URL to trigger the authentication.

The first (and essential) step in configuring Keystone to use OpenID Connect is to install the Apache Httpd plugin `mod_auth_openidc`. Several packages are available at https://github.com/pingidentity/mod_auth_openidc/releases.

```
$ sudo apt-get install libjansson4 libhiredis0.10 libcurl3
$ sudo dpkg -i libapache2-mod-auth-openidc_1.8.3-1_amd64.deb
```

Next, we must update `/etc/keystone/keystone.conf` to support OpenID Connect as an authentication mechanism. In the `[auth]` section set the following:

```
methods = external,password,token,oidc
oidc = keystone.auth.plugins.mapped.Mapped
```

In the `[federation]` section set the following:

```
remote_id_attribute = HTTP_OIDC_ISS
```

We also need to update Keystone's virtual host file and add in new OpenID Connect entries to `/etc/apache2/sites-available/keystone.conf`. The new settings are highlighted below.

```
LoadModule auth_openidc_module /usr/lib/apache2/modules/mod_auth_openidc.so

<VirtualHost *:5000>
    ...
    SetEnv HTTP_OIDC_ISS bluepages
    OIDCOAuthIntrospectionEndpoint "https://localhost:9443/oidc/
        endpoint/OP/introspect"
    OIDCOAuthIntrospectionTokenParamName token
    OIDCOAuthRemoteUserClaim sub
    OIDCOAuthClientID rp
    OIDCOAuthClientSecret LD08LTor
    OIDCOAuthSSLValidateServer Off
    OIDCClaimDelimiter ";"
    OIDCClaimPrefix "OIDC-"

    <LocationMatch "/v3/OS-FEDERATION/identity_providers/bluepages/
        protocols/oidc/.*?">
        AuthType oauth20
        Require valid-user
        LogLevel debug
    </LocationMatch>
</VirtualHost>
```

We are essentially setting the protected URL, in the `LocationMatch` header, and setting various OpenID Connect properties.

Now that we've updated Keystone's configuration file and virtual host file, we need to create a local Keystone group so that users authenticating via federation can be mapped into that group. In this case, we create a group called `federated_users` with the role of `member` on the project `demo`.

```
$ openstack group create federated_users  
$ openstack role add member --group federated_users --project demo
```

Before we create the identity provider, protocol, and mapping entries in Keystone, we can create a file that contains the mapping so as to make our CLI interactions easier. Create the file `mapping.json`, and remember to update the contents of the local section with the correct ID from the previous command.

```
[  
  {  
    "local": [  
      {  
        "group": {  
          "id": "26daaf649d6a46b998f631da79759230"  
        }  
      }  
    ],  
    "remote": [  
      {  
        "type": "HTTP_OIDC_GROUPIDS",  
        "any_one_of": [  
          "cn=SWG_Canada*"  
        ],  
        "regex": true  
      }  
    ]  
  }  
]
```

The result of this mapping will be that any user that is a member of the LDAP group `SWG_Canada` will be placed into the Keystone group `federated_users`.

The last step is to tie all the pieces together and create the Keystone federation resources; let's create the identity provider, the mapping, and the protocol.

```
$ openstack identity provider create bluepages --remote-id bluepages  
$ openstack mapping create ibm_mapping --rules mapping.json  
$ openstack federation protocol create oidc --identity-provider bluepages \  
  --mapping ibm_mapping
```

5.5.3 Testing It All Out

Using the Python-keystoneclient library, we can create a session that uses the OpenID Connect auth plugin. We will create a sample file that can be run to create an unscoped token. We'll call the file *test_oidc.py*.

```
from keystoneclient.contrib.auth.v3 import oidc
from keystoneclient import session

AUTH_URL = 'http://localhost:5000/v3'
IDENTITY_PROVIDER = 'bluepages'
PROTOCOL = 'oidc'
USER_NAME = 'stevemar@ca.ibm.com'
PASSWORD = 'boogabooga'
CLIENT_ID = 'rp'
CLIENT_SECRET = 'LDo8LTor'
ACCESS_TOKEN_ENDPOINT = 'https://localhost:9443/oidc/endpoint/OP/token'

def main():
    oidc_plugin = oidc.OidcPassword(
        AUTH_URL,
        IDENTITY_PROVIDER,
        PROTOCOL,
        username=USER_NAME,
        password=PASSWORD,
        client_id=CLIENT_ID,
        client_secret=CLIENT_SECRET,
        access_token_endpoint=ACCESS_TOKEN_ENDPOINT)
    s = session.Session(verify=False)
    access_info = oidc_plugin.get_unscoped_auth_ref(s)
    print(access_info)

if __name__ == '__main__':
    main()
```

