

OpenStack インストールガイド

Ubuntu 12.04 (LTS) 版

icehouse (March 29, 2014)



OpenStack インストールガイド Ubuntu 12.04 (LTS) 版

[FAMILY Given]

icehouse (2014-03-29)

製作著作 © 2012, 2013 OpenStack Foundation All rights reserved.

概要

The OpenStack® system consists of several key projects that you install separately but that work together depending on your cloud needs. These projects include Compute, Identity Service, Networking, Image Service, Block Storage, Object Storage, Telemetry, and Orchestration. You can install any of these projects separately and configure them stand-alone or as connected entities. This guide walks through an installation by using packages available through Ubuntu 12.04 (LTS). Explanations of configuration options and sample configuration files are included.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

<http://www.apache.org/licenses/LICENSE-2.0>

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

目次

はじめに	8
表記規則	8
ドキュメント変更履歴	8
1. アーキテクチャー	1
概要	1
概念アーキテクチャー	2
サンプルアーキテクチャー	3
2. オペレーティングシステムの基本設定	7
始める前に	7
ネットワーク	7
Network Time Protocol (NTP)	9
パスワード	9
MySQL データベース	10
OpenStack パッケージ	11
メッセージングサーバー	12
3. Identity Service の設定	13
Identity Service の概念	13
Identity Service のインストール	15
ユーザー、プロジェクト、ロールの定義	16
サービスと API エンドポイントの定義	18
Identity Service のインストールの検証	19
4. OpenStack クライアントのインストールと設定	21
概要	21
OpenStack コマンドラインクライアントのインストール	22
OpenStack RC ファイル	24
5. Image Service の設定	26
Image Service の概要	26
Image Service のインストール	27
Image Service のインストールの検証	29
6. Compute Service の設定	32
Compute Service	32
Compute コントローラーサービスのインストール	35
コンピュータノードの設定	37
インスタンスの起動	39
7. Add a networking service	45
Networking (neutron)	45
Legacy networking	62
8. Dashboard の追加	64
システム要件	64
Dashboard のインストール	65
Dashboard 用セッションストレージのセットアップ	66
9. Add the Block Storage service	71
Block Storage	71
Configure a Block Storage service controller	71
Configure a Block Storage service node	73
10. Object Storage の追加	76
Object Storage Service	76
システム要件	77

Object Storage 用ネットワークの計画	78
Object Storage インストールアーキテクチャー例	78
Object Storage のインストール	79
ストレージノードのインストールと設定	81
プロキシノードのインストールと設定	83
ストレージノードでのサービスの起動	85
Object Storage のインストール後作業	86
11. Orchestration Service の追加	88
Orchestration Service 概要	88
Orchestration Service のインストール	88
Orchestration Service のインストールの検証	90
12. Telemetry モジュールの追加	94
Telemetry	94
Telemetry モジュールのインストール	95
Telemetry 用 Compute エージェントのインストール	97
Configure the Image Service for Telemetry	99
Add the Block Storage service agent for Telemetry	99
Configure the Object Storage service for Telemetry	99
Telemetry のインストールの検証	100
A. 予約済みユーザー ID	102
B. コミュニティのサポート	103
ドキュメント	103
ask.openstack.org	104
OpenStack メーリングリスト	104
OpenStack wiki	104
Launchpad バグエリア	105
OpenStack IRC チャンネル	106
ドキュメントへのフィードバック	106
OpenStackディストリビューション	106
用語集	107

図の一覧

1.1. 概念アーキテクチャー	3
1.2. レガシーなネットワークを持つ 2 ノードアーキテクチャー	5
1.3. OpenStack Networking (Neutron) を持つ 3 ノードアーキテクチャー	6
2.1. 基本アーキテクチャー	8
7.1. Initial networks	59

表の一覧

1.1. OpenStack のサービス	1
2.1. Passwords	9
4.1. OpenStack のサービスとクライアント	21
4.2. 前提ソフトウェア	22
10.1. ハードウェア推奨事項	77
A.1. 予約済みユーザー ID	102

例の一覧

2.1. /etc/network/interfaces	8
------------------------------------	---

はじめに

表記規則

OpenStack のドキュメントは、いくつかの植字の表記方法を採用しています。

Admonitions

Admonitions take three forms:



注記

This is a note. The information in a note is usually in the form of a handy tip or reminder.



重要

This is important. The information in an important admonition is something you must be aware of before moving on.



警告

This is a warning. The information in warnings is critical. Warnings provide additional information about risk of data loss or security issues.

Command prompts

Commands prefixed with the # prompt are to be executed by the root user. These examples can also be executed using the sudo command, if available.

Commands prefixed with the \$ prompt can be executed by any user, including root.

ドキュメント変更履歴

このバージョンのガイドはすべての旧バージョンを置き換え、廃止します。以下の表はもっとも最近の変更点を記載しています。

Revision Date	Summary of Changes
October 25, 2013	• Debian の初期サポートの追加。
October 17, 2013	• Havana リリース。
October 16, 2013	• SUSE Linux Enterprise のサポートの追加。
October 8, 2013	• Havana 向け再構成の完了。
September 9, 2013	• openSUSE 版の作成。
August 1, 2013	• Object Storage 検証手順の修正。バグ 1207347 の修正。
July 25, 2013	• cinder ユーザーの作成と service プロジェクトへの追加。バグ 1205057 の修正。
May 8, 2013	• 一貫性のために文書名の更新。

Revision Date	Summary of Changes
May 2, 2013	• 表紙の更新と付録の小さなミスの修正。

第1章 アーキテクチャー

目次

概要	1
概念アーキテクチャー	2
サンプルアーキテクチャー	3



警告

Icehouse 向けにこのドキュメントを更新中です。この作業中、構造や内容の問題が見つかるかもしれません。

概要

OpenStack プロジェクトは、あらゆる種類のクラウド環境をサポートする、オープンソースのクラウドコンピューティングプラットフォームです。シンプルな実装、大規模なスケラビリティ、豊富な機能を目指しています。世界中のクラウドコンピューティング技術者がプロジェクトに貢献しています。

OpenStack はさまざまな相補サービスを通して Infrastructure-as-a-Service (IaaS) ソリューションを提供します。各サービスはこの統合を促す Application Programming Interface (API) を提供します。以下の表は OpenStack サービスの一覧です。

表1.1 OpenStack のサービス

サービス	プロジェクト名	説明
Dashboard	Horizon	インスタンスの起動、IP アドレスの割り当て、アクセス制御の設定など、基礎となる OpenStack サービスを操作するために、ウェブベースのセルフサービスポータルを提供します。
Compute	Nova	OpenStack 環境でコンピュータインスタンスのライフサイクルを管理します。要求に応じて仮想マシンの作成、スケジューリング、廃棄などに責任を持ちます。
Networking	Neutron	OpenStack Compute のような他の OpenStack サービスに対してサービスとしてのネットワーク接続性を可能にします。ユーザーがネットワークやそれらへの接続を定義するための API を提供します。数多くの人気のあるネットワークベンダーや技術をサポートする、プラグイン可能なアーキテクチャーを持ちます。
ストレージ		
Object Storage	Swift	RESTful、HTTP ベースの API 経由で任意の非構造データオブジェクトを保存および取得します。そのデータ複製およびスケールアウトアーキテクチャーで高い耐障害性を持ちます。その実装はマウント可能なディレクトリを持つファイルサーバーのようではありません。
Block Storage	Cinder	実行中のインスタンスに永続的なブロックストレージを提供します。そのプラグイン可能なドライバーアーキテクチャーにより、ブロックストレージデバイスの作成と管理が容易になります。
共有サービス		

サービス	プロジェクト名	説明
Identity Service	Keystone	他の OpenStack サービスに対して認証および認可サービスを提供します。すべての OpenStack サービスに対してエンドポイントのカタログを提供します。
Image Service	Glance	仮想マシンディスクイメージを保存および取得します。OpenStack Compute がインスタンスの配備中に使用します。
Telemetry	Ceilometer	課金、ベンチマーク、スケーラビリティ、統計などの目的のために、OpenStack クラウドを監視および測定します。
高レベルサービス		
Orchestration	Heat	OpenStack ネイティブの REST API および CloudFormation 互換のクエリー API 経由で、ネイティブの HOT テンプレート形式または AWS CloudFormation テンプレート形式を使用することにより、複数の混合クラウドアプリケーションを統合します。

This guide describes how to deploy these services in a functional test environment and, by example, teaches you how to build a production environment.

概念アーキテクチャー

Launching a virtual machine or instance involves many interactions among several services. The following diagram provides the conceptual architecture of a typical OpenStack environment.

The diagram illustrates the OpenStack architecture, showing the interactions between various services. The components and their relationships are as follows:

- Heat** (top center, brown box) orchestrates the cloud.
- Horizon** (top left, white box) provides the UI for the cloud.
- Neutron** (middle left, blue box) provides network connectivity for VMs.
- Nova** (middle center, red box) provisions VMs.
- Glance** (middle right, teal box) provides images to VMs.
- Swift** (far right, green box) stores images in.
- Cinder** (bottom left, green box) provides volumes for VMs.
- Ceilometer** (bottom center, purple box) monitors the system.
- Keystone** (bottom right, orange box) provides authentication for the system.

The interactions are as follows:

- Heat** orchestrates the cloud.
- Horizon** provides the UI for the cloud.
- Horizon** provides network connectivity for **VM**.
- Horizon** provides images to **VM**.
- Horizon** provides authentication for **Keystone**.
- Horizon** provides volumes for **Cinder**.
- Horizon** provides authentication for **Keystone**.
- Neutron** provides network connectivity for **VM**.
- Nova** provisions **VM**.
- Glance** provides images to **VM**.
- Glance** stores images in **Swift**.
- Cinder** provides volumes for **VM**.
- Ceilometer** monitors the system.
- Keystone** provides authentication for the system.

- レガシーなネットワークを持つ 2 ノードアーキテクチャー。図1.2「レガシーなネットワークを持つ 2 ノードアーキテクチャー」 [5]を参照してください。
- The basic controller node runs the Identity service, Image Service, management portion of Compute, and the dashboard necessary to launch a simple instance. It also includes supporting services such as MySQL, AMQP, and NTP.

Optionally, the controller node also runs portions of Block Storage, Object Storage, Database Service, Orchestration, and Telemetry. These components provide additional features for your environment.

- The basic compute node runs the hypervisor portion of Compute, which operates tenant virtual machines or instances. By default, Compute uses KVM as the hypervisor. Compute also provisions and operates tenant networks and implements security groups. You can run more than one compute node.

Optionally, the compute node also runs the Telemetry agent. This component provides additional features for your environment.



注記

When you implement this architecture, skip [「Networking \(neutron\)」 \[45\]](#) in [7章Add a networking service \[45\]](#). To use optional services, you might need to install additional nodes, as described in subsequent chapters.

図1.2 レガシーなネットワークを持つ 2 ノードアーキテクチャー



- OpenStack Networking (Neutron) を持つ 3 ノードアーキテクチャー。図 [1.3「OpenStack Networking \(Neutron\) を持つ 3 ノードアーキテクチャー」](#) [6]を参照してください。
- 基本的なコントローラーノードは、Identity Service、Image Service、および Compute、Networking、Networking プラグイン、ダッシュボードの管理部分を実行します。MySQL、AMQP、NTP のようなサポートサービスも含まれます。

Optionally, the controller node also runs portions of Block Storage, Object Storage, Database Service, Orchestration, and Telemetry. These components provide additional features for your environment.

- The network node runs the Networking plug-in, layer 2 agent, and several layer 3 agents that provision and operate tenant networks. Layer 2 services include provisioning of virtual networks and tunnels. Layer 3 services include routing, NAT, and DHCP. This node also handles external (internet) connectivity for tenant virtual machines or instances.
- The compute node runs the hypervisor portion of Compute, which operates tenant virtual machines or instances. By default Compute uses KVM as the hypervisor. The compute node also runs the Networking plug-in and layer 2 agent which operate tenant networks and implement security groups. You can run more than one compute node.

Optionally, the compute node also runs the Telemetry agent. This component provides additional features for your environment.



注記

When you implement this architecture, skip 「[Legacy networking](#)」 [62] in [7章 Add a networking service](#) [45]. To use optional services, you might need to install additional nodes, as described in subsequent chapters.

図1.3 OpenStack Networking (Neutron) を持つ 3 ノードアーキテクチャー



第2章 オペレーティングシステムの基本 設定

目次

始める前に	7
ネットワーク	7
Network Time Protocol (NTP)	9
パスワード	9
MySQL データベース	10
OpenStack パッケージ	11
メッセージングサーバー	12



警告

Icehouse 向けにこのドキュメントを更新中です。この作業中、構造や内容の問題が見つかるかもしれません。

このガイドは、ほとんどのサービスをホストするためにコントローラーノードを作成し、仮想マシンインスタンスを実行するためにコンピューターノードを作成する方法を示します。その後の章はさらにサービスを実行するために追加のノードを作成します。OpenStack は各サービスの実行方法や実行場所について柔軟性があります。そのため、他の設定をすることができます。しかしながら、各ノードでいくつかのオペレーティングシステムの設定を実行する必要があります。

この章はコントローラーノードとあらゆる追加ノードの設定例を詳細に説明します。他の方法でオペレーティングシステムを設定できますが、このガイドはお使いの環境がここで記載されたものと互換性があることを仮定しています。

All example commands assume you have administrative privileges. Either run the commands as the root user or prefix them with the `sudo` command.

始める前に

コンピューターノードは 64 ビットオペレーティングシステムをインストールすることを強く推奨します。32 ビットオペレーティングシステムを使用している場合、64 ビットのイメージを使用して仮想マシンを起動すると、エラーが発生するでしょう。

システム要件の詳細は [OpenStack 運用ガイド](#) を参照してください。

ネットワーク

OpenStack の本番環境の場合、多くのノードはこれらのネットワークインターフェースカードを持つ必要があります。

- 外部ネットワーク通信用のネットワークインターフェースカード 1 枚

- 他の OpenStack ノードと通信するための別のカード

簡単なテストケースの場合、ネットワークインターフェースカード 1 枚のマシンを使用できます。

以下の例は静的 IP アドレスを持つ 2 つのネットワークを設定し、各マシンのホスト名の一覧を手動で管理します。大規模なネットワークを管理する場合、これを管理するための適切なシステムがすでにあるかもしれません。そうならば、このセクションを読み飛ばしますが、このガイドのこれ以降は各ノードが内部ネットワークで controller と compute1 というホスト名を使用して他のノードにアクセスできることを仮定しています。

eth0 と eth1 を設定します。このガイドの例は、内部ネットワーク用に 192.168.0.x の IP アドレスを、外部ネットワーク用に 10.0.0.x の IP アドレスを使用します。ネットワークデバイスが正しいネットワークに接続できることを確認してください。

このガイドでは、コントローラーノードは 192.168.0.10 と 10.0.0.10 を使用します。コンピューターノードを作成するときは、代わりに 192.168.0.11 と 10.0.0.11 を使用します。後続の章で追加するノードもこのパターンに従います。

図2.1 基本アーキテクチャー



例2.1 /etc/network/interfaces

```
# Internal Network
auto eth0
iface eth0 inet static
    address 192.168.0.10
    netmask 255.255.255.0

# External Network
auto eth1
iface eth1 inet static
    address 10.0.0.10
    netmask 255.255.255.0
```

ネットワークを設定した後、変更を反映するためにデーモンを再起動します。

```
# service networking restart
```

各マシンのホスト名を設定します。コントローラーノードに controller という名前を付け、最初のコントローラーノードに compute1 という名前を付けます。このガイドの例はこれらのホスト名を使用します。

Use the hostname command to set the host name:

```
# hostname controller
```

システムの再起動時に利用可能になるよう、このホスト名を設定するために、ホスト名のみの単一行を含む `/etc/hostname` ファイルにこれを指定する必要があります。

最後に、各ノードがホスト名を使用して他のノードに到達できることを確認します。各ノードで `/etc/hosts` ファイルを手動で編集する必要があります。大規模導入の場合、DNS または Puppet のような構成管理システムを使用します。

```
127.0.0.1    localhost
192.168.0.10 controller
192.168.0.11 compute1
```

Network Time Protocol (NTP)

複数のマシンにわたりサービスを同期するために、NTP をインストールする必要があります。このガイドの例は、コントローラーノードを参照サーバーとして設定し、他のすべてのノードはコントローラーノードから時刻を設定するよう設定します。

OpenStack サービスを実行している各システムに `ntp` パッケージをインストールします。

```
# apt-get install ntp
```

追加ノードでは、他のノードが LAN の外ではなく、コントローラーノードから時刻を同期するよう設定することをお勧めします。そうするために、上のとおり NTP デーモンをインストールし、`/etc/ntp.conf` を編集し、内部時刻ソースとしてコントローラーノードを使用するために `server` ディレクティブを変更します。

パスワード

The various OpenStack services and the required software like the database and the messaging server have to be password protected. You use these passwords when configuring a service and then again to access the service. You have to choose a password while configuring the service and later remember to use the same password when accessing it. Optionally, you can generate random passwords with the `pwgen` program. Or, to create passwords one at a time, use the output of this command repeatedly:

```
$ openssl rand -hex 10
```

このガイドは以下のとおり表記します。SERVICE_PASS が SERVICE にアクセスするためのパスワード、SERVICE_DBPASS がデータベースにアクセスするために SERVICE サービスにより使用されるデータベースのパスワードです。

The complete list of passwords you need to define in this guide are:

表2.1 Passwords

Password name	Description
Database password (no variable used)	Root password for the database
RABBIT_PASS	Password of user guest of RabbitMQ
KEYSTONE_DBPASS	Database password of Identity service
ADMIN_PASS	Password of user admin

Password name	Description
GLANCE_DBPASS	Database password for Image Service
GLANCE_PASS	Password of Image Service user glance
NOVA_DBPASS	Database password for Compute service
NOVA_PASS	Password of Compute service user nova
DASH_DBPASS	Database password for the dashboard
CINDER_DBPASS	Database password for the Block Storage service
CINDER_PASS	Password of Block Storage service user cinder
NEUTRON_DBPASS	Database password for the Networking service
NEUTRON_PASS	Password of Networking service user neutron
HEAT_DBPASS	Database password for the Orchestration service
HEAT_PASS	Password of Orchestration service user heat
CEILOMETER_DBPASS	Database password for the Telemetry service
CEILOMETER_PASS	Password of Telemetry service user ceilometer

MySQL データベース

多くの OpenStack サービスは情報を保存するためにデータベースを必要とします。これらの例はコントローラーノードで MySQL データベースを使用します。コントローラーノードに MySQL データベースをインストールする必要があります。MySQL にアクセスする他のノードすべてに MySQL クライアントソフトウェアをインストールする必要があります。

コントローラーのセットアップ

コントローラーノードに MySQL クライアントとサーバーパッケージ、Python ライブラリをインストールします。

```
# apt-get install python-mysqldb mysql-server
```



注記

サーバーパッケージをインストールするとき、データベースの root パスワードを入力するよう求められます。強いパスワードを選択し、それを覚えておきます。

OpenStack を扱うために、いくつかの MySQL の設定変更が必要になります。

- `/etc/mysql/my.cnf` ファイルを編集します。
 - a. 管理ネットワーク経由で他のノードからアクセスできるようにするために、`[mysqld]` セクションの下で、`bind-address` キーにコントローラーノードの管理 IP アドレスを設定します。

```
[mysqld]
...
bind-address = 192.168.0.10
```

- b. デフォルトで UTF-8 文字を設定するために、`[mysqld]` セクションの下で、以下のキーを設定します。

```
[mysqld]
...
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8
```

変更を適用するために MySQL サービスを再起動します。

```
# service mysql restart
```

You must delete the anonymous users that are created when the database is first started. Otherwise, database connection problems occur when you follow the instructions in this guide. To do this, use the `mysql_secure_installation` command. Note that if `mysql_secure_installation` fails you might need to use `mysql_install_db` first:

```
# mysql_install_db
# mysql_secure_installation
```

This command presents a number of options for you to secure your database installation. Respond yes to all prompts unless you have a good reason to do otherwise.

ノードのセットアップ

コントローラーノード以外のすべてのノードで、MySQL データベースをホストしていないすべてのシステムで MySQL クライアントと MySQL Python ライブラリをインストールします。

```
# apt-get install python-mysqldb
```

OpenStack パッケージ

ディストリビューションはその一部として OpenStack パッケージをリリースしているかもしれませんが、または、OpenStack とディストリビューションのリリース間隔がお互いに独立しているため、他の方法によりリリースしているかもしれません。

このセクションは、最新の OpenStack パッケージをインストールするために、マシンを設定した後完了する必要がある設定について説明します。

Icehouse 用 Ubuntu Cloud Archive の使用法

[Ubuntu Cloud Archive](#) は、Ubuntu の安定サポートバージョンで OpenStack の最新版をインストールできるようにするための特別なリポジトリです。

1. Install the Ubuntu Cloud Archive for Icehouse:

```
# apt-get install python-software-properties
# add-apt-repository cloud-archive:icehouse
```

2. Update the package database, upgrade your system, and reboot for all changes to take effect:

```
# apt-get update && apt-get dist-upgrade
```

```
# reboot
```

メッセージングサーバー

コントローラーノードにメッセージキューサーバーをインストールします。一般的に RabbitMQ ですが、Qpid や ZeroMQ (0MQ) も利用できます。

```
# apt-get install rabbitmq-server
```



重要なセキュリティ考慮事項

rabbitmq-server パッケージは、RabbitMQ サービスが自動的に起動するよう設定し、デフォルトの guest パスワードで guest ユーザーを作成します。このガイドの RabbitMQ の例は guest アカウントを使用しますが、特に IPv6 が利用可能な場合、そのデフォルトのパスワードを変更することを強く推奨します。RabbitMQ サーバーはデフォルトで、ログインとパスワードとしてゲストを使用することにより、すべてのものが接続できます。IPv6 を用いると、外部から到達可能です。

RabbitMQ のデフォルトのゲストパスワードを変更する方法:

```
# rabbitmqctl change_password guest RABBIT_PASS
```

おめでとうございます。これで OpenStack サービスをインストールする準備ができました。

第3章 Identity Service の設定

目次

Identity Service の概念	13
Identity Service のインストール	15
ユーザー、プロジェクト、ロールの定義	16
サービスと API エンドポイントの定義	18
Identity Service のインストールの検証	19

Identity Service の概念

Identity Service は以下の機能を実行します。

- ユーザー管理。ユーザーとその権限を追跡します。
- サービスカタログ。利用可能なサービスのカタログとその API エンドポイントを提供します。

Identity Service を理解するために、以下の概念を理解する必要があります。

ユーザー	人、システム、または OpenStack クラウドサービスを使用するサービスの電子的な表現。Identity Service は遅れられてきたリクエストがどのユーザーにより行われているかを検証します。ユーザーはログインでき、リソースにアクセスするためにトークンを割り当てられるかもしれませんが。ユーザーは特定のテナントに直接割り当てられ、そのテナントに含まれているかのように振る舞います。
クレデンシャル	ユーザーが誰であることを証明するために、ユーザーのみにより知られているデータ。Identity Service では、次のようなものがあります。ユーザー名とパスワード、ユーザー名と API キー、Identity Service により発行された認証トークン。
認証	ユーザーの同一性を確認する動作。Identity Service は、ユーザーに提供された一組のクレデンシャルを検証することにより、送られてきたリクエストを確認します。 これらのクレデンシャルは最初にユーザー名とパスワード、またはユーザー名と API トークンです。これらのクレデンシャルの応答で、Identity Service がユーザーに認証トークンを発行します。ユーザーはこれ以降のリクエストでこのトークンを提供します。
トークン	リソースにアクセスするために使用される任意のビット数のテキスト。各トークンはアクセス可能なリソースを記述した範囲を持ちます。トークンは適宜失効しているかもしれません。また、有限の期間だけ有効です。

Identity Service はこのリリースでトークンによる認証をサポートしますが、その意図は将来的にさらなるプロトコルをサポートすることです。意図は真っ先に統合サービスになるためですが、十分に成熟した認証ストアや管理ソリューションにある熱意はありません。

テナント

リソース、主体オブジェクト、またはその組み合わせをグループ化、または分離するために使用されるコンテナ。サービス操作者に依存して、テナントが顧客、アカウント、組織、プロジェクトに対応付けられるかもしれません。

サービス

Compute (Nova)、Object Storage (Swift)、Image Service (Glance) のような OpenStack サービス。ユーザーがリソースにアクセスでき、操作を実行できる 1 つ以上のエンドポイントを提供します。

エンドポイント

サービスにアクセスするところからネットワークアクセス可能なアドレス。通常は URL により記載されます。テンプレート用の拡張を使用している場合、エンドポイントのテンプレートを作成できます。これはリージョンを越えて利用できる、すべての消費できるサービスのテンプレートです。

役割

ユーザーが特定の操作の組を実行できると仮定する人格。ロールは一組の権利と権限を含みます。そのロールを仮定しているユーザーは、それらの権利と権限を継承します。

Identity Service では、ユーザーに発行されたトークンはユーザーが持つロールの一覧を含みます。そのユーザーにより呼び出されたサービスは、ユーザーが持つロール一覧を解釈する方法と、各ロールがアクセス権を持つ操作やリソースを判断します。

以下の図は Identity Service のプロセスフローを示します。



Identity Service のインストール

1. OpenStack Identity Service と python-keystoneclient (依存関係) をコントローラーノードにインストールします。

```
# apt-get install keystone
```

2. Identity Service は情報を保存するためにデータベースを使用します。設定ファイルでデータベースの場所を指定します。このガイドでは、コントローラーノードにユーザー名 keystone で MySQL データベースを使用します。KEYSTONE_DBPASS をデータベースのユーザーの適切なパスワードで置き換えます。

/etc/keystone/keystone.conf を編集し、[database] セクションを変更します。

```
...
[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://keystone:KEYSTONE_DBPASS@controller/keystone
...
```

3. Ubuntu パッケージはデフォルトで SQLite データベースを作成します。誤って使用されないように、/var/lib/keystone/ ディレクトリに作成された keystone.db ファイルを削除します。
4. root としてログインするために、前に設定したパスワードを使用します。keystone データベースユーザーを作成します。

```
$ mysql -u root -p
mysql> CREATE DATABASE keystone;
mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' ¥
IDENTIFIED BY 'KEYSTONE_DBPASS';
mysql> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' ¥
IDENTIFIED BY 'KEYSTONE_DBPASS';
```

5. Image Service 用のデータベーステーブルを作成します。

```
# keystone-manage db_sync
```

6. Define an authorization token to use as a shared secret between the Identity Service and other OpenStack services. Use openssl to generate a random token and store it in the configuration file:

```
# openssl rand -hex 10
```

/etc/keystone/keystone.conf を編集し、[DEFAULT] セクションを変更します。ADMIN_TOKEN をコマンドの結果で置き換えます。

```
[DEFAULT]
# A "shared secret" between keystone and other openstack services
admin_token = ADMIN_TOKEN
...
```

7. Identity Service を再起動します。

```
# service keystone restart
```


8. Identity Service は標準で、期限切れトークンをデータベースに無期限に保存します。本番環境で監査のために有用であるかもしれませんが、期限切れトークンが蓄積すると、データベースの容量がかなり大きくなり、サービスの性能を劣化させるかもしれません。とくにリソースが限られたテスト環境で顕著かもしれません。期限切れトークンを 1 時間おきに削除するために、cron を使用して定期タスクを設定することを推奨します。

- Run the following command to purge expired tokens every hour and log the output to /var/log/keystone/keystone-tokenflush.log:

```
# crontab -l | grep -q token_flush || \
echo '@hourly /usr/bin/keystone-manage token_flush >/var/log/keystone/keystone-
tokenflush.log 2>&1' >> /var/spool/cron/root
```

ユーザー、プロジェクト、ロールの定義

Identity Service をインストールした後、認証するユーザー、プロジェクト、ロールをセットアップします。これらは次のセクションに記述されるサービスとエンドポイントへのアクセスを許可するために使用されます。

Typically, you would indicate a user and password to authenticate with the Identity Service. At this point, however, we have not created any users, so we have to use the authorization token created in an earlier step, see [「Identity Service のインストール」 \[15\]](#) for further details. You can pass this with the `--os-token` option to the keystone command or set the `OS_SERVICE_TOKEN` environment variable. We'll set `OS_SERVICE_TOKEN`, as well as `OS_SERVICE_ENDPOINT` to specify where the Identity Service is running. Replace `ADMIN_TOKEN` with your authorization token.

```
$ export OS_SERVICE_TOKEN=ADMIN_TOKEN
$ export OS_SERVICE_ENDPOINT=http://controller:35357/v2.0
```

管理ユーザーの作成

管理ユーザー、ロール、プロジェクトを作成するために、以下の手順を実行します。OpenStack クラウドの管理操作のために、このアカウントを使用します。

Identity Service は標準で特別な `_member_` ロールを作成します。OpenStack ダッシュボードは自動的にこのロールを持つユーザーにアクセス権を与えます。admin ユーザーのアクセス権に、このロールに加えて admin ロールを与えます。



注記

作成するすべてのロールは、各 OpenStack サービスに含まれる `policy.json` ファイルで指定されたロールにマップすべきです。多くのサービス用の標準ポリシーファイルは、管理アクセスを admin ロールに許可します。

1. admin ユーザーを作成します。

```
$ keystone user-create --name=admin --pass=ADMIN_PASS --email=ADMIN_EMAIL
```

ADMIN_PASS を安全なパスワードに置き換え、ADMIN_EMAIL をこのアカウントに関連付ける電子メールアドレスに置き換えます。

2. admin ロールを作成します。

```
$ keystone role-create --name=admin
```

3. admin プロジェクトを作成します。

```
$ keystone tenant-create --name=admin --description="Admin Tenant"
```

4. ここで user-role-add オプションを使用して、admin ユーザー、admin ロール、admin プロジェクトをリンクする必要があります。

```
$ keystone user-role-add --user=admin --tenant=admin --role=admin
```

5. admin ユーザー、_member_ ロール、admin プロジェクトをリンクします。

```
$ keystone user-role-add --user=admin --role=_member_ --tenant=admin
```

一般ユーザーの作成

一般ユーザーとプロジェクトを作成し、それらと特別な _member_ ロールをリンクするために、以下の手順を実行します。OpenStack クラウドの日々の非管理操作のために、このアカウントを使用します。別のユーザー名とパスワードを持つユーザーを作成するために、この手順を繰り返すことができます。これらのユーザーを作成するときに、プロジェクトを作成する手順を省略します。

1. demo ユーザーを作成します。

```
$ keystone user-create --name=demo --pass=DEMO_PASS --email=DEMO_EMAIL
```

DEMO_PASS を安全なパスワードに置き換え、DEMO_EMAIL をこのアカウントに関連付ける電子メールアドレスに置き換えます。

2. demo プロジェクトを作成します。

```
$ keystone tenant-create --name=demo --description="Demo Tenant"
```



注記

別のユーザーを追加するときに、この手順を繰り返さないでください。

3. demo ユーザー、_member_ ロール、demo プロジェクトをリンクします。

```
$ keystone user-role-add --user=demo --role=_member_ --tenant=demo
```

service プロジェクトの作成

OpenStack のサービスは、他の OpenStack のサービスにアクセスするために、ユーザー名、プロジェクト、ロールを必要とします。基本的なインストールでは、OpenStack のサービスは一般的に同じ service という名前のプロジェクトを共有します。

各サービスをインストールし、設定するので、このプロジェクトの下に追加のユーザー名とロールを作成します。

- service プロジェクトを作成します。

```
$ keystone tenant-create --name=service --description="Service Tenant"
```

サービスと API エンドポイントの定義

Identity Service が、どの OpenStack サービスがインストールされているか、それらがネットワークのどこにあるかを追跡できるよう、OpenStack インストール環境の各サービスを登録する必要があります。サービスを登録するために、これらのコマンドを実行します。

- `keystone service-create`. Describes the service.
- `keystone endpoint-create`. Associates API endpoints with the service.

Identity Service 自身も登録する必要があります。前に設定した `OS_SERVICE_TOKEN` 環境変数を認証のために使用します。

1. Identity Service のサービスエントリーを作成します。

```
$ keystone service-create --name=keystone --type=identity --description="OpenStack Identity Service"
```

Property	Value
description	OpenStack Identity Service
id	15c11a23667e427e91bc31335b45f4bd
name	keystone
type	identity

サービス ID はランダムに生成され、ここに表示されているものとは異なります。

2. 返されたサービス ID を使用することにより、Identity Service の API エンドポイントを指定します。エンドポイントを指定するとき、パブリック API、内部 API、管理 API の URL を指定します。このガイドでは、`controller` というホスト名を使用します。Identity Service は管理 API 用に異なるポートを使用することに注意してください。

```
$ keystone endpoint-create --service-id=$(keystone service-list | awk '/ identity / {print $2}') --publicurl=http://controller:5000/v2.0 --internalurl=http://controller:5000/v2.0 --adminurl=http://controller:35357/v2.0
```

Property	Value
adminurl	http://controller:35357/v2.0
id	11f9c625a3b94a3f8e66bf4e5de2679f
internalurl	http://controller:5000/v2.0
publicurl	http://controller:5000/v2.0
region	regionOne
service_id	15c11a23667e427e91bc31335b45f4bd



注記

お使いの OpenStack 環境に追加した各サービス用の追加のエンドポイントを作成することが必要になります。各サービスのインストールと関連したこのガイドのセクションに、サービスへのエンドポイントの具体的な作成手順があります。

Identity Service のインストールの検証

1. Identity Service が正しくインストールされ、設定されていることを確認するためには、OS_SERVICE_TOKEN 環境変数と OS_SERVICE_ENDPOINT 環境変数にある値を削除します。

```
$ unset OS_SERVICE_TOKEN OS_SERVICE_ENDPOINT
```

管理ユーザーをブートストラップし、Identity Service に登録するために使用された、これらの変数はもはや必要ありません。

2. これで通常のユーザー名による認証を使用できます。

admin ユーザーと、そのユーザー用に選択したパスワードを使用して認証トークンを要求します。

```
$ keystone --os-username=admin --os-password=ADMIN_PASS ¥  
--os-auth-url=http://controller:35357/v2.0 token-get
```

応答で、ユーザー ID とペアになったトークンを受け取ります。これにより、Identity Service が期待したエンドポイントで実行されていて、ユーザーアカウントが期待したクレデンシャルで確立されていることを検証できます。

3. 認可が期待したとおり動作することを検証します。そうするために、プロジェクトで認可を要求します。

```
$ keystone --os-username=admin --os-password=ADMIN_PASS ¥  
--os-tenant-name=admin --os-auth-url=http://controller:35357/v2.0 ¥  
token-get
```

応答で、指定したプロジェクトの ID を含むトークンを受け取ります。これにより、ユーザーアカウントが指定したプロジェクトで明示的に定義したロールを持ち、プロジェクトが期待したとおり存在することを検証します。

4. コマンドラインの使用を簡単にするために、お使いの環境で --os-* 変数を設定することもできます。admin クレデンシャルと admin エンドポイントを用いて openrc.sh ファイルをセットアップします。

```
export OS_USERNAME=admin  
export OS_PASSWORD=ADMIN_PASS  
export OS_TENANT_NAME=admin  
export OS_AUTH_URL=http://controller:35357/v2.0
```

5. 環境変数を読み込むために、このファイルを source します。

```
$ source openrc.sh
```

6. openrc.sh ファイルが正しく設定されていることを検証します。同じコマンドを --os-* 引数なしで実行します。

```
$ keystone token-get
```

コマンドはトークンと指定されたプロジェクトの ID を返します。これにより、環境変数が正しく設定されていることを確認します。

7. admin アカウントが管理コマンドを実行する権限があることを検証します。

```
$ keystone user-list
```

id	enabled	email	name
a4c2d43f80a549a19864c89d759bb3fe	True	admin@example.com	admin

```
$ keystone user-role-list
```

tenant_id	id	name	user_id
9fe2ff9ee4384b1894a90878d3e92bab	e519b772cb43474582fa303da62559e5	_member_	a4c2d43f80a549a19864c89d759bb3fe
5d3b60b66f1f438b80eaae41a77b5951	e519b772cb43474582fa303da62559e5	admin	a4c2d43f80a549a19864c89d759bb3fe

Seeing that the id in the output from the keystone user-list command matches the user_id in the keystone user-role-list command, and that the admin role is listed for that user, for the related tenant, this verifies that your user account has the admin role, which matches the role used in the Identity Service policy.json file.



注記

コマンドラインや環境変数経由でクレデンシャルと Identity Service エンドポイントを定義する限り、すべてのマシンからすべての OpenStack クライアントコマンドを実行できます。詳細は [4章OpenStack クライアントのインストールと設定 \[21\]](#) を参照してください。

第4章 OpenStack クライアントのインストールと設定

目次

概要	21
OpenStack コマンドラインクライアントのインストール	22
OpenStack RC ファイル	24

概要

API コールを行う簡単なコマンドを実行するために、OpenStack コマンドラインクライアントを使用できます。コマンドラインから、または作業を自動化するためのスクリプトでこれらのコマンドを実行できます。OpenStack クレデンシャルを提供する限り、そのマシンでもこれらのコマンドを実行できます。

内部的に、各クライアントコマンドは API リクエストを組み込んだ cURL コマンドを実行します。OpenStack API は、メソッド、URI、メディアタイプ、応答コードを含む HTTP プロトコルを使用する RESTful API です。

これらのオープンソースの Python クライアントは、Linux または Mac OS X システムで実行します。これらは簡単に習得し、使用できます。OpenStack の各サービスは自身のコマンドラインクライアントを持ちます。いくつかのクライアントコマンドでは、コマンドのベースになる API リクエストを表示するために、debug パラメーターを指定できます。これは OpenStack API コールに慣れるために良い方法です。

以下の表は、各 OpenStack サービスのコマンドラインクライアント、そのパッケージ名、説明の一覧です。

表4.1 OpenStack のサービスとクライアント

サービス	クライアント	パッケージ	説明
Block Storage	cinder	python-cinderclient	ボリュームを作成、管理します。
Compute	nova	python-novaclient	イメージ、インスタンス、フレーバーを作成、管理します。
Identity	keystone	python-keystoneclient	ユーザー、プロジェクト、ロール、エンドポイント、クレデンシャルを作成、管理します。
Image Service	glance	python-glanceclient	イメージを作成、管理します。
Networking	neutron	python-neutronclient	ゲストサーバー用のネットワークを設定します。このクライアントは以前 quantum として知られていました。
Object Storage	swift	python-swiftclient	統計情報を収集し、項目を一覧表示し、メタデータを更新し、Object Storage サービスにより保存されたファイルをアップロード、ダウンロード、削除します。
Orchestration	heat	python-heatclient	テンプレートからスタックを起動し、イベントやリソースを含む実行中のスタックの詳細を表示し、スタックを更新、削除します。

サービス	クライアント	パッケージ	説明
Telemetry	ceilometer	python-ceilometerclient	OpenStack 全体の測定項目を作成、収集します。

An OpenStack common client is in development.

OpenStack コマンドラインクライアントのインストール

前提ソフトウェアと各 OpenStack クライアント用の Python パッケージをインストールします。



注記

各コマンドに対して、nova のように、インストールするクライアントの小文字の名前で PROJECT を置き換えます。各クライアントに対して繰り返します。

表4.2 前提ソフトウェア

前提	説明
Python 2.6 またはそれ以降	現在、クライアントは Python 3 をサポートしません。
setuptools パッケージ	Mac OS X に標準でインストールされます。 多くの Linux ディストリビューションはインストールしやすい setuptools パッケージを提供します。インストールパッケージを検索するために、パッケージマネージャーで setuptools を検索します。見つけられない場合、 http://pypi.python.org/pypi/setuptools から setuptools パッケージを直接ダウンロードします。 Microsoft Windows に setuptools をインストールする推奨の方法は setuptools ウェブサイト で提供されているドキュメントに従うことです。他の選択肢は hristoph Gohlke (http://www.lfd.uci.edu/~gohlke/pythonlibs/#setuptools) によりメンテナンスされている非公式のバイナリインストーラーを使用することです。
pip パッケージ	Linux、Mac OS X、Microsoft Windows システムにクライアントをインストールするために、pip を使用します。これは使いやすく、必ず Python Package Index から最新バージョンのクライアントを取得します。後からパッケージの更新や削除ができます。 お使いのシステムのパッケージマネージャーを利用して pip をインストールします。 Mac OS X. <pre># easy_install pip</pre> Microsoft Windows. Make sure that the C:\Python27\Scripts directory is defined in the PATH environment variable, and use the easy_install command from the setuptools package: <pre>C:\>easy_install pip</pre> Another option is to use the unofficial binary installer provided by Christoph Gohlke (http://www.lfd.uci.edu/~gohlke/pythonlibs/#pip).

前提	説明
	<p>Ubuntu 12.04. A packaged version enables you to use dpkg or aptitude to install the python-novaclient:</p> <pre># aptitude install python-novaclient</pre> <p>Ubuntu.</p> <pre># aptitude install python-pip</pre> <p>RHEL, CentOS, Fedora. A packaged version available in RDO enables you to use yum to install the clients:</p> <pre># yum install python-PROJECTclient</pre> <p>あるいは、クライアントのインストールを管理するために pip をインストールして使用します。</p> <pre># yum install python-pip</pre> <p>openSUSE 12.2 およびそれ以前. A packaged version available in the Open Build Service enables you to use rpm or zypper to install the python-novaclient:</p> <pre># zypper install python-PROJECT</pre> <p>Alternatively, install pip and use it to manage client installation:</p> <pre># zypper install python-pip</pre> <p>openSUSE 12.3 およびそれ以降. A packaged version enables you to use rpm or zypper to install the clients:</p> <pre># zypper install python-PROJECTclient</pre>

クライアントのインストール

OpenStack クライアントを Linux、Mac OS X、Microsoft Windows システムにインストールするために pip を使用します。これは簡単であり、[Python Package Index](#) から確実に最新バージョンのクライアントを取得します。また、pip によりパッケージの更新や削除ができます。クライアントをインストールした後、クライアントや API 経由で OpenStack のサービスをリクエストする前に、必要な環境変数を設定するために、openrc.sh ファイルを読み込む必要があります。

- それぞれ以下のとおりクライアントをインストールします。

- Mac OS X または Linux の場合:

```
# pip install python-PROJECTclient
```

- Microsoft Windows の場合:

```
C:¥>pip install python-PROJECTclient
```

ここで PROJECT はプロジェクトの名前で、以下の値のどれかです。

- ceilometer - Telemetry API。
- cinder - Block Storage API and extensions.
- glance - Image Service API。
- heat - Orchestration API。

- keystone - Identity Service API とその拡張。
- neutron - Networking API。
- nova - Compute API とその拡張。
- swift - Object Storage API。

たとえば、nova クライアントをインストールする場合、このコマンドを実行します。

```
# pip install python-novaclient
```

nova クライアントを削除する場合、このコマンドを実行します。

```
# pip uninstall python-novaclient
```



注記

To upgrade a package, add the `--upgrade` option to the `pip` command.