When ran, the output will be an unscoped Keystone token that contains information about the federated user and the groups they have access to:

```
$ python test_oidc.py
{
    u'extras': {},
    u'methods': [u'oidc'],
    u'version': 'v3',
    u'audit_ids': [u'F4FbnLlaQ_SsU4xrD_jE7w'],
    u'issued_at': u'2015-06-11T07:36:00.337835Z',
    u'auth_token': '87c736b04338482a94cf951ee0798a76',
    u'expires_at': u'2015-06-11T08:36:00.337487Z',
    u'user': {
        u'OS-FEDERATION': {
            u'identity_provider': {u'id': u'bluepages'},
            u'protocol': {u'id': u'oidc'},
            u'groups': [{u'id': u'26daaf649d6a46b998f631da79759230'}]
```

```

        },
        u'domain': {u'id': u'Federated', u'name': u'Federated'},
        u'id': u'stevemar%40ca.ibm.com',
        u'name': u'stevemar@ca.ibm.com'
    }
}

```

5.6 A Practical Guide to Setting Up SSO with Google

In this guide, we will discuss how to set up Keystone and Horizon to allow for single sign-on. In this case, use Google as the Identity Provider and allow users with valid Google accounts to federate into our Keystone deployment.

5.6.1 Configure Keystone to Use OpenID Connect

Similar to “[5.5.2 Configuring Keystone to Use OpenID Connect](#)” on page 90, we need to configure Keystone to use an OpenID Connect provider. First, we install the Apache Httpd plugin `mod_auth_openidc`. Several packages are available at https://github.com/pingidentity/mod_auth_openidc/releases.

```
$ sudo apt-get install libjansson4 libhiredis0.10 libcurl3
$ sudo dpkg -i libapache2-mod-auth-openidc_1.8.3-1_amd64.deb
```

Next, we must update `/etc/keystone/keystone.conf` to support OpenID Connect as an authentication mechanism. In the `[auth]` section set the following:

```
methods = external,password,token,oidc
oidc = keystone.auth.plugins.mapped.Mapped
```

In the `[federation]` section, we must set a list of trusted dashboards, which is the fully qualified domain name of the host that is running Horizon.

```
remote_id_attribute = HTTP_OIDC_ISS
trusted_dashboard = http://sso-demo.test.ibmcloud.com/auth/webss/
```

We also need to update Keystone’s virtual host file and add in new OpenID Connect entries to `/etc/apache2/sites-available/keystone.conf`. The new settings are highlighted below.

```
LoadModule auth_openidc_module /usr/lib/apache2/modules/mod_auth_openidc.so

<VirtualHost *:5000>
    ...
    OIDCClaimPrefix "OIDC-"
    OIDCResponseType "id_token"
    OIDCScope "openid email profile"
    OIDCProviderMetadataURL "https://accounts.google.com/.well-known/
        openid-configuration"
    OIDCClientID "78026256901-9oice6ionj19voicnrii2lfl86a1i4rp.apps.
        googleusercontent.com"
    OIDCClientSecret d6TiBYA3qQUuzlR91Q-YzpJA
```

```

OIDCCryptoPassphrase openstack
OIDCRedirectURI "http://sso-demo.test.ibmcloud.com:5000/v3/auth/
                  OS-FEDERATION/webssso/oidc/redirect"

<Location ~ "/v3/auth/OS-FEDERATION/webssso/oidc">
    AuthType openid-connect
    Require valid-user
    LogLevel debug
</Location>
</VirtualHost>

```

We are essentially setting the protected redirect URL, in the `LocationMatch` header, and setting various OpenID Connect properties. Note that the `OIDCClientID` and `OIDCClientSecret` come from a Google developer project.

Now that we've updated Keystone's configuration file and virtual host file, we need to create a local Keystone group so users authenticating via federation can be mapped into that group. In this case, we create a group called `federated_users` with the role of member on the project `demo`.

```

$ openstack group create federated_users
$ openstack role add member --group federated_users --project demo

```

Before we create the identity provider, protocol, and mapping entries in Keystone, we can create a file that contains the mapping so as to make our CLI interactions easier. Create the file `mapping.json`, and remember to update the contents of the local section with the correct ID from the previous command.

```

[
  {
    "local": [
      {
        "group": {
          "id": "4e48b0c139c94cf088a665d919ee10f1"
        }
      }
    ],
    "remote": [
      {
        "type": "HTTP_OIDC_ISS",
        "any_one_of": [
          "https://accounts.google.com"
        ]
      }
    ]
  }
]

```

The result of this mapping will be that any user that comes from Google, essentially having the issuer set to `https://accounts.google.com`, will be placed into the Keystone group `federated_users`.

The last step is to tie all the pieces together and create the Keystone federation resources. Let's create the identity provider, the mapping, and the protocol.

```
$ openstack identity provider create google \
    --remote-id https://accounts.google.com
$ openstack mapping create google_mapping --rules mapping.json
$ openstack federation protocol create oidc --identity-provider \
    google --mapping google_mapping
```

We must also copy the *sso_callback_template.html* file to */etc/keystone/*. This is simply a small script that posts an unscoped token back to Horizon. It is necessary since the original location of Horizon is lost in all the redirects. The script itself may be overridden by the deployer. What is provided by Keystone is the minimum needed.

```
/$ cp /opt/stack/keystone/etc/sso_callback_template.html /etc/keystone
```

5.6.2 Configure Horizon for Single Sign-On

The last step is to set a few values in Horizon's configuration file, located at *horizon/openstack_dashboard/local/local_settings.py*.

```
OPENSTACK_KEYSTONE_URL = "http://sso-demo.test.ibmcloud.com:5000/v3"
OPENSTACK_API_VERSIONS = {
    "identity": 3
}
WEBSSO_ENABLED = True
WEBSSO_CHOICES = (
    ("credentials", _("Keystone Credentials")),
    ("oidc", _("OpenID Connect"))
)
WEBSSO_INITIAL_CHOICE = "credentials"
```

The OPENSTACK_KEYSTONE_URL and OPENSTACK_KEYSTONE_URL options are not specific to implementing single sign-on, but are necessary to be changed to point to v3 of the Identity API. These values indicate the authentication URL to use for Keystone and which version to use.

The WebSSO options are fairly straightforward, with WEBSSO_ENABLED being the main option to enable single sign-on, and WEBSSO_CHOICES being the contents of a drop-down menu. The contents of WEBSSO_CHOICES are broken up into the federation protocol name and the name the user will see on the Horizon landing page. The term **credentials** is a reserved keyword and should be added to the list to allow service and admin accounts stored in SQL to authenticate. Finally, the option WEBSSO_INITIAL_CHOICE sets the default value of the drop-down menu the user selects.

Once completed, restart Apache to update both Keystone and Horizon.

```
$ sudo service apache2 restart
```

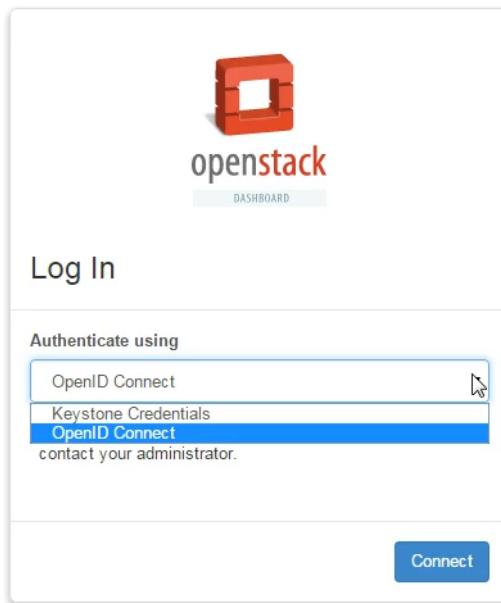
Congratulations! That is the complete guide to configuring both Keystone and Horizon to use single sign-on. In the next subsection, we will step through a visual guide that shows the flow from a user's perspective.