たとえば、nova クライアントを更新する場合、このコマンドを実行します。

```
# pip install --upgrade python-novaclient
```

OpenStack RC ファイル

OpenStack コマンドラインクライアントに必要な環境変数を設定するために、環境ファイルを作成する必要があります。このプロジェクト固有の環境ファイルは、すべての OpenStack サービスを使用するクレデンシャルを含みます。

このファイルを読み込むと、環境変数が現在のシェルに対して設定されます。この変数により OpenStack クライアントコマンドがクラウドで実行中の OpenStack サービスとやりとりできるようになります。



Microsoft Windows における環境変数

環境変数ファイルを用いて環境変数を定義することは、Microsoft Windows で一般的な手法ではありません。環境変数は通常、システムのプロパティダイアログの詳細設定タブで定義されます。

OpenStack RC ファイルの作成と読み込み

1. `openrc.sh` ファイルを作成し、認証情報を追加します。

```
export OS_USERNAME=admin  
export OS_PASSWORD=ADMIN_PASS  
export OS_TENANT_NAME=admin  
export OS_AUTH_URL=http://controller:35357/v2.0
```

2. OpenStack コマンドを実行したいシェルで、それぞれのプロジェクト用の `openrc.sh` ファイルを読み込みます。

```
$ source openrc.sh
```

環境変数値の上書き

When you run OpenStack client commands, you can override some environment variable settings by using the options that are listed at the end of the nova help output. For example, you can override the OS_PASSWORD setting in the openrc.sh file by specifying a password on a nova command, as follows:

```
$ nova --password <password> image-list
```

ここで password はお使いのパスワードです。

第5章 Image Service の設定

目次

Image Service の概要	26
Image Service のインストール	27
Image Service のインストールの検証	29

OpenStack Image Service により、ユーザーが仮想マシンイメージを検索、登録、取得できるようになります。Glance プロジェクトとしても知られている Image Service は、仮想マシンイメージを問い合わせ、実際のイメージを取得できるよう、REST API を提供します。Image Service 経由で利用可能な仮想マシンイメージは、単なるファイルシステムから OpenStack Object Storage のようなオブジェクトストレージシステムまで、さまざまな場所に保存できます。



重要

簡単のため、このガイドは Image Service が file バックエンドを使用するように設定します。これは、Image Service にアップロードされたイメージがこのサービスをホストしているシステムにあるディレクトリに保存されることを意味します。このディレクトリはデフォルトで `/var/lib/glance/images/` です。

続行する前に、仮想マシンイメージとスナップショットを保存するために、このディレクトリに十分な空き容量がシステムにあることを確認してください。最小限で、検証環境で Image Service により使用するために、数ギガバイトの容量が利用可能であるべきです。

Image Service の概要

Image Service は以下のコンポーネントを含みます。

- `glance-api`。イメージの検索・取得・保存に対する Image API コールを受け付けます。
- `glance-registry`。イメージに関するメタデータを保存・処理・取得します。メタデータは容量や形式などの項目を含みます。
- データベース。イメージのメタデータを保存します。お好みに合わせてデータベースを選択できます。多くの環境では MySQL か SQLite を使用します。
- Storage repository for image files. The Image Service supports a variety of repositories including normal file systems, Object Storage, RADOS block devices, HTTP, and Amazon S3. Some types of repositories support only read-only usage.

キャッシュをサポートするために Image Service で実行されるいくつかの定期的なプロセス。複製サービスにより、クラスター全体で一貫性と可用性が確保されます。他の定期的なプロセスにオーディター、アップデーター、リーパーなどがあります。

図1.1「概念アーキテクチャ」 [3]に示されているように、Image Service は IaaS 全体像の中で中心になります。エンドユーザーや Compute のコンポーネントからイメージやイメージのメタデータに対する API リクエストを受け付けます。また、そのディスクファイルを Object Storage Service に保存できます。

Image Service のインストール

OpenStack Image Service は仮想ディスクイメージの登録管理者として動作します。ユーザーは新しいイメージを追加できます。イメージのスナップショットを既存のサーバーの直接ストレージから取得できます。バックアップのため、または新しいサーバーを起動するためのテンプレートとしてスナップショットを使用します。登録済みイメージを Object Storage に保存できます。たとえば、イメージをシンプルなファイルシステムや外部ウェブサーバーに保存できます。



注記

この手順は「[Identity Service のインストールの検証](#)」 [19] に記載されているとおり、適切な環境変数にクレデンシャルを設定していると仮定しています。

1. コントローラーノードに Image Service をインストールします。

```
# apt-get install glance python-glanceclient
```

2. Image Service はイメージに関する情報をデータベースに保存します。このガイドの例は、他の OpenStack サービスにより使用されている MySQL データベースを使用します。

データベースの位置を設定します。Image Service はそれぞれの設定ファイルを用いて glance-api サービスと glance-registry サービスを提供します。このセクションを通して両方の設定ファイルを更新する必要があります。GLANCE_DBPASS をお使いの Image Service データベースのパスワードで置き換えます。

/etc/glance/glance-api.conf と /etc/glance/glance-registry.conf を編集し、[database] セクションを変更します。

```
...
[database]
...
# SQLAlchemy connection string for the reference implementation
# registry server. Any valid SQLAlchemy connection string is fine.
# See: http://www.sqlalchemy.org/docs/05/reference/sqlalchemy/connections.
html#sqlalchemy.create_engine
connection = mysql://glance:GLANCE_DBPASS@controller/glance
...
```

3. Ubuntu パッケージはデフォルトで SQLite データベースを作成します。誤って使用されないように、/var/lib/glance/ に作成された glance.sqlite ファイルを削除します。
4. root としてログインするために、作成したパスワードを使用します。glance データベースユーザーを作成します。

```
$ mysql -u root -p
```

```
mysql> CREATE DATABASE glance;
mysql> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'localhost' ¥
IDENTIFIED BY 'GLANCE_DBPASS';
mysql> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%' ¥
IDENTIFIED BY 'GLANCE_DBPASS';
```

5. Image Service 用のデータベーステーブルを作成します。

```
# glance-manage db_sync
```

6. Image Service が Identity Service で認証するために使用する glance ユーザーを作成します。service プロジェクトを使用し、ユーザーに admin ロールを与えます。

```
$ keystone user-create --name=glance --pass=GLANCE_PASS ¥
--email=glance@example.com
$ keystone user-role-add --user=glance --tenant=service --role=admin
```

7. Image Service が認証用に Identity Service を使用するよう設定します。

/etc/glance/glance-api.conf ファイルと /etc/glance/glance-registry.conf ファイルを編集します。Identity Service で glance ユーザー用に選択したパスワードで GLANCE_PASS を置き換えます。

- a. 以下のキーを [keystone_authtoken] セクションに追加します。

```
[keystone_authtoken]
...
auth_uri = http://controller:5000
auth_host = controller
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = glance
admin_password = GLANCE_PASS
```

- b. 以下のキーを [paste_deploy] セクションに追加します。

```
[paste_deploy]
...
flavor = keystone
```

8. 他の OpenStack サービスから使用できるように、Image Service を Identity Service に登録します。サービスを登録し、エンドポイントを作成します。

```
$ keystone service-create --name=glance --type=image ¥
--description="OpenStack Image Service"
$ keystone endpoint-create ¥
--service-id=$(keystone service-list | awk '/ image / {print $2}') ¥
--publicurl=http://controller:9292 ¥
--internalurl=http://controller:9292 ¥
--adminurl=http://controller:9292
```

9. 新しい設定を用いて glance サービスを再起動します。

```
# service glance-registry restart
# service glance-api restart
```

Image Service のインストールの検証

Image Service のインストールをテストするために、OpenStack で動作することが知られている仮想マシンイメージを何かしらダウンロードします。たとえば、Cirros ([Cirros ダウンロード](#)) は OpenStack 環境をテストするためによく使用される小さなテストイメージです。ここでは 64 ビット Cirros QCOW2 イメージを使用します。

ダウンロード方法とイメージ構築の詳細は[OpenStack 仮想マシンイメージガイド](#)を参照してください。イメージの管理方法の詳細は[OpenStack ユーザーガイド](#)を参照してください。

1. Download the image into a dedicated directory using wget or curl:

```
$ mkdir images
$ cd images/
$ wget http://cdn.download.cirros-cloud.net/0.3.1/cirros-0.3.1-x86_64-disk.img
```

2. イメージを Image Service にアップロードします。

```
$ glance image-create --name=imageLabel --disk-format=fileFormat \
--container-format=containerFormat --is-public=accessValue < imageFile
```

各項目:

imageLabel	任意のラベル。ユーザーがイメージを参照する名前。
fileFormat	イメージファイルの形式を指定します。有効な形式は qcow2, raw, vhd, vmdk, vdi, iso, aki, ari, ami です。

You can verify the format using the file command:

```
$ file cirros-0.3.1-x86_64-disk.img
cirros-0.3.1-x86_64-disk.img: QEMU QCOW Image (v2), 41126400 bytes
```

containerFormat	コンテナの形式を指定します。有効な形式は bare, ovf, aki, ari, ami です。
-----------------	---

仮想マシンに関するメタデータを含むイメージファイルがファイル形式ではないことを示すために bare を指定します。この項目が現在必須となっていますが、実際はすべての OpenStack により使用されるわけではなく、システム動作に影響を与えません。この値がどこでも使用されないため、常に bare をコンテナ形式として指定すると安全です。

accessValue	イメージのアクセス権を指定します。 <ul style="list-style-type: none">• true - すべてのユーザーがイメージを表示および使用できます。• false - 管理者のみがイメージを表示および使用できます。
imageFile	ダウンロードしたイメージファイルの名前を指定します。

例:

```
$ source openrc.sh
$ glance image-create --name="Cirros 0.3.1" --disk-format=qcow2 \
  --container-format=bare --is-public=true < cirros-0.3.1-x86_64-disk.img
```

Property	Value
checksum	d972013792949d0d3ba628fbe8685bce
container_format	bare
created_at	2013-10-08T18:59:18
deleted	False
deleted_at	None
disk_format	qcow2
id	acafc7c0-40aa-4026-9673-b879898e1fc2
is_public	True
min_disk	0
min_ram	0
name	Cirros 0.3.1
owner	efa984b0a914450e9a47788ad330699d
protected	False
size	13147648
status	active
updated_at	2013-05-08T18:59:18



注記

返されたイメージ ID は動的に変更されるため、導入環境によりこの例で示されているものと異なる ID が生成されます。

3. イメージがアップロードされたことを確認し、その属性を表示します。

```
$ glance image-list
```

ID	Name	Disk Format	Container Format
Size Status			
acafc7c0-40aa-4026-9673-b879898e1fc2	Cirros 0.3.1	qcow2	bare
13147648 active			

代わりに、Image Service にアップロードしたものは、`--copy-from` パラメーターを使用することにより、ファイルを保存するためのローカルディスク領域を使用する必要なく実行できます。

例:

```
$ glance image-create --name="Cirros 0.3.1" --disk-format=qcow2 \
  --container-format=bare --is-public=true --copy-from http://cdn.download.cirros-cloud.net/0.3.1/cirros-0.3.1-x86_64-disk.img
```

Property	Value
checksum	d972013792949d0d3ba628fbe8685bce
container_format	bare

created_at	2014-03-05T06:13:18
deleted	False
disk_format	qcow2
id	3cce1e32-0971-4958-9719-1f92064d4f54
is_public	True
min_disk	0
min_ram	0
name	Cirros 0.3.1
owner	e519b772cb43474582fa303da62559e5
protected	False
size	13147648
status	active
updated_at	2014-03-05T06:13:20

第6章 Compute Service の設定

目次

Compute Service	32
Compute コントローラーサービスのインストール	35
コンピュータノードの設定	37
インスタンスの起動	39

Compute Service

Compute Service はクラウドコンピューティングのファブリックコントローラーです。これは Iaas システムの中心部です。クラウドコンピューティングシステムをホストして管理するために使用します。主要なモジュールは Python で実装されます。

Compute は、認証のために Identity Service と、イメージのために Image Service と、ユーザーと管理者のインターフェースのために Dashboard とやりとりします。イメージへのアクセスはプロジェクトやユーザーにより制限されます。クォータはプロジェクトごとに制限されます（例：インスタンス数）。Compute Service は、標準的なハードウェアで水平的にスケールし、必要に応じてインスタンスを起動するためにイメージをダウンロードします。

The Compute service is made up of the following functional areas and their underlying components:

API

- nova-api サービス。エンドユーザーの Compute API コールを受け付けて処理します。OpenStack Compute API、Amazon EC2 API、および管理操作を実行するための特権ユーザー用の特別な Admin API をサポートします。また、インスタンスの実行やいくつかのポリシーの強制など、多くのオーケストレーション作業を開始します。
- nova-api-metadata サービス。インスタンスからメタデータリクエストを受け取ります。nova-api-metadata サービスは一般的に、nova-network を用いてマルチホストモードで実行しているときのみ使用されます。詳細は [クラウド管理者ガイドのメタデータサービス](#) を参照してください。

Debian システムの場合、nova-api パッケージに含まれます。debconf 経由で選択できます。

Compute コア

- nova-compute プロセス。ハイパーバイザーの API 経由で仮想マシンインスタンスを作成および終了するワーカーデーモンです。たとえば、XenServer/XCP 用の XenAPI、KVM や QEMU 用の libvirt、VMware 用の VMwareAPI などです。そのように実行されるプロセスはかなり複雑ですが、基本はシンプルです。キューから操作を受け取り、KVM インスタンスの起動などの一連のシステムコマンドを実行し、データベースで状態を更新している間にそれらを実施します。

- nova-scheduler プロセス。Compute のコードの中で概念的に最も簡単なものです。キューから仮想マシンインスタンスのリクエストを受け取り、どのコンピュートノードで実行すべきかを判断します。
- nova-conductor モジュール。nova-compute とデータベースの間のやりとりを取り次ぎます。nova-compute により行われるクラウドデータベースへの直接アクセスを削減することが目標です。nova-conductor モジュールは水平的にスケールします。しかしながら、nova-compute を実行しているノードに導入しません。詳細は [A new Nova service: nova-conductor](#) を参照してください。

仮想マシン用ネットワーク

- nova-network ワーカーデーモン。nova-compute と同じように、キューからネットワークのタスクを受け取り、ネットワークを操作するためにタスクを実行します。ブリッジインターフェースのセットアップや iptables ルールの変更などです。この機能は別の OpenStack サービスである OpenStack Networking に移行されています。
- nova-dhcpbridge スクリプト。dnsmasq dhcp-script 機能を使用して、IP アドレスのリース情報を追跡し、それらをデータベースに記録します。この機能は OpenStack Networking に移行されています。OpenStack Networking は別のスクリプトを提供します。

コンソールインターフェース

- nova-consoleauth デーモン。コンソールプロキシを提供するユーザーのトークンを認可します。nova-novncproxy と nova-xvpncproxy を参照してください。このサービスはコンソールプロキシを動作させるために実行する必要があります。どちらの種類の多くのプロキシもクラスター設定で単一の nova-consoleauth サービスに対して実行されます。詳細は [nova-consoleauth について](#) を参照してください。
- nova-novncproxy デーモン。VNC 接続で実行中の仮想マシンにアクセスするためのプロキシを提供します。ブラウザーベースの novnc クライアントをサポートします。
- nova-console デーモン。Grizzly で非推奨になりました。代わりに nova-xvpncproxy が使用されます。
- nova-xvpncproxy デーモン。VNC 接続で実行中の仮想マシンにアクセスするためのプロキシを提供します。OpenStack 向けに特別に設計された Java クライアントをサポートします。
- nova-cert デーモン。x509 証明書を管理します。

イメージ管理 (EC2 シナリオ)

- nova-objectstore デーモン。イメージを Image Service に登録するための S3 インターフェースを提供します。主に euca2ools をサポートする必要があるインストール環境のために使用されます。euca2ools は S3 言語 で nova-objectstore とやりとりします。また、nova-objectstore は S3 リクエストを Image Service リクエストに変換します。
- euca2ools クライアント。クラウドリソースを管理するための一組のコマンドラインインタプリターコマンドです。OpenStack のモジュールではありませんが、この EC2 インターフェースをサポートするために、nova-api を設定できます。詳細は [Eucalyptus 2.0 のドキュメント](#) を参照してください。

コマンドラインクライアントと他のインターフェース

- nova クライアント。ユーザーがプロジェクト管理者やエンドユーザーとしてコマンドを投入できます。
- nova-manage クライアント。クラウド管理者がコマンドを投入できます。

他のコンポーネント

- キュー。デーモン間でメッセージを受け渡しするための中央ハブです。一般的に [RabbitMQ](#) で実装されますが、[Apache Qpid](#) や [Zero MQ](#) のような何らかの AMPQ メッセージキューで構いません。
- SQL database. Stores most build-time and runtime states for a cloud infrastructure. Includes instance types that are available for use, instances in use, available networks, and projects. Theoretically, OpenStack Compute can support any database that SQL-Alchemy supports, but the only databases widely used are SQLite3 databases (only appropriate for test and development work), MySQL, and PostgreSQL.

The Compute service interacts with other OpenStack services: Identity Service for authentication, Image Service for images, and the OpenStack dashboard for a web interface.

Compute コントローラーサービスのインストール

Compute は仮想マシンインスタンスを起動できるようにするためのサービス群です。これらのサービスを別々のノードで実行することも同じノードで実行することも設定できます。このガイドでは、多くのサービスはコントローラーノードで実行し、仮想マシンを起動するサービスはコンピュート専用ノードで実行します。このセクションは、コントローラーノードにこれらのサービスをインストールし、設定する方法を示します。

1. コントローラーノードに必要な Compute のパッケージをインストールします。

```
# apt-get install nova-api nova-cert nova-conductor nova-consoleauth \
nova-novncproxy nova-scheduler python-novaclient
```

2. Compute は情報を保存するためにデータベースを使用します。このガイドでは、コントローラーノードで MySQL データベースを使用します。Compute をデータベースの位置とクレデンシャルで設定します。NOVA_DBPASS を後のステップで作成するデータベース用パスワードで置き換えます。

このキーを変更するために /etc/nova/nova.conf ファイルの [database] セクションを編集します。

```
[database]
connection = mysql://nova:NOVA_DBPASS@controller/nova
```

3. /etc/nova/nova.conf ファイルの [DEFAULT] 設定グループにこれらの設定キーを設定することにより、Compute サービスが RabbitMQ メッセージブローカーを使用するように設定します。

```
rpc_backend = nova.rpc.impl_kombu
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

4. Ubuntu パッケージはデフォルトで SQLite データベースを作成します。誤って使用されないように、/var/lib/nova/ に作成された nova.sqlite ファイルを削除します。
5. root としてログインするために、前に作成したパスワードを使用します。nova データベースユーザーを作成します。

```
$ mysql -u root -p
mysql> CREATE DATABASE nova;
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost' \
IDENTIFIED BY 'NOVA_DBPASS';
mysql> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' \
IDENTIFIED BY 'NOVA_DBPASS';
```

6. Compute サービスのテーブルを作成します。

```
# nova-manage db sync
```

7. my_ip、vncserver_listen、vncserver_proxyclient_address 設定オプションをコントローラーノードの内部 IP アドレスに設定します。

/etc/nova/nova.conf ファイルを編集し、[DEFAULT] セクションにこれらの行を追加します。

```
...
[DEFAULT]
...
my_ip=192.168.0.10
vncserver_listen=192.168.0.10
vncserver_proxyclient_address=192.168.0.10
```

8. Compute が Identity Service で認証するために使用する nova ユーザーを作成します。service プロジェクトを使用し、ユーザーに admin ロールを与えます。

```
$ keystone user-create --name=nova --pass=NOVA_PASS --email=nova@example.com
$ keystone user-role-add --user=nova --tenant=service --role=admin
```

9. コントローラーで実行している Identity Service でこれらのクレデンシャルを使用するよう Compute を設定します。NOVA_PASS をお使いの Compute パスワードで置き換えます。

このキーを追加するために /etc/nova/nova.conf ファイルの [DEFAULT] セクションを編集します。

```
[DEFAULT]
...
auth_strategy=keystone
```

これらのキーを [keystone_authtoken] セクションに追加します。

```
[keystone_authtoken]
...
auth_uri = http://controller:5000
auth_host = controller
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = NOVA_PASS
```

10. 他の OpenStack サービスから使用できるように、Compute を Identity Service に登録します。サービスを登録し、エンドポイントを指定します。

```
$ keystone service-create --name=nova --type=compute ¥
--description="OpenStack Compute"
$ keystone endpoint-create ¥
--service-id=$(keystone service-list | awk '/ compute / {print $2}') ¥
--publicurl=http://controller:8774/v2/%(tenant_id)s ¥
--internalurl=http://controller:8774/v2/%(tenant_id)s ¥
--adminurl=http://controller:8774/v2/%(tenant_id)s ¥
```

11. Compute サービスを再起動します。

```
# service nova-api restart
# service nova-cert restart
# service nova-consoleauth restart
# service nova-scheduler restart
```

```
# service nova-conductor restart
# service nova-novncproxy restart
```

12. 設定を検証するために、使用可能なイメージを一覧表示します。