5.6.3 Let's See It with Screenshots!

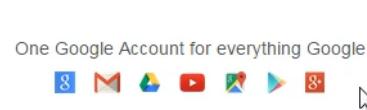
Step 1: We hit the Horizon landing page, with Keystone Credentials selected as the default authentication method. We are still able to authenticate with a username and password of a user in SQL or LDAP.

The screenshot shows the OpenStack Horizon login interface. At the top, there is a logo consisting of a red square with a white 'O' shape inside, followed by the word 'openstack' in lowercase. Below the logo is a 'DASHBOARD' link. The main area is titled 'Log In'. It contains two input fields: 'User Name' and 'Password', each with a corresponding input box. Below these fields is a dropdown menu labeled 'Authenticate using' with the option 'Keystone Credentials' selected. A note at the bottom of the form states: 'If you are not sure which authentication method to use, contact your administrator.' At the bottom right of the form is a blue 'Sign In' button.

Step 2: We proceed by selecting the drop-down menu and choosing the OpenID Connect option. This will cause the username and password fields to be removed. Let's go ahead and opt to Sign In.



Step 3: Assuming Keystone and Horizon are properly configured, we should be redirected to a Google-branded sign in page. Simply authenticate with Google with your usual username and password, even if multi-factor authentication is enabled—it doesn't matter, as long as you can authenticate. The Apache modules will be able to read the incoming OpenID Connect claim and set user attributes as HTTP headers.



Step 4: Once Keystone invokes the correct mapping engine and returns a token back to Horizon, the user should be able to log in to view and perform tasks as normal.

A screenshot of the Horizon dashboard under the 'openstack' project. The left sidebar shows navigation links for Project, Admin, System, and Identity. The main content area is titled 'Overview' and contains a 'Usage Summary' section. It prompts the user to 'Select a period of time to query its usage:' with 'From: 2015-05-01' and 'To: 2015-05-13' fields, a 'Submit' button, and a note about date format. Below this, it displays 'Active Instances: 0 Active RAM: 0 Bytes This Period's VCPU-Hours: 0 This Period's GB-Hours: 0 This Period's RAM-Hours: 0'. There is also a 'Download CSV Summary' link. The 'Usage' section has a table header with columns: Project Name, VCPUs, Disk, RAM, VCPU Hours ⓘ, Disk GB Hours ⓘ, and Memory MB Hours ⓘ. The table body is currently empty, showing 'Displaying 0 items'.

5.7 Tips, Common Pitfalls, and Troubleshooting

Ensure All Libraries Are Installed

Similar to LDAP, the federation capabilities of Keystone require extra dependencies that are not installed by default. This includes the Python library pysaml2 and the package xmlsec1. To install these on Ubuntu, use the following commands:

```
$ sudo pip install pysaml2  
$ sudo apt-get install xmlsec1
```

Known Limitations of Social Media Logins

In our earlier example of authenticating via OpenID Connect and a command line, we were leveraging the “Resource Owner Password Credential” flow of the OpenID Connect specification. In our example of WebSphere Liberty acting as an Identity Provider, that particular flow is enabled. Most social media logins will disable this flow, since they do not want their users to enter passwords into a non-branded or trusted medium.

Using SAML from the Command Line

Users are able to authenticate via SAML using the command line as well as a browser. The Identity Provider must be configured to expose the Enhanced Client or Proxy (ECP) flow of the SAML specification. Keystone has supported this mode of authentication since the Icehouse release.