```
$ nova image-list
```

ID	Name	Status	Server
acafc7c0-40aa-4026-9673-b879898e1fc2	Cirros 0.3.1	ACTIVE	

コンピュートノードの設定

コントローラーノードで Compute サービスを設定した後、他のシステムをコンピュートノードとして設定する必要があります。コンピュートノードはコントローラーノードからリクエストを受け取り、仮想マシンインスタンスをホストします。単一ノードですべてのサービスを実行することもできます。しかし、このガイドの例では分離したシステムを使用します。これにより、このセクションにある説明に従って、追加のコンピュートノードを追加して、水平的にスケールさせることが容易になります。

Compute サービスは仮想マシンインスタンスを実行するためにハイパーバイザーに依存します。OpenStack はさまざまなハイパーバイザーを使用できますが、このガイドは KVM を使用します。

1. システムを設定します。[2章オペレーティングシステムの基本設定 \[7\]](#)にある方法を使用します。以下の項目はコントローラーノードと異なることに注意してください。
 - eth0 を設定するとき、違う IP アドレスを使用します。このガイドは内部ネットワーク用に 192.168.0.11 を使用します。eth1 を静的 IP アドレスで設定しないでください。OpenStack のネットワークコンポーネントが IP アドレスを割り当て、設定します。
 - ホスト名を compute1 に設定します。確認するために、`uname -n` を使用します。両方のノードの IP アドレスとホスト名が各システムの `/etc/hosts` にあることを確認します。
 - コントローラーノードから同期します。「[Network Time Protocol \(NTP\)](#)」[\[9\]](#)にある手順に従ってください。
 - MySQL クライアントライブラリをインストールします。MySQL データベースサーバーをインストールする必要や MySQL サービスを起動する必要がありません。
 - 使用しているディストリビューションの OpenStack パッケージを有効化します。「[OpenStack パッケージ](#)」[\[11\]](#)を参照してください。
2. オペレーティングシステムの設定後、Compute サービス向けに適切なパッケージをインストールします。

このコマンドを実行します。

```
# apt-get install nova-compute-kvm python-guestfs
```

When prompted to create a supermin appliance, respond yes.

3. For security reasons, the Linux kernel is not readable by normal users which restricts hypervisor services such as qemu and libguestfs. For details, see [this bug](#). To make the current kernel readable, run:

```
# dpkg-statoverride --update --add root root 0644 /boot/vmlinuz-$(uname -r)
```

To also enable this override for all future kernel updates, create the file `/etc/kernel/postinst.d/statoverride` containing:

```
#!/bin/sh
version="$1"
# passing the kernel version is required
[ -z "${version}" ] && exit 0
dpkg-statoverride --update --add root root 0644 /boot/vmlinuz-${version}
```

Remember to make the file executable:

```
# chmod +x /etc/kernel/postinst.d/statoverride
```

4. `/etc/nova/nova.conf` 設定ファイルを編集し、これらの行を適切なセクションに追加します。

```
...
[DEFAULT]
...
auth_strategy=keystone
...
[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://nova:NOVA_DBPASS@controller/nova

[keystone_authtoken]
auth_uri = http://controller:5000
auth_host = controller
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = NOVA_PASS
```

5. `/etc/nova/nova.conf` ファイルの `[DEFAULT]` 設定グループにこれらの設定キーを設定することにより、Compute サービスが RabbitMQ メッセージブローカーを使用するよう設定します。

```
rpc_backend = nova.rpc.impl_kombu
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

6. インスタンスへのリモートコンソールアクセスを提供するよう Compute を設定します。

`/etc/nova/nova.conf` を編集し、以下のキーを `[DEFAULT]` セクションに追加します。

```
[DEFAULT]
...
my_ip=192.168.0.11
vnc_enabled=True
vncserver_listen=0.0.0.0
vncserver_proxyclient_address=192.168.0.11
novncproxy_base_url=http://controller:6080/vnc_auto.html
```

7. Image Service を実行するホストを指定します。/etc/nova/nova.conf ファイルを編集し、これらの行を [DEFAULT] セクションに追加します。

```
[DEFAULT]
...
glance_host=controller
```

8. Compute をテスト目的で仮想マシンにインストールする場合、ハイパーバイザーと CPU がネストハードウェア支援をサポートするかどうかを、以下のコマンドを使用して確認する必要があります。

```
# egrep -c '(vmx|svm)' /proc/cpuinfo
```

このコマンドが 1 以上の値を返す場合、ハイパーバイザーと CPU がネストハードウェア支援をサポートし、追加の設定は必要ありません。

このコマンドが 0 を返したならば、ハイパーバイザーと CPU がネストハードウェア支援をサポートしません。libvirt は KVM の代わりに QEMU を使用する必要があります。この項目を変更するために /etc/nova/nova-compute.conf ファイルの [libvirt] セクションを編集します。

```
[libvirt]
...
virt_type = qemu
```

9. Compute Service を再起動します。

```
# service nova-compute restart
```

10. パッケージにより作成された SQLite データベースを削除します。

```
# rm /var/lib/nova/nova.sqlite
```

インスタンスの起動



警告

Icehouse 向けにこのドキュメントを更新中です。この作業中、構造や内容の問題が見つかるかもしれません。

Compute サービスを設定した後、インスタンスを起動できます。インスタンスは OpenStack が Compute サーバーに展開する仮想マシンです。この例はダウンロードしたイメージを使用することにより、小さなインスタンスを起動する方法を示します。



注記

この手順は以下を仮定します。

- コマンドを実行するマシンに nova クライアントライブラリがインストールされていること（不確かならばコントローラーにログインしてください）。
- クレデンシャルを指定する環境変数が設定されていること。「[Identity Service のインストールの検証](#)」[19] 参照。
- イメージがダウンロードされていること。「[Image Service のインストールの検証](#)」[29] 参照。
- Configured networking. See [7章Add a networking service](#) [45].

1. OpenStack でインスタンスを起動できるようにするために、秘密鍵と公開鍵から構成されるキーペアを生成します。これらのキーはインスタンスにパスワード無しでアクセスできるようにするために、インスタンスの中に注入されます。これは必要なツールがイメージの中に同梱されている方法に依存します。詳細は [OpenStack 管理ユーザーガイド](#)を参照してください。

```
$ ssh-keygen
$ cd ~/.ssh
$ nova keypair-add --pub_key id_rsa.pub mykey
```

これで mykey キーペアを作成しました。id_rsa 秘密鍵がローカルの ~/.ssh に保存されます。キーペアとして mykey を使用して起動したインスタンスに接続するために使用できます。次のとおり利用可能なキーペアを表示します。

```
$ nova keypair-list
```

Name	Fingerprint
mykey	b0:18:32:fa:4e:d4:3c:1b:c4:6c:dd:cb:53:29:13:82

2. インスタンスを起動するために、このインスタンス用に使用したいフレーバーの ID を指定する必要があります。フレーバーはリソース割り当てプロファイルです。たとえば、インスタンスがどのくらいの仮想 CPU を持つか、どのくらいのメモリを持つかを指定します。次のとおり利用可能なプロファイルの一覧を参照します。

```
$ nova flavor-list
```

ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor	Is_Public
1	m1.tiny	512	1	0		1	1.0	True
2	m1.small	2048	20	0		1	1.0	True
3	m1.medium	4096	40	0		2	1.0	True
4	m1.large	8192	80	0		4	1.0	True
5	m1.xlarge	16384	160	0		8	1.0	True

3. インスタンスに使用するイメージの ID を取得します。

```
$ nova image-list
```

ID	Name	Status	Server
9e5c2bee-0373-414c-b4af-b91b0246ad3b	Cirros 0.3.1	ACTIVE	

4. SSH と ping を使用するために、セキュリティグループのルールを設定する必要があります。[OpenStack ユーザーガイド](#)を参照してください。

```
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
```

5. インスタンスを起動します。

```
$ nova boot --flavor flavorType --key_name keypairName --image ID newInstanceName
```

フレーバー 1 か 2 を使用してインスタンスを作成します。例:

```
$ nova boot --flavor 1 --key_name mykey --image 9e5c2bee-0373-414c-b4af-b91b0246ad3b --security_group default cirrOS
```

Property	Value
OS-EXT-STS:task_state	scheduling
image	Cirros 0.3.1
OS-EXT-STS:vm_state	building
OS-EXT-SRV-ATTR:instance_name	instance-00000001
OS-SRV-USG:launched_at	None
flavor	m1.tiny
id	3bdf98a0-c767-4247-bf41-2d147e4aa043
security_groups	[{'name': 'default'}]
user_id	530166901fa24d1face95cda82cfae56
OS-DCF:diskConfig	MANUAL
accessIPv4	
accessIPv6	
progress	0
OS-EXT-STS:power_state	0
OS-EXT-AZ:availability_zone	nova
config_drive	
status	BUILD
updated	2013-10-10T06:47:26Z
hostId	
OS-EXT-SRV-ATTR:host	None
OS-SRV-USG:terminated_at	None
key_name	mykey
OS-EXT-SRV-ATTR:hypervisor_hostname	None
name	cirrOS
adminPass	DWCDW6FnsKNq
tenant_id	e66d97ac1b704897853412fc8450f7b9
created	2013-10-10T06:47:23Z
os-extended-volumes:volumes_attached	[]
metadata	{}



注記

インスタンスに十分なメモリが利用できない場合、Compute はインスタンスを作成しますが、起動しません。インスタンスの状態が ERROR に設定されます。

6. After the instance launches, use the `nova list` to view its status. The status changes from BUILD to ACTIVE:

```
$ nova list
+-----+-----+-----+-----+-----+
+-----+
| ID              | Name      | Status | Task State | Power State |
+-----+-----+-----+-----+-----+
| Networks       |           |        |            |             |
+-----+-----+-----+-----+-----+
| dcc4a894-869b-479a-a24a-659eef7a54bd | cirrOS    | BUILD  | spawning  | NOSTATE     |
| vmnet=10.0.0.3 |           |        |            |             |
+-----+-----+-----+-----+-----+
$ nova list
+-----+-----+-----+-----+-----+
+-----+
| ID              | Name      | Status | Task State | Power State |
+-----+-----+-----+-----+-----+
| Networks       |           |        |            |             |
+-----+-----+-----+-----+-----+
| dcc4a894-869b-479a-a24a-659eef7a54bd | cirrOS    | ACTIVE | None       | Running     |
| vmnet=10.0.0.3 |           |        |            |             |
+-----+-----+-----+-----+-----+
```



注記

指定したインスタンスの詳細を表示する方法:

```
$ nova show dcc4a894-869b-479a-a24a-659eef7a54bd
+-----+-----+
+-----+
| Property              | Value                                     |
+-----+-----+
| status                | ACTIVE                                  |
+-----+-----+
| updated               | 2013-10-16T21:55:24Z                    |
+-----+-----+
| OS-EXT-STS:task_state | None                                     |
+-----+-----+
| OS-EXT-SRV-ATTR:host  | compute-node                            |
+-----+-----+
| key_name              | mykey                                   |
+-----+-----+
| image                 | cirros                                  |
| (918a1017-8a1b-41ff-8809-6106ba45366e) |                                         |
+-----+-----+
| vmnet network         | 10.0.0.3                                |
+-----+-----+
```

```
| hostId |
| 306d7c693911170ad4e5218f626f531cc68caa45f3a0f70f1aeba94d |
| OS-EXT-STS:vm_state | active |
| OS-EXT-SRV-ATTR:instance_name | instance-0000000a |
| OS-SRV-USG:launched_at | 2013-10-16T21:55:24.000000 |
| OS-EXT-SRV-ATTR:hypervisor_hostname | compute-node |
| flavor | m1.tiny (1) |
| id | dcc4a894-869b-479a-a24a-659eef7a54bd |
| security_groups | [{u'name': u'default'}] |
| OS-SRV-USG:terminated_at | None |
| user_id | 887ac8736b5b473b9dc3c5430a88b15f |
| name | cirrOS |
| created | 2013-10-16T21:54:52Z |
| tenant_id | 43ab520b2b484578bb6924c0ea926190 |
| OS-DCF:diskConfig | MANUAL |
| metadata | {} |
| os-extended-volumes:volumes_attached | [] |
| accessIPv4 | |
| accessIPv6 | |
| progress | 0 |
| OS-EXT-STS:power_state | 1 |
| OS-EXT-AZ:availability_zone | nova |
| config_drive | |
+-----+
+-----+
```

- After the instance boots and initializes and you have configured security groups, you can ssh into the instance without a password by using the keypair you specified in the nova boot command. Use the nova list command to get the IP address for the instance. You do not need to specify the private key because it was stored in the default location, ~/.ssh/.id_rsa, for the ssh client.



注記

インスタンスを起動するために CirrOS イメージを使用しているならば、root ユーザーではなく、cirros ユーザーとしてログインする必要があります。

cubswin:) パスワードを使用することにより、SSH キーなしで cirros アカウントにログインすることもできます。

```
$ ssh cirros@10.0.0.3
```

第7章 Add a networking service

目次

Networking (neutron)	45
Legacy networking	62



警告

Icehouse 向けにこのドキュメントを更新中です。この作業中、構造や内容の問題が見つかるかもしれません。

Configuring networking in OpenStack can be a bewildering experience. This guide provides step-by-step instructions for both OpenStack Networking (neutron) and the legacy (nova) networking service. If you are unsure which to use, we recommend trying OpenStack Networking because it offers a considerable number of features and flexibility including plug-ins for a variety of emerging products supporting virtual networking. See the [Networking](#) chapter of the OpenStack Cloud Administrator Guide for more information.

Networking (neutron)

Neutron の概念

Nova Networking のように、Neutron は OpenStack インストール環境の SDN を管理します。しかしながら、Nova Networking と異なり、テナントごとのプライベートネットワークなど、より高度な仮想ネットワークポロジー向けに Neutron を設定できます。

Neutron はネットワーク、サブネット、ルーターのオブジェクトの抽象化を実現します。それぞれ、対応する物理的なものの機能を模倣します。ネットワークがサブネットを含みます。ルーターがサブネットやネットワーク間の通信を中継します。

すべての Neutron 環境は少なくとも 1 つの外部ネットワークを持ちます。このネットワークは、他のネットワークと異なり、ほとんど仮想的に定義されたネットワークではありません。これは OpenStack インストール環境の外部からアクセス可能な外部ネットワークの一部のビューであることを意味します。Neutron 外部ネットワークの IP アドレスは外部ネットワークにある何らかの物理的なものによりアクセスできます。このネットワークがほとんど外部ネットワークの一部を表すため、DHCP はこのネットワークで無効化されます。

外部ネットワークに加えて、あらゆる Neutron のセットアップ環境は 1 つ以上の内部ネットワークを持ちます。これらの SDN は仮想マシンに直接接続します。あらゆる指定された内部ネットワークにある仮想マシン、またはインターフェース経由で同様のルーターに接続されたサブネットにある仮想マシンのみが、そのネットワークに接続された仮想マシンに直接アクセスできます。

外部ネットワークが仮想マシンにアクセスするため、またその逆のため、ネットワーク間のルーターが必要になります。各ルーターはネットワークに接続された 1 つのゲート

ウェイとサブネットに接続された多くのインターフェースを持ちます。物理ルーターのように、同じルーターに接続された他のサブネットにあるマシンにサブネットがアクセスできます。また、マシンはルーターに対するゲートウェイ経由で外部ネットワークにアクセスできます。

さらに、内部ネットワークにたどり着くために外部ネットワークに IP アドレスを割り当てることができます。何かがサブネットに接続されたとき必ず、その接続がポートと呼ばれます。外部ネットワークの IP アドレスを仮想マシンのポートに関連づけられます。このように、外部ネットワークのものが仮想マシンにアクセスできます。

Neutron は セキュリティグループ もサポートします。セキュリティグループにより、管理者がグループでファイアウォールルールを定義できます。仮想マシンは 1 つ以上のセキュリティグループに属します。Neutron が、ポート、ポート範囲、または通信種別をブロックするかブロックしないかのために、これらのセキュリティグループにあるルールを仮想マシンに対して適用します。

Neutron が使用する各プラグインはそれぞれ独自の概念を持ちます。Neutron を稼働させるために必須ではありませんが、これらの概念を理解することにより、Neutron をセットアップする役に立つでしょう。すべての Neutron インストール環境は、コアプラグインとセキュリティグループプラグイン（またはただの No-Op セキュリティグループプラグイン）を使用します。さらに、Firewall-as-a-service (FWaaS) と Load-balancing-as-a-service (LBaaS) プラグインが利用可能です。

Open vSwitch の概念

Open vSwitch プラグインは最も人気のあるコアプラグインの一つです。Open vSwitch の設定はブリッジとポートから構成されます。ポートは物理インターフェースやパッチケーブルのような他のものへの接続を意味します。ブリッジのあらゆるポートからのパケットは、そのブリッジにあるすべての他のポートと共有されます。ブリッジは Open vSwitch 仮想パッチケーブルまたは Linux 仮想イーサネットケーブル (veth) から接続されます。また、ブリッジは Linux にネットワークインターフェースとして認識されるため、それらに IP アドレスを割り当てることができます。

Neutron では、br-int という統合ブリッジが仮想マシンおよび関連するサービスを直接接続します。br-ex という外部ブリッジが外部ネットワークに接続します。最後に、Open vSwitch プラグインの VLAN 設定が各物理ネットワークと関連づけられたブリッジを使用します。

ブリッジの定義に加えて、Open vSwitch は OpenFlow に対応しています。これにより、ネットワークのフロールールを定義できるようになります。特定の設定が VLAN 間のパケットを転送するために、これらのルールを使用します。

最後に、Open vSwitch のいくつかの設定はネットワーク名前空間を使用します。この名前空間により、Linux が他の名前空間に認識されない一意な名前空間の中にアダプターをグループ化できます。これで、同じネットワークノードが複数の Neutron ルーターを管理できるようになります。

Open vSwitch を用いると、仮想ネットワークを作成するために、2 つの異なる技術 GRE と VLAN を使用できます。

Generic Routing Encapsulation (GRE) は多くの VLAN で使用される技術です。異なるルーティング情報を用いて新しいパケット全体を作成するために IP パケットをラップで

きます。新しいパケットがその宛て先に到達したとき、ラップが外され、元のパケットが中継されます。Open vSwitch を用いて GRE を使用するために、Neutron が GRE トンネルを作成します。これらのトンネルはブリッジにポートを作成し、異なるシステムにあるブリッジが 1 つのブリッジのように動作できるようにします。これにより、コンピュータノードとネットワークノードがルーティング目的に 1 つのものとして動作できます。

一方、Virtual LAN (VLAN) はイーサネットヘッダーに対する特別な変更を使用します。1 から 4096 までの範囲で 4 バイトの VLAN タグを追加します (タグ 0 は特別です。すべてのものからなるタグ 4095 はタグなしパケットにあたります)。特別な NIC、スイッチ、ルーターは Open vSwitch が実行するように、VLAN タグを解釈する方法について理解しています。1 つの VLAN にタグ付けされたパケットは、すべてのデバイスが同じ物理ネットワークにあるときさえ、その VLAN 上に設定された他のデバイスのみと共有されます。

Open vSwitch とともに使用される最も一般的なセキュリティグループはハイブリッド iptables/Open vSwitch プラグインです。これは iptables ルールと OpenFlow ルールの組み合わせを使用します。Linux でファイアウォールを作成し、NAT をセットアップするために iptables ツールを使用します。このツールは、Neutron セキュリティグループにより必要となる複雑なルールを実現するために、複雑なルールシステムとルールのチェーンを使用します。

コントローラーノードの設定

前提

個々のノードを Networking 用に設定する前に、必要となる OpenStack コンポーネント (ユーザー、サービス、データベース、1 つ以上のエンドポイント) を作成する必要があります。コントローラーノードでこれらの手順を完了した後、OpenStack Networking ノードをセットアップするために、このガイドにある説明に従います。

1. MySQL データベースに root ユーザーとして接続し、neutron データベースを作成し、適切なアクセス権限を与えます。

```
$ mysql -u root -p
mysql> CREATE DATABASE neutron;
mysql> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'localhost' ¥
IDENTIFIED BY 'NEUTRON_DBPASS';
mysql> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%' ¥
IDENTIFIED BY 'NEUTRON_DBPASS';
```

2. Networking が Identity Service と通信できるよう、必要となるユーザー、サービス、エンドポイントを作成します。

neutron ユーザーを作成します。

```
$ keystone user-create --name=neutron --pass=NEUTRON_PASS --email=neutron@example.com
```

ユーザーロールを neutron ユーザーに追加します。

```
$ keystone user-role-add --user=neutron --tenant=service --role=admin
```

neutron サービスを作成します。

```
$ keystone service-create --name=neutron --type=network ¥
--description="OpenStack Networking"
```


Networking エンドポイントを作成します。

```
$ keystone endpoint-create ¥
--service-id $(keystone service-list | awk '/ network / {print $2}') ¥
--publicurl http://controller:9696 ¥
--adminurl http://controller:9696 ¥
--internalurl http://controller:9696
```

サーバーコンポーネントのインストールと設定

1. Networking および依存関係のあるサーバーコンポーネントをインストールします。

```
# apt-get install neutron-server
```

2. Networking がお使いの MySQL データベースを使用するよう設定します。/etc/neutron/neutron.conf ファイルを編集し、以下のキーを [database] セクションに追加します。Neutron データベース用に選択したパスワードで NEUTRON_PASS を置き換えます。

```
[database]
...
connection = mysql://neutron:NEUTRON_DBPASS@controller/neutron
```

3. Networking が認証用に Identity Service として keystone を使用するよう設定します。

- /etc/neutron/neutron.conf ファイルを編集し、以下のキーを [DEFAULT] セクションに追加します。

```
[DEFAULT]
...
auth_strategy = keystone
```

以下のキーを [keystone_authtoken] セクションに追加します。Keystone で Neutron ユーザー用に選択したパスワードで NEUTRON_PASS を置き換えます。

```
[keystone_authtoken]
...
auth_uri = http://controller:5000
auth_host = controller
auth_protocol = http
auth_port = 35357
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

4. Networking がお使いのメッセージブローカーを使用するよう設定します。/etc/neutron/neutron.conf ファイルを編集し、以下のキーを [DEFAULT] セクションに追加します。

RABBIT_PASS を RabbitMQ 用に選んだパスワードで置き換えます。

```
[DEFAULT]
...
rpc_backend = neutron.openstack.common.rpc.impl_kombu
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

Open vSwitch (OVS) プラグインのインストールと設定

OpenStack Networking はさまざまなプラグインをサポートします。簡単のために、最も一般的なプラグイン Open vSwitch を取り扱うことにし、プロジェクトのネットワーク通信のために基本的な GRE トンネルを使用するよう設定します。

1. Open vSwitch プラグインをインストールします。

```
# apt-get install neutron-plugin-openvswitch
```

2. どのネットワーク技術を Open vSwitch と一緒に使用するかによらず、いくつかの共通設定オプションを設定する必要があります。OVS を使用するために Networking コアを設定する必要があります。/etc/neutron/neutron.conf ファイルを編集します。

```
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.OVSNeutronPluginV2
```



注記

コントローラー専用ノードは Open vSwitch や Open vSwitch エージェントを実行する必要がありません。

3. OVS プラグインに GRE トンネリングを使用するよう設定します。/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini ファイルを編集します。

```
[ovs]
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
```

Compute サービスの Networking 用設定

- Compute が OpenStack Networking サービスを使用するよう設定します。/etc/nova/nova.conf ファイルを設定します。

```
network_api_class=nova.network.neutronv2.api.API
neutron_url=http://controller:9696
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_admin_username=neutron
neutron_admin_password=NEUTRON_PASS
neutron_admin_auth_url=http://controller:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSIfaceDriver
firewall_driver=nova.virt.firewall.NoopFirewallDriver
security_group_api=neutron
```



注記

- ネットワークノードとコンピュートノードを設定するときに、どのファイアウォールドライバーを選択しても、このドライバーを No-Op ファイアウォールとして設定します。このファイアウォールは nova ファイアウォールです。neutron がファイアウォールを取り扱うので、nova にこれを使用しないよう通知する必要があります。

Networking がファイアウォールを取り扱うとき、firewall_driver オプションは指定したプラグインに合わせて設定されるべきです。

たとえば、OVS を用いる場合、`/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` ファイルを編集します。

```
[securitygroup]
# Firewall driver for realizing neutron security group function.
firewall_driver=neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
```

- If you do not want to use a firewall in Compute or Networking, set `firewall_driver=nova.virt.firewall.NoopFirewallDriver` in both config files, and comment out or remove `security_group_api=neutron` in the `/etc/nova/nova.conf` file, otherwise you may encounter `ERROR: The server has either erred or is incapable of performing the requested operation. (HTTP 500)` when issuing `nova list` commands.