CHAPTER 6

Future Work

Keystone has made substantial strides in capabilities over the past few years and has matured significantly. Nonetheless, there are still several areas in which there are opportunities for improvement. In this chapter, we describe several areas that we anticipate will be the focus for future work on Keystone. These include multi-factor authentication, improved Horizon integration for multi-region K2K Federation, replacement of service accounts with X509 certificates, alternative LDAP support models, centralized policies, and integration with other technologies. In essence, we will be continuing on the path of Keystone becoming an interface layer to additional established (and emerging) enterprise-focused identity and authentication capabilities.

6.1 Multi-Factor Authentication

Keystone currently only supports the use of a single authentication method. In certain circumstances, it may be desirable to require multiple methods of authentication to be utilized to ensure authentication requirements are satisfied. The use of multiple authentication mechanisms is typically referred to as multi-factor authentication. With multi-factor authentication, a user not only needs to provide a valid password but also must perform some other form of authentication, such as entering a personal identification number (PIN) that is sent by text message. As OpenStack and Keystone adoption continues to grow, we anticipate that there will be usage environments in which multi-factor authentication is desired.

6.2 Integration with Horizon for Multi-Region Keystone to Keystone Federation Support

Keystone to Keystone (K2K) Federation is a relatively nascent enhancement, and its ease of use increases when its capabilities are more tightly integrated into OpenStack’s dashboard (Horizon). One such opportunity for increased consumability is the addition of multi-region K2K support to Horizon. In this scenario, Horizon is enhanced to support multiple federated OpenStack regions via a pull-down list. As such, the user is provided with a relatively straightforward way of directing Horizon about which region it should be displaying without having to expose to the user the complexity of the multiple Keystones that are working together to provide support for federated regions.

6.3 Using LDAP as a Federated Identity Provider

The LDAP support described in this book uses Keystone’s driver architecture to make the users and groups in the corporate directory accessible via the Keystone API. This works well for environments where users need to be articulated through Keystone. For companies that evolve to a federated model, there is an alternative to using LDAP that might fit better in that environment—basically to use LDAP as a pseudo-IDP. This would work by using something like the `mod_lookup_identity` Apache module to look up the user in LDAP, and then pass that on to Keystone. In this model, none of the LDAP users would appear in Keystone (just like in Federation), and Keystone’s Identity store would just store Groups to which these LDAP users could be mapped (using the Federation mapping capabilities described in [Chapter 5](#)).

6.4 Replacement of Service Users with X.509 Certificates and Barbican Integration

Traditionally, Keystone required the other OpenStack projects to create service user accounts as their means for interacting with and invoking operations on Keystone. The use of service accounts can add complexity to managing an OpenStack environment, as in some enterprises it can problematic to get these fictitious users added to identity-management systems such as LDAP. Moreover, this approach requires a service user password in the Keystone configuration file and a token to be issued.

As an alternative to the service user account approach, it is possible to replace service users with X.509 certificates. There are many advantages to this approach. First, it no longer requires a password. Furthermore, the approach no longer requires a token to be issued because an authorization credential can be directly created by examining the identity data in the X.509 certificate. Finally, this approach increases security for

the services, as the use of X.509 certificates provides the equivalent of a non-bearer token, as the SSL client must possess the corresponding private key.

As Keystone evolves to embrace increased utilization of X.509 certificates, whether they are used to replace service users or used for federation, it will need to worry about the secure storage, provisioning, and management of the X.509 certificates. Fortunately, the mission of the OpenStack Barbican project is to provide this exact set of capabilities. Thus, in the future we anticipate greater integration between the Keystone and Barbican projects.

6.5 Centralized Policy and Distribution

In OpenStack, a key feature of many of the OpenStack services is to use a policy engine to determine access management (this is what the roles in a token are eventually used for). The policy engines rely on policy files to specify the access management policy for the different types of users and the various operations. Currently, the process of distributing these policy files throughout all the OpenStack services running across the cloud is done by operators using an out-of-band mechanism, such as a configuration-management system. Having to perform this activity with a separate tool may be seen by some as unnecessary overhead. As an alternative, Keystone may look into providing its own API for allowing services to obtain these policies directly from Keystone (along with performing associated cache management on the policies). Indeed, some basic APIs for this have been available in Keystone for a number of releases, but are not yet used by any of the services. Having this foundation in place would enable Keystone to start providing more robust policy support—for instance, a common name space across policies for different services where they could share rules. The discussions around this as to how far this support should be taken (and how the services should utilize it) are probably some of the most controversial in Keystone today—and we anticipate substantial future discussions on this topic and would like to invite the audience reading this book to participate.