インストールの完了

- Compute と Networking のサービスを再起動します。

```
# service nova-api restart
# service nova-scheduler restart
# service nova-conductor restart
# service neutron-server restart
```

ネットワークノードの設定



注記

始める前に、ネットワーク専用ノードとしてマシンをセットアップします。ネットワーク専用ノードは `MGMT_INTERFACE` (管理インターフェース) NIC、`DATA_INTERFACE` (データインターフェース) NIC、`EXTERNAL_INTERFACE` (外部インターフェース) NIC を持ちます。

管理ネットワークはノード間の通信を処理します。データネットワークは仮想マシンとの通信を処理します。外部 NIC は仮想マシンが外部と接続できるようにネットワークノードを接続します。オプションとしてコントローラーノードに接続します。

エージェントのインストールおよび共通コンポーネントの設定

1. Networking パッケージおよび依存関係のあるパッケージをインストールします。

```
# apt-get install neutron-dhcp-agent neutron-l3-agent
```

2. ネットワークノードが仮想マシンの通信を制御できるように、パケット転送を有効化し、パケット宛先フィルタリングを無効化します。`/etc/sysctl.conf` を以下のように編集します。

```
net.ipv4.ip_forward=1
net.ipv4.conf.all.rp_filter=0
net.ipv4.conf.default.rp_filter=0
```

Use the `sysctl` command to ensure the changes made to the `/etc/sysctl.conf` file take effect:

```
# sysctl -p
```



注記

Networking 関連の設定の値を変更した後、Networking Service を再起動することを推奨します。これにより、すべての変更した値がすぐに確実に適用されます。

```
# service networking restart
```

3. 認証に keystone を使用するよう neutron を設定するために /etc/neutron/neutron.conf ファイルを編集します。
 - a. このファイルの DEFAULT セクションで auth_strategy 設定キーを keystone に設定します。

```
auth_strategy = keystone
```

- b. このファイルの [keystone_authtoken] セクションにこれらの行を追加します。

```
[keystone_authtoken]
...
auth_uri = http://controller:5000
auth_host = controller
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

4. RabbitMQ のアクセス権を設定します。DEFAULT セクションにある以下のパラメーターを変更するために、/etc/neutron/neutron.conf ファイルを編集します。

```
rabbit_host = controller
rabbit_userid = guest
rabbit_password = RABBIT_PASS
```

Open vSwitch (OVS) プラグインのインストールと設定

OpenStack Networking はさまざまなプラグインをサポートします。簡単のために、最も一般的なプラグイン Open vSwitch を取り扱うことにし、プロジェクトのネットワーク通信のために基本的な GRE トンネルを使用するよう設定します。

1. Open vSwitch プラグインと依存パッケージをインストールします。

```
# apt-get install neutron-plugin-openvswitch-agent openvswitch-datapath-dkms
```

2. Open vSwitch を起動します。

```
# service openvswitch-switch restart
```

3. どのネットワーク技術を使用するにしても、br-int 統合ブリッジを追加する必要があります。このブリッジは、仮想マシンと、外部に接続する br-ex 外部ブリッジを接続します。

```
# ovs-vsctl add-br br-int
```

```
# ovs-vsctl add-br br-ex
```

- EXTERNAL_INTERFACE インターフェースのポート（接続）を br-ex インターフェースに追加します。

```
# ovs-vsctl add-port br-ex EXTERNAL_INTERFACE
```



警告

ホストは EXTERNAL_INTERFACE 以外にインターフェースと関連づけられた IP アドレスを持つ必要があります。リモートターミナルセッションがこの他の IP アドレスと関連づけられる必要があります。

If you associate an IP address with EXTERNAL_INTERFACE, that IP address stops working after you issue the `ovs-vsctl add-port br-ex EXTERNAL_INTERFACE` command. If you associate a remote terminal session with that IP address, you lose connectivity with the host.

この動作に関する詳細は、[Open vSwitch FAQ](#) の Configuration Problems（接続の問題）を参照してください。

- EXTERNAL_INTERFACE を IP アドレスなしでプロミスカスモードに設定します。さらに、前に EXTERNAL_INTERFACE に含めた IP アドレスを持つよう、新しく作成した br-ex インターフェースを設定する必要があります。



警告

Generic Receive Offload (GRO) はこのインターフェースで有効化すべきではありません。深刻なパフォーマンス問題を引き起こす可能性があるためです。ethtool ユーティリティを用いて無効化できます。

- どのネットワーク技術を Open vSwitch と一緒に使用するかによらず、いくつかの共通設定オプションを設定する必要があります。OVS と名前空間を使用するために L3 エージェントと DHCP エージェントを設定する必要があります。それぞれ `/etc/neutron/l3_agent.ini` と `/etc/neutron/dhcp_agent.ini` ファイルを編集します。

```
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
use_namespaces = True
```



注記

このガイドの例はデフォルトでネットワークの名前空間を有効化するとしても、問題が発生したり、カーネルがそれらをサポートしなかったりする場合、それらを無効化できます。`/etc/neutron/l3_agent.ini` ファイルと `/etc/neutron/dhcp_agent.ini` ファイルをそれぞれ編集します。

```
use_namespaces = False
```

IP アドレスのオーバーラップを無効化するために `/etc/neutron/neutron.conf` を編集します。

```
allow_overlapping_ips = False
```

ネットワーク名前空間が無効化されているとき、各ネットワークノードに対してルーターを一つのみ持つことができ、IP アドレスのオーバーラップがサポートされないことに注意してください。

初期 Neutron 仮想ネットワークとルーターを作成した後、追加のステップを完了する必要があります。

7. 同様に、OVS を使用するよう Neutron コアに通知する必要があります。/etc/neutron/neutron.conf ファイルを編集します。

```
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.OVSNeutronPluginV2
```

8. ファイアウォールプラグインを設定します。OpenStack によりセキュリティグループと呼ばれるファイアウォールルールを強制したくない場合、neutron.agent.firewall.NoopFirewall を使用できます。そうでなければ、Networking ファイアウォールプラグインのどれかを選択できます。最も一般的な選択は OVS-iptables ハイブリッドドライバですが、FWaaS（ファイアウォールアズアサービス）ドライバを使用することもできます。/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini ファイルを編集します。

```
[securitygroup]
# Firewall driver for realizing neutron security group function.
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```



警告

少なくとも No-Op ファイアウォールを使用する必要があります。そうでなければ、Horizon と他の OpenStack サービスが必要となる仮想マシンのブートオプションを取得および設定できません。

9. GRE トンネリング、br-int 統合ブリッジ、br-tun トンネリングブリッジ、DATA_INTERFACE トンネル IP 用ローカル IP を使用するよう OVS プラグインを設定します。/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini ファイルを編集します。

```
[ovs]
...
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
integration_bridge = br-int
tunnel_bridge = br-tun
local_ip = DATA_INTERFACE_IP
```

エージェントの設定

1. SDN で DHCP を実行するために、Networking はいくつかのプラグインをサポートします。しかしながら一般的に、dnsmasq プラグインを使用します。

/etc/neutron/dhcp_agent.ini ファイルを設定します。

```
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
```

2. 仮想マシンが Compute メタデータ情報にアクセスできるようにするために、Networking メタデータエージェントが有効化されて設定される必要があります。エージェントが Compute メタデータサービスのプロキシとして動作します。

Compute サービスと Networking メタデータエージェントの間で共有される秘密鍵を定義するために、コントローラーで `/etc/nova/nova.conf` ファイルを編集します。

[DEFAULT] セクションに追加します。

```
[DEFAULT]
...
neutron_metadata_proxy_shared_secret = METADATA_PASS
service_neutron_metadata_proxy = true
```

nova-api サービスを再起動します。

```
# service nova-api restart
```

ネットワークノードでメタデータエージェント設定を変更します。

`/etc/neutron/metadata_agent.ini` ファイルを編集し、[DEFAULT] セクションを変更します。

```
[DEFAULT]
...
auth_url = http://controller:5000/v2.0
auth_region = regionOne
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
nova_metadata_ip = controller
metadata_proxy_shared_secret = METADATA_PASS
```



注記

`auth_region` の値は大文字小文字を区別します。Keystone で定義されたエンドポイントのリージョンと一致する必要があります。



注記

自己署名証明書を用いた HTTPS 経由で OpenStack Networking API を提供する場合、Networking がサービスカタログから SSL 証明書を検証できないため、メタデータエージェントに追加の設定をする必要があります。

このステートメントを [DEFAULT] セクションに追加します。

```
neutron_insecure = True
```

インストールの完了

- Networking サービスを再起動します。

```
# service neutron-dhcp-agent restart
# service neutron-l3-agent restart
# service neutron-metadata-agent restart
```

```
# service neutron-plugin-openvswitch-agent restart
```

コンピュータノードの設定



注記

このセクションは nova-compute コンポーネントを実行するあらゆるノードのセットアップについて詳細に説明しますが、すべてのネットワークスタックを実行しません。

前提

- Networking Service が仮想マシンへの通信をルーティングできるように、パケット宛先フィルタリング（ルート検証）を無効化します。/etc/sysctl.conf を編集し、変更を有効化するために以下のコマンドを実行します。

```
net.ipv4.conf.all.rp_filter=0  
net.ipv4.conf.default.rp_filter=0
```

```
# sysctl -p
```

Open vSwitch プラグインのインストール

OpenStack Networking はさまざまなプラグインをサポートします。簡単のために、最も一般的なプラグイン Open vSwitch を取り扱うことにし、プロジェクトのネットワーク通信のために基本的な GRE トンネルを使用するよう設定します。

- Open vSwitch プラグインと依存パッケージをインストールします。

```
# apt-get install neutron-plugin-openvswitch-agent openvswitch-datapath-dkms
```

- Open vSwitch を再起動します。

```
# service openvswitch-switch restart
```

- どのネットワーク技術を Open vSwitch と一緒に使用するかによらず、いくつかの共通設定オプションを設定する必要があります。仮想マシンを接続する br-int 統合ブリッジを追加する必要があります。

```
# ovs-vsctl add-br br-int
```

- どのネットワーク技術を OVS と一緒に使用するかによらず、いくつかの共通設定オプションを設定する必要があります。/etc/neutron/neutron.conf ファイルを編集します。

```
core_plugin = neutron.plugins.openvswitch.ovs_neutron_plugin.OVSNeutronPluginV2
```

- 同様にファイアウォールを設定する必要があります。ネットワークノードをセットアップするときに選択したものと同じファイアウォールプラグインを使用すべきです。そうするために、/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini ファイルを編集し、securitygroup の下にある firewall_driver 値をネットワークノードで使用したものと同一値に設定します。たとえば、ハイブリッド OVS iptables プラグインを使用したい場合、設定はこのようなようになります。

```
[securitygroup]  
# Firewall driver for realizing neutron security group function.  
firewall_driver = neutron.agent.linux.iptables_firewall.OVSHybridIptablesFirewallDriver
```




警告

少なくとも No-Op ファイアウォールを使用する必要があります。そうでなければ、Horizon と他の OpenStack サービスが必要となる仮想マシンのブートオプションを取得および設定できません。

6. br-int 統合ブリッジを持つ GRE トンネリング、br-tun トンネリングブリッジ、DATA_INTERFACE の IP のトンネル用ローカル IP を使用するよう OVS プラグインに通知します。/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini ファイルを編集します。

```
[ovs]
...
tenant_network_type = gre
tunnel_id_ranges = 1:1000
enable_tunneling = True
integration_bridge = br-int
tunnel_bridge = br-tun
local_ip = DATA_INTERFACE_IP
```

共通コンポーネントの設定

1. 認証に keystone を使用するよう neutron を設定するために /etc/neutron/neutron.conf ファイルを編集します。
 - a. このファイルの [DEFAULT] セクションで auth_strategy 設定キーを keystone に設定します。

```
[DEFAULT]
...
auth_strategy = keystone
```

- b. このファイルの [keystone_authtoken] セクションにこれらの行を追加します。

```
[keystone_authtoken]
...
auth_uri = http://controller:5000
auth_host = controller
auth_protocol = http
auth_port = 35357
admin_tenant_name = service
admin_user = neutron
admin_password = NEUTRON_PASS
```

2. RabbitMQ のアクセス権を設定します。[DEFAULT] セクションにある以下のパラメーターを変更するために、/etc/neutron/neutron.conf ファイルを編集します。

```
rabbit_host = controller
rabbit_userid = guest
rabbit_password = RABBIT_PASS
```

Compute サービスの Networking 用設定

- OpenStack Compute が OpenStack Networking Service を使用するよう設定します。/etc/nova/nova.conf ファイルを編集します。

```
network_api_class=nova.network.neutronv2.api.API
neutron_url=http://controller:9696
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_admin_username=neutron
neutron_admin_password=NEUTRON_PASS
neutron_admin_auth_url=http://controller:35357/v2.0
linuxnet_interface_driver = nova.network.linux_net.LinuxOVSInterfaceDriver
firewall_driver=nova.virt.firewall.NoopFirewallDriver
security_group_api=neutron
```



注記

- ネットワークとコンピュータノードを設定するときに、どのファイアウォールドライバを選択しても、ファイアウォールドライバを `nova.virt.firewall.NoopFirewallDriver` に設定するために、`/etc/nova/nova.conf` ファイルを編集する必要があります。OpenStack Networking はファイアウォールを取り扱うので、このステートメントは Compute がファイアウォールを使用しないことを指定します。
- Networking にファイアウォールを取り扱わせたい場合、`firewall_driver` オプションをプラグイン用のファイアウォールに設定するために、`/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` ファイルを編集します。たとえば、OVS を使用する場合、ファイルを次のとおり編集します。

```
[securitygroup]
# Firewall driver for realizing neutron security group function.
firewall_driver=neutron.agent.linux.iptables_firewall.
OVSHybridIptablesFirewallDriver
```

- Compute や Networking でファイアウォールを使用したくない場合、両方の設定ファイルを編集し、`firewall_driver=nova.virt.firewall.NoopFirewallDriver` を設定します。また、`/etc/nova/nova.conf` ファイルを編集し、`security_group_api=neutron` ステートメントをコメントアウトまたは削除します。

Otherwise, when you issue `nova list` commands, the `ERROR: The server has either erred or is incapable of performing the requested operation. (HTTP 500)` error might be returned.

インストールの完了

1. Networking サービスを再起動します。

```
# service neutron-plugin-openvswitch-agent restart
```

2. Compute Service を再起動します。

```
# service nova-compute restart
```

初期ネットワークの作成

Before launching your first instance, you must create the necessary virtual network infrastructure to which the instance will connect, including the [external network](#) and [tenant network](#). See [図7.1 「Initial networks」 \[59\]](#). After creating this infrastructure, we recommend that you [verify connectivity](#) and resolve any issues before proceeding further.



External network

The external network typically provides internet access for your instances. By default, this network only allows internet access from instances using Network Address Translation (NAT). You can enable internet access to individual instances using a floating IP address and suitable security group rules. The admin tenant owns this network because it provides external network access for multiple tenants. You must also enable sharing to allow access by those tenants.



注記

Perform these commands on the controller node.

To create the external network:

1. Source the admin tenant credentials:

```
$ source admin-openrc.sh
```

2. Create the network:

```
$ neutron net-create ext-net --shared --router:external=True
```

Like a physical network, a virtual network requires a subnet assigned to it. The external network shares the same subnet and gateway associated with the physical network connected to the external interface on the network node. You should specify an exclusive slice of this subnet for router and floating IP addresses to prevent interference with other devices on the external network.

Replace FLOATING_IP_START and FLOATING_IP_END with the first and last IP addresses of the range that you want to allocate for floating IP addresses. Replace EXTERNAL_NETWORK_CIDR with the subnet associated with the physical network. Replace EXTERNAL_NETWORK_GATEWAY with the gateway associated with the physical network, typically the ".1" IP address. You should disable DHCP on this subnet because instances do not connect directly to the external network and floating IP addresses require manual assignment.

To create a subnet on the external network:

- Create the subnet:

```
$ neutron subnet-create ext-net --name ext-subnet ¥  
  --allocation-pool start=FLOATING_IP_START,end=FLOATING_IP_END ¥  
  --disable-dhcp --gateway EXTERNAL_NETWORK_GATEWAY EXTERNAL_NETWORK_CIDR
```

For example, using 203.0.113.0/24 with floating IP address range 203.0.113.101 to 203.0.113.200:

```
$ neutron subnet-create ext-net --name ext-subnet ¥  
  --allocation-pool start=203.0.113.101,end=203.0.113.200 ¥  
  --disable-dhcp --gateway 203.0.113.1 203.0.113.0/24
```

Tenant network

The tenant network provides internal network access for instances. The architecture isolates this type of network from other tenants. The demo tenant

owns this network because it only provides network access for instances within it.



注記

Perform these commands on the controller node.

To create the tenant network:

1. Source the demo tenant credentials:

```
$ source demo-openrc.sh
```

2. Create the network:

```
$ neutron net-create demo-net
```

Like the external network, your tenant network also requires a subnet attached to it. You can specify any valid subnet because the architecture isolates tenant networks. Replace `TENANT_NETWORK_CIDR` with the subnet you want to associate with the tenant network. Replace `TENANT_NETWORK_GATEWAY` with the gateway you want to associate with this network, typically the ".1" IP address. By default, this subnet will use DHCP so your instances can obtain IP addresses.

To create a subnet on the tenant network:

- Create the subnet:

```
$ neutron subnet-create demo-net --name demo-subnet ¥  
--gateway TENANT_NETWORK_GATEWAY TENANT_NETWORK_CIDR
```

Example using 192.168.1.0/24:

```
$ neutron subnet-create demo-net --name demo-subnet ¥  
--gateway 192.168.1.1 192.168.1.0/24
```

A virtual router passes network traffic between two or more virtual networks. Each router requires one or more interfaces and/or gateways that provide access to specific networks. In this case, you will create a router and attach your tenant and external networks to it.

To create a router on the tenant network and attach the external and tenant networks to it:

1. Create the router:

```
$ neutron router-create demo-router
```

2. Attach the router to the demo tenant subnet:

```
$ neutron router-interface-add demo-router demo-subnet
```

3. Attach the router to the external network by setting it as the gateway:

```
$ neutron router-gateway-set demo-router ext-net
```

Verify connectivity

We recommend that you verify network connectivity and resolve any issues before proceeding further. Following the external network subnet example using 203.0.113.0/24, the tenant router gateway should occupy the lowest IP address in the floating IP address range, 203.0.113.101. If you configured your external physical network and virtual networks correctly, you should be able to ping this IP address from any host on your external physical network.



注記

If you are building your OpenStack nodes as virtual machines, you must configure the hypervisor to permit promiscuous mode on the external network.

To verify network connectivity:

- Ping the tenant router gateway:

```
$ ping -c 4 203.0.113.101
PING 203.0.113.101 (203.0.113.101) 56(84) bytes of data.
64 bytes from 203.0.113.101: icmp_req=1 ttl=64 time=0.619 ms
64 bytes from 203.0.113.101: icmp_req=2 ttl=64 time=0.189 ms
64 bytes from 203.0.113.101: icmp_req=3 ttl=64 time=0.165 ms
64 bytes from 203.0.113.101: icmp_req=4 ttl=64 time=0.216 ms

--- 203.0.113.101 ping statistics ---
4 packets transmitted, 4 received, 0% packet loss, time 2999ms
rtt min/avg/max/mdev = 0.165/0.297/0.619/0.187 ms
```

Legacy networking

コンピュータノードの設定

Legacy networking only involves compute nodes. This section covers deployment of a simple flat network that provides IP addresses to your instances via DHCP. If your environment includes multiple compute nodes, the multi-host feature provides redundancy by spreading network functions across compute nodes.

To use legacy networking:

1. パッケージをインストールします。

```
# apt-get install nova-network nova-api-metadata
```

2. Edit the `/etc/nova/nova.conf` file:

- Add the following keys under the `[DEFAULT]` section:

```
[DEFAULT]
...
network_manager = nova.network.manager.FlatDHCPManager
firewall_driver = nova.virt.libvirt.firewall.IptablesFirewallDriver
network_size = 254
allow_same_net_traffic = False
multi_host = True
send_arp_for_ha = True
share_dhcp_address = True
force_dhcp_release = True
flat_network_bridge = br100
flat_interface = eth1
public_interface = eth1
```

3. Restart the services:

```
# service nova-network restart
# service nova-api-metadata restart
```

Create initial network

Before launching your first instance, you must create the necessary virtual network infrastructure to which the instance will connect. This network typically provides internet access from instances. You can enable internet access to individual instances using a floating IP address and suitable security group rules. The admin tenant owns this network because it provides external network access for multiple tenants.

This network shares the same subnet associated with the physical network connected to the external interface on the compute node. You should specify an exclusive slice of this subnet to prevent interference with other devices on the external network.



注記

Perform these commands on the controller node.

To create the network:

1. Source the admin tenant credentials:

```
$ source admin-openrc.sh
```

2. Create the network:

Replace NETWORK_CIDR with the subnet associated with the physical network.

```
$ nova network-create demo-net --bridge br100 --multi-host T ¥
--fixed-range-v4 NETWORK_CIDR
```

For example, using an exclusive slice of 203.0.113.0/24 with IP address range 203.0.113.24 to 203.0.113.32:

```
$ nova network-create demo-net --bridge br100 --multi-host T ¥
--fixed-range-v4 203.0.113.24/29
```


第8章 Dashboard の追加

目次

システム要件	64
Dashboard のインストール	65
Dashboard 用セッションストレージのセットアップ	66

OpenStack Dashboard は [Horizon](#) としても知られ、クラウド管理者やユーザーがさまざまな OpenStack のリソースとサービスを管理できるようになるウェブインターフェースです。

Dashboard は OpenStack API を経由して OpenStack Compute クラウドコントローラーとウェブベースで操作できます。

ここからの説明は Apache ウェブサーバーを用いて設定する導入例を示します。

[Dashboard のインストールと設定](#)をした後、以下の作業を完了できます。

- Dashboard のカスタマイズ。[OpenStack クラウド管理者ガイドの Dashboard のカスタマイズ](#)セクション参照。
- Dashboard 用セッションストレージのセットアップ。「[Dashboard 用セッションストレージのセットアップ](#)」[\[66\]](#) 参照。

システム要件

OpenStack Dashboard をインストールする前に、以下のシステム要件を満たしている必要があります。

- OpenStack Compute のインストール。ユーザーとプロジェクトの管理用の Identity Service の有効化。

Identity Service と Compute のエンドポイントの URL を記録します。

- sudo 権限を持つ Identity Service のユーザー。Apache は root ユーザーのコンテンツを処理しないため、ユーザーは sudo 権限を持つ Identity Service のユーザーとしてダッシュボードを実行する必要があります。
- Python 2.6 または 2.7。Python が Django をサポートするバージョンである必要があります。この Python のバージョンは Mac OS X を含め、あらゆるシステムで実行すべきです。インストールの前提条件はプラットフォームにより異なるかもしれません。

そして、Identity Service と通信できるノードに Dashboard をインストールし、設定します。

ユーザーのローカルマシンからウェブブラウザ経由で Dashboard にアクセスできるよう、以下の情報をユーザーに提供します。

- Dashboard にアクセスできるパブリック IP アドレス。
- Dashboard にアクセスできるユーザー名とパスワード。

お使いのウェブブラウザが HTML5 をサポートし、クッキーと JavaScript を有効化されている必要があります。



注記

Dashboard で VNC クライアントを使用する場合、ブラウザが HTML5 Canvas と HTML5 WebSockets をサポートする必要があります。

noVNC をサポートするブラウザの詳細はそれぞれ <https://github.com/kanaka/noVNC/blob/master/README.md> と <https://github.com/kanaka/noVNC/wiki/Browser-support> を参照してください。

Dashboard のインストール

Dashboard をインストールし、設定する前に 「システム要件」 [64] にある要件を満たしている必要があります。



注記

object Storage と Identity Service のみをインストールしたとき、Dashboard をインストールしても、プロジェクトが表示されず、使用することもできません。

Dashboard の導入方法の詳細は [deployment topics in the developer documentation](#) を参照してください。

1. Identity Service と通信できるノードに root として Dashboard をインストールします。

```
# apt-get install memcached libapache2-mod-wsgi openstack-dashboard
```



Ubuntu ユーザー向け注記

Remove the openstack-dashboard-ubuntu-theme package. This theme prevents translations, several menus as well as the network map from rendering correctly:

```
# apt-get remove --purge openstack-dashboard-ubuntu-theme
```

2. /etc/memcached.conf に設定したものと一致されるために、/etc/openstack-dashboard/local_settings.py の CACHES['default']['LOCATION'] の値を変更します。

/etc/openstack-dashboard/local_settings.py を開き、この行を探します。

```
CACHES = {
    'default': {
        'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
        'LOCATION': '127.0.0.1:11211'
    }
}
```



注

- アドレスとポートは `/etc/memcached.conf` に設定したものと一致する必要があります。

memcached 設定を変更する場合、変更を反映するために Apache ウェブサーバーを再起動する必要があります。

- セッションストレージのために memcached 以外のオプションを使用することもできます。SESSION_ENGINE オプションによりセッションバックエンドを設定します。
- タイムゾーンを変更する場合、ダッシュボードを使用します。または `/etc/openstack-dashboard/local_settings.py` ファイルを編集します。

次のパラメーターを変更します。 `TIME_ZONE = "UTC"`

3. Dashboard にアクセスしたいアドレスを含めるために `local_settings.py` の `ALLOWED_HOSTS` を更新します。

```
filename os="centos;fedora;rhel">/etc/openstack-dashboard/local_settings
ALLOWED_HOSTS = ['localhost', 'my-desktop']
```

4. このガイドはコントローラーノードで Dashboard を実行していると仮定します。 `local_settings.py` の設定を適切に変更することにより、別のサーバーで Dashboard を簡単に実行できます。

`/etc/openstack-dashboard/local_settings.py` を編集し、`OPENSTACK_HOST` を Identity Service のホスト名に変更します。

```
OPENSTACK_HOST = "controller"
```

5. Apache ウェブサーバーと memcached を起動します。

```
# service apache2 restart
# service memcached restart
```

6. これで Dashboard に `http://controller/horizon` からアクセスできます。

OpenStack Identity Service で作成したどれかのユーザーのクレデンシャルでログインします。

Dashboard 用セッションストレージのセットアップ

Dashboard はユーザーのセッションデータを処理するために [Django セッションフレームワーク](#) を使用します。しかしながら、あらゆる利用可能なセッションバックエンドを使用できます。 `local_settings` ファイル (Fedora/RHEL/CentOS の場合: `/etc/openstack-dashboard/local_settings`、Ubuntu/Debian の場合: `/etc/openstack-`

dashboard/local_settings.py、openSUSE の場合: /srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py) にある SESSION_ENGINE 設定によりセッションバックエンドをカスタマイズします。

以下のセクションは、Dashboard の導入に関する各選択肢の賛否について記載します。

ローカルメモリキャッシュ

ローカルメモリストレージは、外部にまったく何も依存しないため、セットアップすることが最速かつ容易なバックエンドです。以下の重大な弱点があります。

- プロセスやワーカーをまたがる共有ストレージがありません。
- プロセス終了後の永続性がありません。

ローカルメモリバックエンドは、依存関係がないため、Horizon 単体のデフォルトとして有効化されています。本番環境や深刻な開発作業の用途に推奨しません。以下のように有効化します。

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    'BACKEND': 'django.core.cache.backends.locmem.LocMemCache'
}
```

キーバリューストア

外部キャッシュのために Memcached や Redis のようなアプリケーションを使用できます。これらのアプリケーションは永続性と共有ストレージを提供します。小規模な環境や開発環境に有用です。

Memcached

Memcached is a high-performance and distributed memory object caching system providing in-memory key-value store for small chunks of arbitrary data.

要件:

- Memcached サービスが実行中であり、アクセス可能であること。
- Python モジュール python-memcached がインストールされていること。

以下のように有効化します。

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache'
    'LOCATION': 'my_memcached_host:11211',
}
```

Redis

Redis はオープンソースで BSD ライセンスの高度なキーバリューストアです。しばしばデータ構造サーバーとして参照されます。

要件:

- Redis サービスが実行中であり、アクセス可能であること。

- Python モジュール `redis` と `django-redis` がインストールされていること。

以下のように有効化します。

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    "default": {
        "BACKEND": "redis_cache.cache.RedisCache",
        "LOCATION": "127.0.0.1:6379:1",
        "OPTIONS": {
            "CLIENT_CLASS": "redis_cache.client.DefaultClient",
        }
    }
}
```

データベースの初期化と設定

データベースのセッションバックエンドはスケラブルかつ永続的です。高い多重度と高可用性を実現できます。

しかしながら、データベースのセッションバックエンドは、より低速なセッションストレージの一つであり、高負荷環境で大きなオーバーヘッドを引き起こします。データベース環境の適切な設定は大きな仕事であり、このドキュメントの範囲を越えています。

1. `mysql` コマンドラインクライアントを実行します。

```
$ mysql -u root -p
```

2. プロンプトが表示されたら、MySQL の `root` ユーザのパスワードを入力します。
3. MySQL データベースを設定するために、`dash` データベースを作成します。

```
mysql> CREATE DATABASE dash;
```

4. 新しく作成した `dash` データベース用の MySQL ユーザーを作成し、データベースのフルアクセスを許可します。`DASH_DBPASS` を新しいユーザー用のパスワードで置き換えます。

```
mysql> GRANT ALL ON dash.* TO 'dash'@'%' IDENTIFIED BY 'DASH_DBPASS';
mysql> GRANT ALL ON dash.* TO 'dash'@'localhost' IDENTIFIED BY 'DASH_DBPASS';
```

5. `mysql>` プロンプトで `quit` と入力し、MySQL から抜けます。
6. `local_settings` ファイル (Fedora/RHEL/CentOS の場合: `/etc/openstack-dashboard/local_settings`、Ubuntu/Debian の場合: `/etc/openstack-dashboard/local_settings.py`、openSUSE の場合: `/srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py`) で、これらのオプションを変更します。

```
SESSION_ENGINE = 'django.core.cache.backends.db.DatabaseCache'
DATABASES = {
    'default': {
        # Database configuration here
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dash',
        'USER': 'dash',
        'PASSWORD': 'DASH_DBPASS',
        'HOST': 'localhost',
        'default-character-set': 'utf8'
    }
}
```

7. After configuring the local_settings as shown, you can run the manage.py syncdb command to populate this newly-created database.

```
$ /usr/share/openstack-dashboard/manage.py syncdb
```

openSUSE ではパスが /srv/www/openstack-dashboard/manage.py であることに注意してください。

結果として、以下の出力が返されます。

```
Installing custom SQL ...
Installing indexes ...
DEBUG:django.db.backends:(0.008) CREATE INDEX `django_session_c25c2c28` ON
`django_session` (`expire_date`);; args=()
No fixtures found.
```

8. Ubuntu の場合: apache2 を再起動するときに、警告を避けたい場合、以下のようにダッシュボードのディレクトリにブラックホールディレクトリを作成します。

```
# mkdir -p /var/lib/dash/.blackhole
```

9. デフォルトのサイトとシンボリックの設定を取得するために Apache を再起動します。

On Ubuntu:

```
# /etc/init.d/apache2 restart
```

On Fedora/RHEL/CentOS:

```
# service httpd restart
```

```
# service apache2 restart
```

On openSUSE:

```
# systemctl restart apache2.service
```

10. Ubuntu の場合、API サーバーがエラーなくダッシュボードに接続できることを確実にするために nova-api サービスを再起動します。

```
# service nova-api restart
```

キャッシュ付きデータベース

To mitigate the performance issues of database queries, you can use the Django cached_db session back end, which utilizes both your database and caching infrastructure to perform write-through caching and efficient retrieval.

前に説明したように、データベースとキャッシュの両方を設定することにより、このハイブリッド設定を有効化します。そして、以下の値を設定します。

```
SESSION_ENGINE = "django.contrib.sessions.backends.cached_db"
```

クッキー

If you use Django 1.4 or later, the signed_cookies back end avoids server load and scaling problems.

このバックエンドは、ユーザーのブラウザーにより保存されるクッキーにセッションデータを保存します。バックエンドは、セッションデータが転送中に改ざんされていないことを保証するために、暗号的な署名技術を使用します。これは暗号化とは違います。セッションデータは攻撃者により読み取りできます。

このエンジンのいいところは、追加の依存関係や環境のオーバーヘッドが必要ないことです。また、保存されるセッションデータの量が通常のクッキーに収まる限り、どこまでもスケールします。

最大の欠点は、ユーザーのマシンのストレージにセッションデータを保存し、ネットワーク経由で送信されることです。また、保存できるセッションデータの量に限りがあります。

Django [cookie-based sessions](#) ドキュメントを参照してください。

第9章 Add the Block Storage service

目次

Block Storage	71
Configure a Block Storage service controller	71
Configure a Block Storage service node	73

The OpenStack Block Storage service works through the interaction of a series of daemon processes named `cinder-*` that reside persistently on the host machine or machines. You can run the binaries from a single node or across multiple nodes. You can also run them on the same node as other OpenStack services. The following sections introduce Block Storage service components and concepts and show you how to configure and install the Block Storage service.

Block Storage

The Block Storage service enables management of volumes, volume snapshots, and volume types. It includes the following components:

- `cinder-api`: Accepts API requests and routes them to `cinder-volume` for action.
- `cinder-volume`: Responds to requests to read from and write to the Block Storage database to maintain state, interacting with other processes (like `cinder-scheduler`) through a message queue and directly upon block storage providing hardware or software. It can interact with a variety of storage providers through a driver architecture.
- `cinder-scheduler` daemon: Like the `nova-scheduler`, picks the optimal block storage provider node on which to create the volume.
- Messaging queue: Routes information between the Block Storage service processes.

The Block Storage service interacts with Compute to provide volumes for instances.

Configure a Block Storage service controller



注記

このセクションは、コントローラーノードで OpenStack Block Storage Service を設定する方法を説明します。2 番目のノードが `cinder-volume` サービス経由でストレージを提供すると仮定します。2 番目のノードを設定する方法の説明は [「Configure a Block Storage service node」 \[73\]](#) を参照してください。

さまざまなストレージシステムを使用するよう OpenStack を設定できます。このガイドの例は LVM を設定する方法を示します。

1. Install the appropriate packages for the Block Storage service:

```
# apt-get install cinder-api cinder-scheduler
```

2. Block Storage が MySQL データベースを使用するように設定します。/etc/cinder/cinder.conf ファイルを編集し、以下のキーを [database] セクションに追加します。後の手順で作成する Block Storage データベース用のパスワードで CINDER_DBPASS を置き換えます。



注記

いくつかのディストリビューションでパッケージ化された /etc/cinder/cinder.conf ファイルは、[database] セクションヘッダーが含まれていません。ここから進む前に、このセクションヘッダーをファイルの最後に追加する必要があります。

```
[database]
...
connection = mysql://cinder:CINDER_DBPASS@controller/cinder
```

3. cinder データベースを作成するために、root としてログインするために設定したパスワードを使用します。

```
$ mysql -u root -p
mysql> CREATE DATABASE cinder;
mysql> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'localhost' ¥
IDENTIFIED BY 'CINDER_DBPASS';
mysql> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%' ¥
IDENTIFIED BY 'CINDER_DBPASS';
```

4. Create the database tables for the Block Storage service.

```
# cinder-manage db sync
```

5. Create a cinder user. The Block Storage service uses this user to authenticate with the Identity service. Use the service tenant and give the user the admin role.

```
$ keystone user-create --name=cinder --pass=CINDER_PASS --email=cinder@example.com
$ keystone user-role-add --user=cinder --tenant=service --role=admin
```

6. /etc/cinder/api-paste.ini ファイルにクレデンシャルを追加します。テキストエディターでファイルを開き、[filter:authtoken] セクションを探します。以下のオプションを設定します。

```
[filter:authtoken]
paste.filter_factory=keystoneclient.middleware.auth_token:filter_factory
auth_host=controller
auth_port = 35357
auth_protocol = http
auth_uri = http://controller:5000
admin_tenant_name=service
admin_user=cinder
admin_password=CINDER_PASS
```

7. /etc/cinder/cinder.conf ファイルの [DEFAULT] 設定グループにこれらの設定キーを設定することにより、Block Service が RabbitMQ メッセージブローカーを使用す

るよう設定します。RABBIT_PASS を RabbitMQ 用に選択したパスワードに置き換えます。

```
[DEFAULT]
...
rpc_backend = cinder.openstack.common.rpc.impl_kombu
rabbit_host = controller
rabbit_port = 5672
rabbit_userid = guest
rabbit_password = RABBIT_PASS
```

8. Register the Block Storage service with the Identity Service so that other OpenStack services can locate it. Register the service and specify the endpoint using the keystone command.

```
$ keystone service-create --name=cinder --type=volume ¥
--description="OpenStack Block Storage"
$ keystone endpoint-create ¥
--service-id=$(keystone service-list | awk '/ volume / {print $2}') ¥
--publicurl=http://controller:8776/v1/%(tenant_id)s ¥
--internalurl=http://controller:8776/v1/%(tenant_id)s ¥
--adminurl=http://controller:8776/v1/%(tenant_id)s
```

9. Also register a service and endpoint for version 2 of the Block Storage service API.