6.6 Integrating with Other Technologies

In many cloud environments, the OpenStack Infrastructure-as-a-Service is typically combined with a Platform-as-a-Service environment such as IBM Bluemix, which is Cloud Foundry based. In addition, many cloud environments provide some notion of support for Container Technologies such as Docker.

In these types of environments, it is critical to keep the necessary user authentication and authorization as streamlined as possible. For example, if an enterprise customer has already performed the necessary federation to integrate Keystone with their federated identity management system they may want to avoid having to repeat this integration with Bluemix's authentication system. Because both Bluemix and Keystone

support standard federated identity protocols such as OpenId Connect, it would be fairly straightforward to ensure these two technologies integrate together to provide a more seamless authentication and authorization environment for customers.

In a similar fashion, it may be worthwhile to investigate how Keystone's authentication, authorization, and integration support for federated identity management systems could be utilized by Container Technologies such as Docker.

Index

A

access management (authorization), xvii, 12
account locked, 52
Active Directory (AD), ix
 LDAP integration, 45
 LDAP support, 14
actors, 2
 (see also user groups)
aliasing, 53
API, for multi-domain LDAP setup, 58-60
assertions
 and mappings, 79
 defined, 78
 SAML, 81
assignments
 defined, 3
 LDAP problems with, 46, 72
authentication, 9-11
 and federated flow, 85
 and tokens, 11
 as new user, 30
 defined, xvi
 for multi-domain LDAP user, 68, 70-71
 multi-factor, 101
 through certificate, 15
 troubleshooting, 33
 with Horizon, 70-71
 with password, 10
authorization, xvii, 12, 34
 (see also access management)

B

backends, 7, 9, 13
Barbican, 103

Bluemix, 103

Bluepages LDAP server, 87

C

catalogs, 5
certificate, authentication through, 15
chaining, referral chasing vs., 53
claims
 defined, 78
 mappings and, 79
configuration files, multi-domain LDAP using, 57
cryptographic message syntax (CMS), 37
cURL
 assigning role to user for project, 29
 authenticating as new user for project, 30
 creating domains, 27
 creating project within domain, 28
 creating user within domain, 29
 listing domains, 26
 listing groups, 24
 listing projects, 22
 listing roles, 25
 listing users, 21
 obtaining tokens, 19

D

debug option, 34
debugging, 53
dereferencing, 53
DevStack, 17
 (see also OpenStackClient)
directory-mapping table, 75
domains

and authentication, 10, 33
creating, 27
creating project within, 27
creating user within, 28
defined, 2
listing, 26
multiple with LDAP (see multi-domain LDAP)
region vs., 14
targets and, 3
users and, 14, 74

E

ephemeral users, 82

F

federated identity, ix, 77-99
and identity providers, 8
approach to federation, 78
authentication flow, 85
configuring Keystone to use OpenIDConnect, 90-91
integration with Horizon for multi-region Keystone-to-Keystone federation support, 102
Keystone-specific federation concepts, 79
leveraging existing technology for, 78
mapping engine, 82
mapping rules, 83-85
OpenIDConnect claims, 80
SAML assertions, 81
SSO flow, 86
SSO setup with Google, 93-98
tips, pitfalls, and troubleshooting, 99
translating user attributes to Keystone concepts, 80-85
using LDAP as provider, 102
with IBM WebSphere Liberty and Blueprint pages, 87
Fernet tokens, 36, 40
future work, 101-104
Barbican integration, 102
centralized policy and distribution, 103
integration with Horizon for multi-region Keystone-to-Keystone federation support, 102
integration with other technologies, 103
multi-factor authentication, 101

replacement of service users with X.509 certificates, 102
using LDAP as federated identity provider, 102