```
# keystone service-create --name=cinderv2 --type=volumev2 ¥
--description="OpenStack Block Storage v2"
$ keystone endpoint-create ¥
--service-id=$(keystone service-list | awk '/ volumev2 / {print $2}') ¥
--publicurl=http://controller:8776/v2/%(tenant_id)s ¥
--internalurl=http://controller:8776/v2/%(tenant_id)s ¥
--adminurl=http://controller:8776/v2/%(tenant_id)s
```

10. 新しい設定を用いて Cinder サービスを再起動します。

```
# service cinder-scheduler restart
# service cinder-api restart
```

Configure a Block Storage service node

After you configure the services on the controller node, configure a second system to be a Block Storage service node. This node contains the disk that serves volumes.

さまざまなストレージシステムを使用するよう OpenStack を設定できます。このガイドの例は LVM を設定する方法を示します。

1. システムを設定するために [2章オペレーティングシステムの基本設定 \[7\]](#) にある方法を使用します。以下の項目はコントローラーノードのインストール説明と異なることに注意してください。
 - ホスト名を block1 に設定します。両方のノードの IP アドレスとホスト名が各システムの /etc/hosts にあることを確認します。
 - コントローラーノードから同期するために、「[Network Time Protocol \(NTP\)](#)」 [\[9\]](#) にある説明に従います。

2. Install the required LVM packages, if they are not already installed:

```
# apt-get install lvm2
```

3. LVM の物理ボリュームと論理ボリュームを作成します。このガイドはこの目的のために使用される 2 番目のディスク /dev/sdb を仮定します。

```
# pvcreate /dev/sdb
# vgcreate cinder-volumes /dev/sdb
```

4. 仮想マシンにより使用されるデバイスをスキャンすることから LVM を保護するために、/etc/lvm/lvm.conf のデバイスのセクションにフィルター項目を追加します。



注記

You must add required physical volumes for LVM on the Block Storage host. Run the `pvdisplay` command to get a list of required volumes.

フィルター配列にある各項目は、許可するために `a` から、拒否するために `r` から始まります。Block Storage のホストで必要となる物理ボリュームは `a` から始まる名前を持つ必要があります。配列は一覧に無いすべてのデバイスを拒否するために `"r/*/"` で終わる必要があります。

この例では、/dev/sda1 がノードのオペレーティングシステム用のボリュームが置かれるボリュームです。/dev/sdb は cinder-volumes のために予約されたボリュームです。

```
devices {
...
filter = [ "a/sda1/", "a/sdb/", "r/*/" ]
...
}
```

5. After you configure the operating system, install the appropriate packages for the Block Storage service:

```
# apt-get install cinder-volume
```

6. コントローラーから /etc/cinder/api-paste.ini ファイルをコピーします。または、テキストエディターで開き、[filter:authtoken] セクションを見つけます。以下のオプションが設定されていることを確認します。

```
[filter:authtoken]
paste.filter_factory=keystoneclient.middleware.auth_token:filter_factory
auth_host=controller
auth_port = 35357
auth_protocol = http
admin_tenant_name=service
admin_user=cinder
admin_password=CINDER_PASS
```

7. /etc/cinder/cinder.conf ファイルの [DEFAULT] 設定グループにこれらの設定キーを設定することにより、Block Service が RabbitMQ メッセージブローカーを使用するように設定します。RABBIT_PASS を RabbitMQ 用に選択したパスワードに置き換えます。

```
[DEFAULT]
...
rpc_backend = cinder.openstack.common.rpc.impl_kombu
rabbit_host = controller
rabbit_port = 5672
rabbit_userid = guest
rabbit_password = RABBIT_PASS
```

8. Block Storage が MySQL データベースを使用するように設定します。/etc/cinder/cinder.conf ファイルを編集し、以下のキーを [database] セクションに追加します。Block Storage データベース用に選択したパスワードで CINDER_DBPASS を置き換えます。



注記

いくつかのディストリビューションでパッケージ化された /etc/cinder/cinder.conf ファイルは、[database] セクションヘッダーが含まれていません。ここから進む前に、このセクションヘッダーをファイルの最後に追加する必要があります。

```
[database]
...
connection = mysql://cinder:CINDER_DBPASS@controller/cinder
```

9. 新しい設定を用いて Cinder サービスを再起動します。

```
# service cinder-volume restart
# service tgt restart
```

第10章 Object Storage の追加

目次

Object Storage Service	76
システム要件	77
Object Storage 用ネットワークの計画	78
Object Storage インストールアーキテクチャー例	78
Object Storage のインストール	79
ストレージノードのインストールと設定	81
プロキシノードのインストールと設定	83
ストレージノードでのサービスの起動	85
Object Storage のインストール後作業	86

OpenStack Object Storage Service はオブジェクトストレージと REST API 経由の取得を提供するために一緒に動作します。このアーキテクチャー例は、Keystone として知られる Identity Service がすでにインストールされている必要があります。

Object Storage Service

Object Storage Service は高いスケーラビリティを持つ、永続的なマルチテナントのオブジェクトストレージシステムです。RESTful HTTP API 経由で利用する低コストで大規模な非構造データに向いています。

以下のコンポーネントを含みます。

- プロキシサーバー (swift-proxy-server)。ファイルのアップロード、メタデータの変更、コンテナの作成をするために、Object Storage API と生の HTTP リクエストを受け付けます。ウェブブラウザにファイルやコンテナを一覧表示します。パフォーマンスを改善するために、プロキシサーバーはオプションとしてキャッシュを使用できます。通常は memcache を用います。
- アカウントサーバー (swift-account-server)。Object Storage Service で定義されたアカウントを管理します。
- コンテナサーバー (swift-container-server)。Object Storage Service の中で、コンテナやフォルダーの対応付けを管理します。
- オブジェクトサーバー (swift-object-server)。ストレージノードでファイルのような実際のオブジェクトを管理します。
- いくつかの定期的なプロセス。大規模なデータストアでハウスキーピング作業を実行します。複製サービスにより、クラスター全体で一貫性と可用性が確保されます。他の定期的なプロセスにオーディター、アップデーター、リーパーなどがあります。
- 認証を処理する、設定可能な WSGI ミドルウェア。通常は Identity Service。

システム要件

ハードウェア: OpenStack Object Storage は一般的なハードウェアで実行するために設計されています。



注記

Object Storage と Identity Service のみをインストールするとき、Compute と Image Service もインストールしなければ、ダッシュボードを使用できません。

表10.1 ハードウェア推奨事項

Server	推奨ハードウェア	注
Object Storage オブジェクトサーバー	プロセッサ: 4 コア 2 個 メモリ: 8 ~ 12 GB RAM ディスク容量: 容量単価に最適なもの ネットワーク: 1 GB NIC 1 個	ディスクの合計容量はどのくらいラック効率良く収容できるかに依存します。エンタープライズ向けの一般的な故障率を達成しながら、GB 単価に最適なものにしたいです。Rackspace の場合、ストレージサーバーは現在、24 本の 2TB SATA ディスクと 8 コアのプロセッサを持つごく一般的な 4U サーバーを実行しています。ストレージディスクの RAID は必要ではなく、推奨しません。Swift のディスク利用パターンは RAID に対して考えられる最悪のケースです。RAID 5 や 6 を使用すると、パフォーマンスが非常にすぐに劣化します。 例として、Rackspace は 24 本の 2TB SATA ディスクと 8 コアのプロセッサを持つ Cloud Files ストレージサーバーを稼働しています。多くのサービスは、設定でワーカーと多重度をサポートします。これにより、サービスが利用可能なコアを効率的に使用できます。
Object Storage コンテナ/アカウントサーバー	プロセッサ: 4 コア 2 個 メモリ: 8 ~ 12 GB RAM ネットワーク: 1 GB NIC 1 個	SQLite データベースと関わるため IOPS に最適化します。
Object Storage プロキシサーバー	プロセッサ: 4 コア 2 個 ネットワーク: 1 GB NIC 1 個	より高いネットワークスループットにより、多くの API リクエストをサポートするためのより良いパフォーマンスを提供します。 最高の CPU パフォーマンスのためにプロキシサーバーを最適化します。プロキシサービスはより多くの CPU 処理とネットワーク I/O 集中が発生します。10 ギガネットワークをプロキシに使用している場合、または SSL 通信をプロキシで終了している場合、さらに多くの CPU パワーが必要になります。

オペレーティングシステム: OpenStack Object Storage は現在 Ubuntu、RHEL、CentOS、Fedora、openSUSE、SLES で動作します。

ネットワーク: 内部的に 1Gbps か 10 Gbps が推奨されます。OpenStack Object Storage の場合には、外部ネットワークが外部とプロキシサーバーを接続すべきです。また、ストレージネットワークがプライベートネットワークで分離されていることを意図しています。

データベース: OpenStack Object Storage の場合には、SQLite データベースが OpenStack Object Storage のコンテナとアカウントの管理プロセスの一部です。

権限: OpenStack Object Storage を root としてインストールできます。または、すべての権限を有効化するように sudoers ファイルを設定する場合、sudo 権限を持つユーザーとしてインストールできます。

Object Storage 用ネットワークの計画

ネットワークリソースの節約のため、およびネットワーク管理者が必要に応じて API とストレージのネットワークへのアクセスを提供するためのネットワークとパブリック IP アドレスの必要性について確実に理解するために、このセクションは推奨量と必須の最小量を提供します。少なくとも 1000 Mbps のスループットが推奨されます。

このガイドは以下のネットワークを記載します。

- 必須のパブリックネットワーク。プロキシサーバーに接続します。
- 必須のストレージネットワーク。クラスターの外部からアクセスできません。すべてのノードがこのネットワークに接続されます。
- オプションの複製ネットワーク。クラスターの外部からアクセスできません。ストレージノード間の複製通信専用です。リングで設定される必要があります。

すべての OpenStack Object Storage サービスとストレージノードの rsync デーモンは、デフォルトで STORAGE_LOCAL_NET IP アドレスをリッスンするように設定されます。

リングで複製ネットワークを設定する場合、アカウントサーバー、コンテナサーバー、オブジェクトサーバーが STORAGE_LOCAL_NET と STORAGE_REPLICATION_NET の IP アドレスをリッスンします。rsync デーモンは STORAGE_REPLICATION_NET IP アドレスのみをリッスンします。

パブリックネットワーク (パブリックにルーティング可能な IP 範囲)

クラウドインフラストラクチャーの中で API エンドポイントにアクセス可能なパブリック IP を提供します。

最小量: 各プロキシサーバーに対して IP アドレス 1 つ。

ストレージネットワーク (RFC1918 IP 範囲、パブリックにルーティングできません)

Object Storage インフラストラクチャーの中ですべてのサーバー間通信を管理します。

最小量: 各ストレージノードとプロキシサーバーに対して IP アドレス 1 つ。

推奨量: 上のとおり、クラスターの最大量に拡張するための余地を持ちます。たとえば、255 や CIDR /24 です。

複製ネットワーク (RFC1918 IP 範囲、パブリックにルーティングできません)

Object Storage インフラストラクチャーの中でストレージサーバー間の複製関連の通信を管理します。

推奨量: STORAGE_LOCAL_NET に限ります。

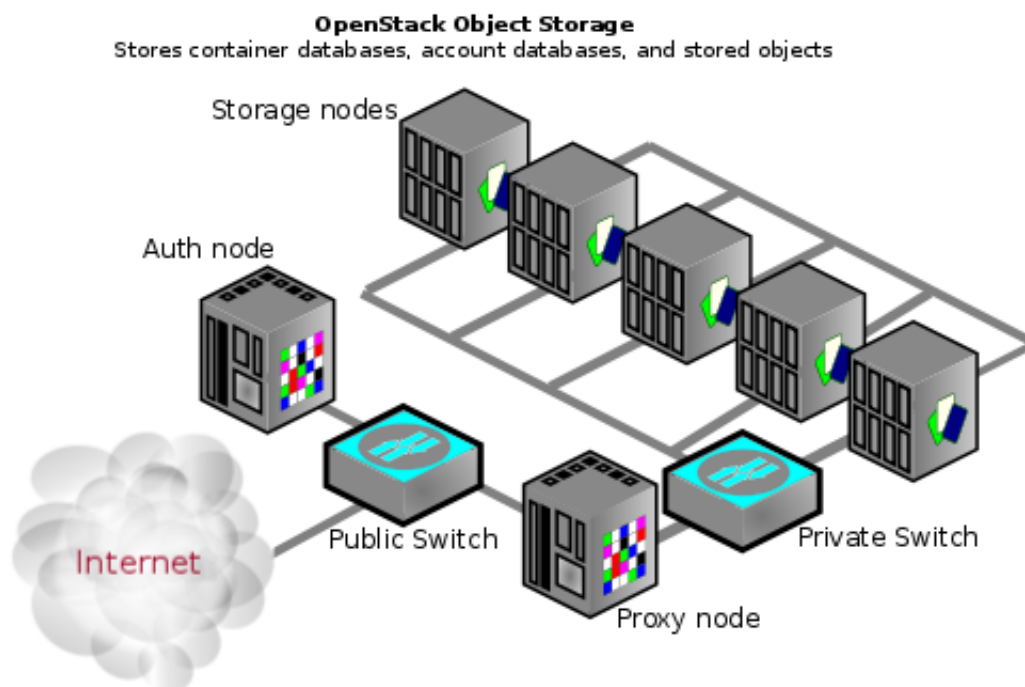
Object Storage インストールアーキテクチャー例

- ノード。1 つ以上の OpenStack Object Storage サービスを実行するホストマシン。

- プロキシノード。プロキシサービスを実行します。
- ストレージノード。アカウントサービス、コンテナサービス、オブジェクトサービスを実行します。
- リング。OpenStack Object Storage のデータと物理デバイスの一組のマッピング。
- レプリカ。オブジェクトのコピー。デフォルトで 3 つのコピーがクラスターに維持されます。
- ゾーン。独立した障害特性に関連した、クラスターの論理的な分離部分。

信頼性とパフォーマンスを向上させるために、追加のプロキシノードを追加できます。

このドキュメントは、リングで別々のゾーンとして各ストレージノードについて記載します。最低限、5 つのゾーンが推奨されます。ゾーンは他のノードから独立したノード（別々のサーバー、ネットワーク、電力、設置場所さえ）のグループです。リングはすべての複製が別々のゾーンに保存されていることを保証します。この図は最小インストールに対する設定の可能性の 1 つを示します。



Object Storage のインストール

OpenStack Object Storage を開発もしくはテスト目的で一つのサーバーにインストールすることができますが、複数のサーバーにインストールすることで、本番環境の分散オブジェクトストレージシステムに期待する高可用性と冗長性を実現できます。

開発目的でソースコードから単一ノードのインストールを実行するために、Swift All In One 手順 (Ubuntu) や DevStack (複数のディストリビューション) を使用します。手動インストールは http://swift.openstack.org/development_saio.html を参照してください。Identity Service (keystone) を用いた認証を含む、オールインワンは <http://devstack.org> を参照してください。

始める前に

新規サーバーにインストールしている場合、利用可能なオペレーティングシステムのインストールメディアを準備します。

これらの手順は [OpenStack パッケージ](#) に示されている、お使いのオペレーティングシステム用のパッケージのリポジトリをセットアップしていることを仮定します。

このドキュメントは以下の種類のノードを使用したクラスターをインストールする方法を説明しています。

- swift-proxy-server プロセスを実行する 1 台のプロキシノード。このプロキシサーバーは適切なストレージノードにリクエストを中継します。
- swift-account-server、swift-container-server、swift-object-server プロセスを実行する 5 台のストレージノード。これはアカウントデータベース、コンテナデータベース、実際のオブジェクトの保存を制御します。



注記

最初はより少ない台数のストレージノードを使用することができますが、本番環境のクラスターは少なくとも 5 台が推奨されます。

一般的なインストール手順

1. Object Storage Service が Identity Service で認証するために使用する swift ユーザーを作成します。swift ユーザー用のパスワードと電子メールアドレスを選択します。service プロジェクトを使用し、ユーザーに admin ロールを与えます。

```
$ keystone user-create --name=swift --pass=SWIFT_PASS ¥  
--email=swift@example.com  
$ keystone user-role-add --user=swift --tenant=service --role=admin
```

2. Object Storage Service のサービスエントリを作成します。

```
$ keystone service-create --name=swift --type=object-store ¥  
--description="OpenStack Object Storage"
```

Property	Value
description	OpenStack Object Storage
id	eede9296683e4b5ebfa13f5166375ef6
name	swift
type	object-store



注記

サービス ID はランダムに生成され、ここに表示されているものとは異なります。

3. 返されたサービス ID を使用することにより、Object Storage Service の API エンドポイントを指定します。エンドポイントを指定するとき、パブリック API、内部 API、管理 API の URL を指定します。このガイドでは、controller というホスト名を使用します。

```
$ keystone endpoint-create ¥
--service-id=$(keystone service-list | awk '/ object-store / {print $2}') ¥
--publicurl='http://controller:8080/v1/AUTH_$(tenant_id)s' ¥
--internalurl='http://controller:8080/v1/AUTH_$(tenant_id)s' ¥
--adminurl=http://controller:8080
```

Property	Value
adminurl	http://controller:8080/
id	9e3ce428f82b40d38922f242c095982e
internalurl	http://controller:8080/v1/AUTH_\$(tenant_id)s
publicurl	http://controller:8080/v1/AUTH_\$(tenant_id)s
region	regionOne
service_id	eede9296683e4b5ebfa13f5166375ef6

- すべてのノードに設定用ディレクトリを作成します。

```
# mkdir -p /etc/swift
```

- すべてのノードで /etc/swift/swift.conf を作成します。

```
[swift-hash]
# random unique string that can never change (DO NOT LOSE)
swift_hash_path_suffix = fLibertYgibbitZ
```



注記

/etc/swift/swift.conf のサフィックス値は、リングでマッピングを決めるためのハッシュをするときに、ソルトとして使用するために何かランダムな文字列に設定すべきです。このファイルはクラスター上のすべてのノードで同じにする必要があります。

次にストレージノードとプロキシノードをセットアップします。この例では、共通の認証部品として Identity Service を使用します。

ストレージノードのインストールと設定



注記

Object Storage は拡張属性 (XATTRS) をサポートするすべてのファイルシステムで動作します。Rackspace でかなりテストしてベンチマークしたところ、swift のユースケースの場合は XFS が最もパフォーマンスが良かったです。これは徹底的にテストされた唯一のファイルシステムでもあります。その他の推奨は [OpenStack 設定リファレンス](#) を参照してください。

- ストレージノードのパッケージをインストールします。

```
# apt-get install swift-account swift-container swift-object xfsprogs
```

- ストレージ用に使用したいノードで各デバイスに対して、XFS ボリュームをセットアップします (例として /dev/sdb が使用されます)。ドライブに単一のパーティションを使用します。たとえば、12 本のディスクを持つサーバーで、この手順で触れませんが、オペレーティングシステム用に1~2 本のディスクを使用するかもしれ

ません。他の 10~11 本のディスクは単一のパーティションを持ち、XFS でフォーマットされるべきです。

```
# fdisk /dev/sdb
# mkfs.xfs /dev/sdb1
# echo "/dev/sdb1 /srv/node/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 0" >> /etc/fstab
# mkdir -p /srv/node/sdb1
# mount /srv/node/sdb1
# chown -R swift:swift /srv/node
```

3. /etc/rsyncd.conf を作成します。

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = STORAGE_LOCAL_NET_IP

[account]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/account.lock

[container]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/container.lock

[object]
max connections = 2
path = /srv/node/
read only = false
lock file = /var/lock/object.lock
```

4. (オプション) rsync と複製の通信を複製ネットワークと分離したい場合、STORAGE_LOCAL_NET_IP の代わりに STORAGE_REPLICATION_NET_IP を設定します。

```
address = STORAGE_REPLICATION_NET_IP
```

5. /etc/default/rsync で以下の行を編集します。

```
RSYNC_ENABLE=true
```

6. rsync サービスを起動します。

```
# service rsync start
```



注記

rsync サービスは認証を必要としないため、ローカルのプライベートネットワークで実行します。

7. swift recon キャッシュディレクトリを作成し、そのパーミッションを設定します。

```
# mkdir -p /var/swift/recon
```

```
# chown -R swift:swift /var/swift/recon
```

プロキシノードのインストールと設定

プロキシサーバーは各リクエストを受け取り、アカウント、コンテナ、オブジェクトの位置を検索し、リクエストを正しくルーティングします。プロキシサーバーは API リクエストも処理します。/etc/swift/proxy-server.conf ファイルでアカウント管理を設定することにより有効化できます。



注記

The Object Storage processes run under a separate user and group, set by configuration options, and referred to as swift:swift. The default user is swift.

1. swift-proxy サービスをインストールします。

```
# apt-get install swift-proxy memcached python-keystoneclient python-swiftclient python-webob
```

2. memcached が標準のインターフェースでローカルの非パブリックなネットワークをリスンするよう変更します。/etc/memcached.conf ファイルのこの行を編集します。

```
-l 127.0.0.1
```

これを次のように変更します。

```
-l PROXY_LOCAL_NET_IP
```

3. memcached サービスを再起動します。

```
# service memcached restart
```

4. /etc/swift/proxy-server.conf を作成します。

```
[DEFAULT]
bind_port = 8080
user = swift

[pipeline:main]
pipeline = healthcheck cache authtoken keystoneauth proxy-server

[app:proxy-server]
use = egg:swift#proxy
allow_account_management = true
account_autocreate = true

[filter:keystoneauth]
use = egg:swift#keystoneauth
operator_roles = Member,admin,swiftoperator

[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory

# Delaying the auth decision is required to support token-less
# usage for anonymous referrers ('.r:*').
delay_auth_decision = true
```

```
# cache directory for signing certificate
signing_dir = /home/swift/keystone-signing

# auth_* settings refer to the Keystone server
auth_protocol = http
auth_host = controller
auth_port = 35357

# the service tenant and swift username and password created in Keystone
admin_tenant_name = service
admin_user = swift
admin_password = SWIFT_PASS

[filter:cache]
use = egg:swift#memcache

[filter:catch_errors]
use = egg:swift#catch_errors

[filter:healthcheck]
use = egg:swift#healthcheck
```



注記

複数の memcache サーバーを実行している場合、`/etc/swift/proxy-server.conf` ファイルの `[filter:cache]` セクションで複数の IP:port の一覧を置きます。

```
10.1.2.3:11211,10.1.2.4:11211
```

プロキシサーバーのみが memcache を使用します。

5. アカウント、コンテナ、オブジェクトリングを作成します。builder コマンドがいくつかのパラメーターを用いてビルダーファイルを作成します。18 という値を持つパラメーターは、パーティションの大きさが 2 の 18 乗となるを意味します。この “partition power” (パーティションのべき乗) の値は、リング全体が使用したストレージの合計量に依存します。3 という値は各オブジェクトの複製数を表します。最後の値は一度ならずパーティションが移動することを制限する時間数です。

```
# cd /etc/swift
# swift-ring-builder account.builder create 18 3 1
# swift-ring-builder container.builder create 18 3 1
# swift-ring-builder object.builder create 18 3 1
```

6. 各ノードですべてのストレージデバイスに対して、各リングに項目を追加します。

```
# swift-ring-builder account.builder add
zZONE-STORAGE_LOCAL_NET_IP:6002[RSTORAGE_REPLICATION_NET_IP:6005]/DEVICE 100
# swift-ring-builder container.builder add
zZONE-STORAGE_LOCAL_NET_IP_1:6001[RSTORAGE_REPLICATION_NET_IP:6004]/DEVICE 100
# swift-ring-builder object.builder add
zZONE-STORAGE_LOCAL_NET_IP_1:6000[RSTORAGE_REPLICATION_NET_IP:6003]/DEVICE 100
```