G

Google, SSO setup with, 93-98
and OpenIDConnect, 93-95
Horizon configuration, 95
login page screenshots, 96-98
groups (see user groups)

H

Horizon
and PKI token troubleshooting, 42
and projects, 32
and SSO, 86, 95
and users, 32
basic Keystone operations using, 31
for multi-domain LDAP user authentication, 70-71
identity operations with, 32
integration for multi-region Keystone to Keystone federation support, 102
hybrid cloud, federated identity support for, x

I

IBM Bluemix, 103
IBM WebSphere Liberty
configuring Keystone to use OpenIDConnect, 90-91
downloading, installing, and configuring, 87-89
federating identities for, 87
testing, 92
Identity Providers (IdPs), 8
as Keystone resource, 6, 79
defined, 77
identity providers (IdPs)
use cases for identity backend, 9
identity(-ies), 5-9
and identity providers, 8
and LDAP, 6, 64
defined, xvi
multiple backends for, 7
operations with Horizon, 32
restrictions when using multi-domain LDAP, 61

SQL, 6
use cases for backends, 9
images, for multi-domain LDAP user, 69

K

Keystone
basic operations using Horizon, 31
basic operations using OpenStackClient, 18-31
concepts, 1-5
fundamental topics, 1-15
in development environment, 17-34
primary benefits of, xvii
tips, common pitfalls, and troubleshooting, 33
Keystone API, 58-60
Keystone to Keystone (K2K) federation, 102

L

LDAP (Lightweight Directory Access Protocol), ix, 45-75
approach to integration with Keystone, 45
as federated identity provider, 102
choosing the right mix of SQL and LDAP domains, 61
identities stored in, 6
Keystone configuration for integration with, 46-55
Keystone configuration options in classic LDAP support, 50-55
missing LDAP python libraries, 73
projects, roles, and assignments from, 72
tips, pitfalls, and troubleshooting, 73-75
tools for determining attributes, 73
use cases for identity backend, 9
with multiple domains (see multi-domain LDAP)
libraries, federation and, 99
Lightweight Directory Access Protocol (see LDAP)
login page, federated identity and, 86

M

man-in-the-middle (MITM) attacks, 51
mapping engine, 82
mapping rules, 83-85
mappings, 79

Microsoft Active Directory (see Active Directory (AD))
multi-domain LDAP
choosing the right mix of SQL and LDAP, 61
configuration file approach to setup, 57
corporate directory support requirements, 55-57
directory-mapping table with, 75
Keystone API-based approach, 58-60
LDAP for all domains, 64
LDAP for all domains except SQL service domain, 63
LDAP setup for, 64-67
restrictions when using identity, 61
running admin commands, 67
running LDAP user commands, 68
tips for using, 74
user authentication with Horizon, 70-71
with Keystone, 64-71
with SQL as default domain, 62
multi-factor authentication, 101

O

OpenIDConnect
claims, 80
configuring Keystone to use, 90-91
defined, 78
for SSO setup with Google, 93-95

OpenStack Barbican project, 103

OpenStackClient
assigning role to user for project, 29
authenticating as new user for project, 30
basic Keystone operations using, 18-31
creating domain, 27
creating project within domain, 27
creating user within domain, 28
debug option, 34
issuing tokens, 19
listing domains, 26
listing groups, 24
listing projects, 22
listing roles, 25
listing users, 21

P

password, xvi, 10
payload, 4, 10
PKI tokens, 37-39

origins of, 35
Swift or Horizon not working with, 42
PKI tokens, 37, 39, 42
policy
 and authorization, 34
 centralized, 103
policy engines, xvii, 103
policy files, 103
policy.json file, 12
project
 assigning role to group, 68
 assigning role to user, 29
 creating within domain, 27
 defined, 1
 Horizon and, 32
 listing, 22
 targets and, 3
 tenant vs., 14
Protocol (Keystone resource), 79
python libraries, LDAP, 73

R

referral chasing, 53
referrals, 53
region, domain vs., 14
Resource, LDAP problems with, 46, 72
Role-Based Access Control (RBAC), 12
roles
 and authorization, 34
 and tokens, 14
 assigning to a group, 68
 assigning to user, 29
 defined, 3
 listing, 25
rules, 12, 79

S

SAML
 assertions, 81
 authentication from command line, 99
 defined, 77
scoped tokens, 14
service accounts, 7
service catalogs, 5
 (see also catalogs)
Service Provider (SP), 77
service users, 102
services, summary of, 13
single sign-on (SSO)

and federated identity, 86
with Google as identity provider, 93-98
social media logins, federated identity and, 99
SQL
 default domain for multi-domain LDAP, 7, 62
 domain-specific configuration file for, 58
 for service domain only, 63
 identities stored in, 6
 use cases for identity backend, 9
SSO (see single sign-on [SSO])
Swift, 42

T

targets, 3, 12
tenant, 1, 14
 (see also project)
token formats, 35-43
 Fernet, 40
 history, 35-37
 PKI, 37-39, 42
 tips, pitfalls, and troubleshooting, 42
 UUID, 35, 37, 42
tokens
 and authentication, xvii, 11
 defined, 4
 for multi-domain LDAP, 68
 formats (see token formats)
 roles and, 14
 scoped vs. unscoped, 14
 with cURL, 19
 with OpenStackClient, 19
Transport Layer Security (TLS), 50
trees, 52

U

Ubuntu virtual machine, 17
universally unique identifier (UUID), 2
 (see also UUID tokens)
unscoped tokens, 14
user groups
 assigning role to, 68
 defined, 2
 finding in multi-domain LDAP, 67
 listing all members in multi-domain LDAP, 68
 listing with cURL, 24
 listing with OpenStackClient, 24
 mapping engine and, 82

- user(s)
 - and identity, [xvi](#)
 - assigning role to, [29](#)
 - authenticating as new, [30](#)
 - creating within domain, [28](#)
 - defined, [2](#)
 - domains and, [14](#), [74](#)
 - federated identity and, [80-85](#)
 - finding in multi-domain LDAP, [67](#)
 - Horizon and, [32](#)
 - listing with cURL, [21](#)
 - listing with LDAP, [61](#), [74](#)
 - listing with OpenStackClient, [21](#)
 - mapping engine and, [82](#)
 - translating attributes to Keystone concepts, [80-85](#)
- UUID tokens, [37](#)
 - early usage of, [35](#)
 - performance degradation for authentication operations, [42](#)

V

VM (virtual machine), [17](#), [69](#)

W

WebSphere Liberty (see IBM WebSphere Liberty)

X

X.509 certificates, [102](#)

About the Authors

Steve Martinelli is an OpenStack Active Technical Contributor and a Keystone Core Contributor. He primarily focuses on enabling Keystone, which is OpenStack's Identity Manager, to better integrate into enterprise environments. Steve was responsible for adding Federated Identity and OAuth support to Keystone and was one of the leading contributors to Keystone to Keystone federation support for interoperable hybrid cloud enablement. In his spare time he also contributes to OpenStackClient, pyCADF, and oslo.policy and is a core contributor in each of these projects. Steve received his B.ASc. in Computer Engineering from York University.

Henry Nash works in IBM's Cloud division as an OpenStack Architect and a core contributor to OpenStack Keystone, driving enterprise capabilities into OpenStack as well as IBM's products that use OpenStack. He has a long history of developing enterprise software, graphics, and communication systems as well as nanotechnology, having founded numerous successful companies in Europe and the United States, finally coming to IBM via acquisition in 2009. He holds a first class honors degree in Electrical Engineering from the University of Southampton, UK.

Dr. Brad Topol is an IBM Distinguished Engineer in the IBM Cloud Architecture and Technology organization. In his current role, Brad leads a development team focused on contributing to and improving OpenStack and he has cross-IBM responsibility for coordinating its contributions to OpenStack. Brad is an OpenStack core contributor to Keystone-Specs, pyCADF, and Heat-Translator and has personally contributed to multiple OpenStack projects including Keystone, Pycadf, Heat-Translator, and DevStack. He received a Ph.D. in Computer Science from the Georgia Institute of Technology in 1998.