注記

複製のために専用のネットワークを使用したくなれば、オプションの `STORAGE_REPLICATION_NET_IP` パラメーターを省略する必要があります。

たとえば、ストレージノードが IP 10.0.0.1 でゾーン 1 にパーティションを持つならば、ストレージノードは複製ネットワークのアドレス 10.0.1.1 を持ちます。このパーティションのマウントポイントは /srv/node/sdb1 です。/etc/rsyncd.conf のパスは /srv/node/ です。DEVICE が sdb1 になり、コマンドは次のとおりです。

```
# swift-ring-builder account.builder add z1-10.0.0.1:6002R10.0.1.1:6005/sdb1 100
# swift-ring-builder container.builder add z1-10.0.0.1:6001R10.0.1.1:6005/sdb1 100
# swift-ring-builder object.builder add z1-10.0.0.1:6000R10.0.1.1:6005/sdb1 100
```



注記

各ゾーンに対して 1 つのノードを持つ 5 つのゾーンを仮定する場合、ZONE を 1 から始めます。それぞれの追加ノードに対して、ZONE を 1 増やします。

7. 各リングのリングコンテンツを検証します。

```
# swift-ring-builder account.builder
# swift-ring-builder container.builder
# swift-ring-builder object.builder
```

8. リングを再バランスします。

```
# swift-ring-builder account.builder rebalance
# swift-ring-builder container.builder rebalance
# swift-ring-builder object.builder rebalance
```



注記

リングの再バランスには少し時間がかかります。

9. account.ring.gz、container.ring.gz、object.ring.gz ファイルをそれぞれのプロキシノードとストレージノードの /etc/swift にコピーします。
10. swift ユーザーがすべての設定ファイルを所有していることを確認します。

```
# chown -R swift:swift /etc/swift
```

11. プロキシサービスを再起動します。

```
# service swift-proxy restart
```

ストレージノードでのサービスの起動

これで、リングファイルが各ストレージノードに存在するので、サービスを起動できます。各ストレージノードで以下のコマンドを実行します。

```
# for service in ¥
swift-object swift-object-replicator swift-object-updater swift-object-auditor ¥
swift-container swift-container-replicator swift-container-updater swift-container-auditor ¥
swift-account swift-account-replicator swift-account-reaper swift-account-auditor; do ¥
    service $service start; done
```



注記

すべての Swift サービスを起動するために、次のコマンドを実行します。

```
# swift-init all start
```

swift-init コマンドについて詳しく知りたい場合、以下を実行します。

```
$ man swift-init
```

Object Storage のインストール後作業

インストールの検証

プロキシサーバー、または Identity Service にアクセスできるすべてのサーバーから、これらのコマンドを実行できます。

1. クレデンシャルが openrc.sh ファイルに正しくセットアップされていることを確認します。このファイルを以下のように読み込みます。

```
$ source openrc.sh
```

2. Run the following swift command:

```
$ swift stat
Account: AUTH_11b9758b7049476d9b48f7a91ea11493
Containers: 0
  Objects: 0
  Bytes: 0
Content-Type: text/plain; charset=utf-8
X-Timestamp: 1381434243.83760
X-Trans-Id: txdcdd594565214fb4a2d33-0052570383
X-Put-Timestamp: 1381434243.83760
```

3. Run the following swift commands to upload files to a container. Create the test.txt and test2.txt test files locally if needed.

```
$ swift upload myfiles test.txt
$ swift upload myfiles test2.txt
```

4. Run the following swift command to download all files from the myfiles container:

```
$ swift download myfiles
test2.txt [headers 0.267s, total 0.267s, 0.000s MB/s]
test.txt [headers 0.271s, total 0.271s, 0.000s MB/s]
```

プロキシサーバーの追加

信頼性のために、プロキシサーバーを追加します。最初のプロキシノードをセットアップした方法と同じように追加のプロキシノードをセットアップできますが、追加の設定手順があります。

複数のプロキシを導入した後、それらを負荷分散する必要があります。ストレージエンドポイント（クライアントがストレージに接続するために使用するもの）も変更します。負荷分散のために複数の方式から選択できます。たとえば、ラウンドロビン DNS を使用できます。また、ソフトウェアやハードウェアの負荷分散装置（pound など）をプロキシの前で使用でき、ストレージ URL を負荷分散装置に向けます。

最初のプロキシノードを設定します。そして、プロキシサーバーを追加するために、これらの手順を完了します。

1. 追加のプロキシサーバーのために /etc/swift/proxy-server.conf ファイルにある memcache サーバーの一覧を更新します。複数の memcache サーバーを実行している場合、各プロキシサーバー設定ファイルで複数の IP:port の一覧に対してこのパターンを使用します。

```
10.1.2.3:11211,10.1.2.4:11211
```

```
[filter:cache]  
use = egg:swift#memcache  
memcache_servers = PROXY_LOCAL_NET_IP:11211
```

2. 新しいプロキシノードを含め、すべてのノードにリング情報をコピーします。また、リング情報がすべてのストレージノードに到達していることを確認します。
3. すべてのノードを同期した後、管理者が /etc/swift にあるキーを持ち、リングファイルの所有者が正しいことを確認します。

第11章 Orchestration Service の追加

目次

Orchestration Service 概要	88
Orchestration Service のインストール	88
Orchestration Service のインストールの検証	90

HOT と呼ばれるテンプレート言語を使用してクラウドリソースを作成するために Orchestration モジュールを使用します。統合プロジェクト名は Heat です。

Orchestration Service 概要

Orchestration Service は、クラウドアプリケーションを稼働済みにして生成するために OpenStack API コールを実行することにより、クラウドアプリケーションを記載するためのテンプレートベースのオーケストレーションを提供します。このソフトウェアは OpenStack の他のコアコンポーネントを一つのテンプレートシステムに統合します。テンプレートにより、インスタンス、Floating IP、ボリューム、セキュリティグループ、ユーザーなどのような、多くの OpenStack リソース種別を作成できます。また、インスタンスの高可用性、インスタンスのオートスケール、入れ子のスタックなどのより高度な機能をいくつか提供します。他の OpenStack コアプロジェクトと非常に緊密に統合することにより、すべての OpenStack コアプロジェクトが大規模なユーザーグループを受け取れます。

このサービスにより、開発者が Orchestration Service 直接、またはカスタムプラグイン経由で統合できるようになります。

Orchestration Service は以下のコンポーネントから構成されます。

- heat コマンドラインクライアント。AWS CloudFormation API を実行するために、heat-api と通信する CLI です。エンドの開発者は直接 Orchestration REST API を使用することもできます。
- heat-api コンポーネント。RPC 経由で API リクエストを heat-engine に送信して処理する OpenStack ネイティブの REST API を提供します。
- heat-api-cfn コンポーネント。AWS CloudFormation と互換性があり、RPC 経由で API リクエストを heat-engine に送信して処理する AWS Query API を提供します。
- heat-engine。テンプレートの開始を指示し、API コンシューマーにイベントを送り返します。

Orchestration Service のインストール

1. コントローラーノードに Orchestration モジュールをインストールします。

```
# apt-get install heat-api heat-api-cfn heat-engine
```

2. Orchestration Service がデータを保存するデータベースの場所を設定ファイルで指定します。これらの例はコントローラーノードにユーザー名 heat で MySQL データベースを使用します。HEAT_DBPASS をデータベースのユーザーの適切なパスワードで置き換えます。

/etc/heat/heat.conf を編集し、[DEFAULT] セクションを変更します。

```
[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://heat:HEAT_DBPASS@controller/heat
...
```

3. Ubuntu パッケージはデフォルトで SQLite データベースを作成します。誤って使用されないように、/var/lib/heat/ に作成された heat.sqlite ファイルを削除します。
4. root としてログインするために前に設定したパスワードを使用し、heat データベースユーザーを作成します。

```
$ mysql -u root -p
mysql> CREATE DATABASE heat;
mysql> GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'localhost' ¥
IDENTIFIED BY 'HEAT_DBPASS';
mysql> GRANT ALL PRIVILEGES ON heat.* TO 'heat'@'%' ¥
IDENTIFIED BY 'HEAT_DBPASS';
```

5. heat サービスのテーブルを作成します。

```
# heat-manage db_sync
```



注記

DeprecationWarning エラーを無視します。

6. Ubuntu パッケージはロギングを正しくセットアップしません。/etc/heat/heat.conf ファイルを編集し、[DEFAULT] セクションを変更します。

```
[DEFAULT]
...
# Print more verbose output (set logging level to INFO instead
# of default WARNING level). (boolean value)
verbose = True
...
# (Optional) The base directory used for relative --log-file
# paths (string value)
log_dir=/var/log/heat
```

7. Orchestration Service が RabbitMQ メッセージブローカーを使用するように設定します。

/etc/heat/heat.conf を編集し、[DEFAULT] セクションを変更します。

```
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

8. Orchestration サービスが Identity Service で認証するために使用する heat ユーザーを作成します。service プロジェクトを使用し、ユーザーに admin ロールを与えます。

```
$ keystone user-create --name=heat --pass=HEAT_PASS ¥  
--email=heat@example.com  
$ keystone user-role-add --user=heat --tenant=service --role=admin
```

9. Edit the `/etc/heat/heat.conf` file to change the `[keystone_authtoken]` and `[ec2authtoken]` sections to add credentials to the Orchestration Service:

```
[keystone_authtoken]  
auth_host = controller  
auth_port = 35357  
auth_protocol = http  
auth_uri = http://controller:5000/v2.0  
admin_tenant_name = service  
admin_user = heat  
admin_password = HEAT_PASS  
[ec2authtoken]  
auth_uri = http://controller:5000/v2.0
```

10. 他の OpenStack サービスから使用できるように、Heat と CloudFormation API を Identity Service に登録します。サービスを登録し、エンドポイントを指定します。

```
$ keystone service-create --name=heat --type=orchestration ¥  
--description="Orchestration"  
$ keystone endpoint-create ¥  
--service-id=$(keystone service-list | awk '/ orchestration / {print $2}') ¥  
--publicurl=http://controller:8004/v1/%(tenant_id)s ¥  
--internalurl=http://controller:8004/v1/%(tenant_id)s ¥  
--adminurl=http://controller:8004/v1/%(tenant_id)s ¥  
$ keystone service-create --name=heat-cfn --type=cloudformation ¥  
--description="Orchestration CloudFormation"  
$ keystone endpoint-create ¥  
--service-id=$(keystone service-list | awk '/ cloudformation / {print $2}') ¥  
--publicurl=http://controller:8000/v1 ¥  
--internalurl=http://controller:8000/v1 ¥  
--adminurl=http://controller:8000/v1
```

11. 新しい設定を用いてサービスを再起動します。

```
# service heat-api restart  
# service heat-api-cfn restart  
# service heat-engine restart
```

Orchestration Service のインストールの検証

Orchestration Service が正しくインストールされ、設定されていることを検証するために、クレデンシャルが `openrc.sh` ファイルに正しくセットアップされていることを確認します。このファイルを以下のように読み込みます。

```
$ source openrc.sh
```

次に、サンプルを使用して、いくつかのスタックを作成します。

スタックの管理と作成

The template languages are described in [the Template Guide](#) in the [Heat developer documentation](#).

サンプルテンプレートファイルからのスタックの作成

1. サンプルテンプレートファイルからスタックまたはテンプレートを作成するために、以下のコマンドを実行します。

```
$ heat stack-create mystack --template-file=/PATH_TO_HEAT_TEMPLATES/  
WordPress_Single_Instance.template  
--parameters="InstanceType=m1.  
large;DBUsername=USERNAME;DBPassword=PASSWORD;KeyName=HEAT_KEY;LinuxDistribution=F17"
```

指定する `--parameters` の値は、テンプレートで定義したパラメーターに依存します。ウェブサイトがテンプレートファイルを公開している場合、`--template-file` パラメーターの代わりに `--template-url` パラメーターを用いて URL を指定できます。

このコマンドは以下の出力を返します。

id	stack_name	stack_status
4c712026-dcd5-4664-90b8-0915494c1332	mystack	CREATE_IN_PROGRESS

2. You can also use the `stack-create` command to validate a template file without creating a stack from it.

そうするために、以下のコマンドを実行します。

```
$ heat stack-create mystack --template-file=/PATH_TO_HEAT_TEMPLATES/  
WordPress_Single_Instance.template
```

検証に失敗した場合、応答がエラーメッセージを返します。

スタック情報の取得

特定のスタックの状態と履歴を調査するために、いろいろなコマンドを実行できます。

- どのスタックが現在のユーザーから参照できるかを確認するために、以下のコマンドを実行します。

```
$ heat stack-list
```

id	stack_name	stack_status	creation_time
4c712026-dcd5-4664-90b8-0915494c1332	mystack	CREATE_COMPLETE	2013-04-03T23:22:08Z
7edc7480-bda5-4e1c-9d5d-f567d3b6a050	my-otherstack	CREATE_FAILED	2013-04-03T23:28:20Z

- スタックの詳細を表示するために、以下のコマンドを実行します。

```
$ heat stack-show mystack
```

- スタックはリソースの集合から構成されます。

リソースとその状態を一覧表示するために、以下のコマンドを実行します。

```
$ heat resource-list mystack
+-----+-----+-----+-----+
| logical_resource_id | resource_type | resource_status | updated_time |
+-----+-----+-----+-----+
| WikiDatabase       | AWS::EC2::Instance | CREATE_COMPLETE | 2013-04-03T23:25:56Z |
+-----+-----+-----+-----+
```

- スタックにある指定したリソースの詳細を表示するために、以下のコマンドを実行します。

```
$ heat resource-show mystack WikiDatabase
```

いくつかのリソースはリソースのライフサイクルを通して変更できるメタデータと関連づけられています。

```
$ heat resource-metadata mystack WikiDatabase
```

- 一連のイベントはスタックのライフサイクルを通して生成されます。

ライフサイクルイベントを表示するために、以下を実行します。

```
$ heat event-list mystack
+-----+-----+-----+-----+
+-----+
| logical_resource_id | id | resource_status_reason | resource_status | event_time |
|                   |   |                       |                 |           |
+-----+-----+-----+-----+
+-----+
| WikiDatabase       | 1 | state changed         | IN_PROGRESS     | 2013-04-03T23:22:09Z |
| WikiDatabase       | 2 | state changed         | CREATE_COMPLETE | 2013-04-03T23:25:56Z |
+-----+-----+-----+-----+
```

- 特定のイベントの詳細を表示するために、以下のコマンドを実行します。

```
$ heat event-show WikiDatabase 1
```

スタックの更新

- 修正したテンプレートファイルから既存のスタックを更新する場合、以下のようなコマンドを実行します。

```
$ heat stack-update mystack --template-file=/path/to/heat/templates/
WordPress_Single_Instance_v2.template
--parameters="InstanceType=m1.large;DBUsername=wp;DBPassword=
verybadpassword;KeyName=heat_key;LinuxDistribution=F17"
+-----+-----+-----+-----+
+-----+
| id | stack_name | stack_status | creation_time |
|    |           |             |              |
+-----+-----+-----+-----+
```

```
+-----+-----+-----+
+-----+
| 4c712026-dcd5-4664-90b8-0915494c1332 | mystack      | UPDATE_COMPLETE |
| 2013-04-03T23:22:08Z |
| 7edc7480-bda5-4e1c-9d5d-f567d3b6a050 | my-otherstack | CREATE_FAILED    |
| 2013-04-03T23:28:20Z |
+-----+-----+-----+
```

いくつかのリソースはすぐに更新され、他のものは新しいリソースで置き換えられます。

第12章 Telemetry モジュールの追加

目次

Telemetry	94
Telemetry モジュールのインストール	95
Telemetry 用 Compute エージェントのインストール	97
Configure the Image Service for Telemetry	99
Add the Block Storage service agent for Telemetry	99
Configure the Object Storage service for Telemetry	99
Telemetry のインストールの検証	100

Telemetry は OpenStack クラウドのモニタリングとメータリングのフレームワークを提供します。これは Ceilometer プロジェクトとしても知られています。

Telemetry

The Telemetry module:

- CPU とネットワークのコストに関する統計データを効率的に収集します。
- サービスから送られた通知を監視すること、またはインフラストラクチャーをポーリングすることにより、データを収集します。
- さまざまな運用環境に適合するよう、収集するデータの種類を設定します。REST API 経由で統計データにアクセスおよび追加をします。
- 追加のプラグインによりカスタム利用データを収集するためにフレームワークを拡張します。
- 否認できない書名付き統計情報メッセージを作成します。

システムは以下の基本的なコンポーネントから構成されます。

- コンピュートエージェント (ceilometer-agent-compute)。各コンピュートノードで実行され、リソースの使用状況の統計情報を収集します。将来的に別の種類のエージェントができるかもしれませんが、今のところコンピュートエージェントの作成に注力しています。
- 中央エージェント (ceilometer-agent-central)。インスタンスやコンピュートノードに結びつけられていないリソースに対して、リソースの利用状況の統計情報を収集するために、中央管理サーバーで実行されます。
- コレクター (ceilometer-collector)。(エージェントから送られてくる通知や統計情報に対する) メッセージキューを監視するために、一つまたは複数の中央管理サーバーで実行されます。通知メッセージが処理され、統計情報メッセージに変えられます。適切なトピックを使用してメッセージバスの中に送り返されます。Telemetry メッセージは変更せずにデータストアに書き込まれます。

- アラーム通知 (ceilometer-alarm-notifier)。いくつかの標本に対する閾値評価に基づいてアラームを設定できるようにするために、一つまたは複数の中央管理サーバーで実行されます。
- データストア。(一つまたは複数のコレクターインスタンスからの) 同時書き込みや (API サーバーからの) 同時読み込みを処理できる能力のあるデータベースです。
- An API server (ceilometer-api). Runs on one or more central management servers to provide access to the data from the data store.

これらのサービスは標準的な OpenStack メッセージバスを使用して通信します。コレクターと API サーバーのみがデータストアにアクセスできます。

Telemetry モジュールのインストール

Telemetry は情報収集機能とさまざまな種類のエージェントを提供する API サービスです。コンピュータノードのようなノードにこれらのエージェントをインストールする前に、コントローラーノードにコアコンポーネントをインストールするために、この手順を使用する必要があります。

1. コントローラーノードに Telemetry Service をインストールします。

```
# apt-get install ceilometer-api ceilometer-collector ceilometer-agent-central python-ceilometerclient
```

2. Telemetry Service は情報を保存するためにデータベースを使用します。設定ファイルでデータベースの場所を指定します。この例はコントローラーノードで MongoDB データベースを使用します。

```
# apt-get install mongodb-server
```



注記

MongoDB はデフォルトで、データベースのジャーナリングをサポートするために、`/var/lib/mongodb/journal/` ディレクトリにいくつかの 1GB のファイルを作成するよう設定されます。

データベースのジャーナリングをサポートするために割り当てられる領域を最小化する場合、`/etc/mongodb.conf` 設定ファイルにある `smallfiles` 設定キーを `true` に設定します。この設定により、各ジャーナルファイルの容量が 512MB に減ります。

MongoDB サービスを初めて起動するときに、このファイルが作成されるので、この変更を反映するためにサービスを停止して、このファイルを削除する必要があります。

```
# service mongodb stop
# rm /var/lib/mongodb/journal/j._0
# rm /var/lib/mongodb/journal/prealloc.1
# rm /var/lib/mongodb/journal/prealloc.2
# service mongodb start
```

`smallfiles` 設定キーの詳細は MongoDB のドキュメント <http://docs.mongodb.org/manual/reference/configuration-options/#smallfiles> を参照してください。

データベースのジャーナリング自体を無効化する手順の詳細は <http://docs.mongodb.org/manual/tutorial/manage-journaling/> を参照してください。

3. コントローラーのパブリック IP アドレスをリッスンするよう MongoDB を設定します。/etc/mongodb.conf ファイルを編集し、bind_ip キーを変更します。

```
bind_ip = 192.168.0.10
```

4. 設定の変更を適用するために MongoDB のサービスを再起動します。

```
# service mongodb restart
```

5. データベースと ceilometer データベースユーザーを作成します。

```
# mongo --host controller --eval '
db = db.getSiblingDB("ceilometer");
db.addUser({user: "ceilometer",
            pwd: "CEILOMETER_DBPASS",
            roles: [ "readWrite", "dbAdmin" ]})'
```

6. Telemetry Service がデータベースを使用するよう設定します。

/etc/ceilometer/ceilometer.conf ファイルを編集し、[database] セクションを変更します。

```
[database]
# The SQLAlchemy connection string used to connect to the
# database (string value)
connection = mongodb://ceilometer:CEILOMETER_DBPASS@controller:27017/ceilometer
```

7. You must define an secret key that is used as a shared secret among Telemetry service nodes. Use openssl to generate a random token and store it in the configuration file:

```
# openssl rand -hex 10
```

/etc/ceilometer/ceilometer.conf ファイルを編集し、[publisher_rpc] セクションを変更します。CEILOMETER_TOKEN を openssl コマンドの結果で置き換えます。

```
[publisher_rpc]
# Secret value for signing metering messages (string value)
metering_secret = CEILOMETER_TOKEN
```

8. RabbitMQ のアクセス権を設定します。

/etc/ceilometer/ceilometer.conf ファイルを編集し、[DEFAULT] セクションを更新します。

```
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

9. ログディレクトリを設定します。

/etc/ceilometer/ceilometer.conf ファイルを編集し、[DEFAULT] セクションを更新します。

```
[DEFAULT]
log_dir = /var/log/ceilometer
```

10. Telemetry Service が Identity Service で認証するために使用する ceilometer ユーザーを作成します。service プロジェクトを使用し、ユーザーに admin ロールを与えます。

```
$ keystone user-create --name=ceilometer --pass=CEILOMETER_PASS --
email=ceilometer@example.com
$ keystone user-role-add --user=ceilometer --tenant=service --role=admin
```

11. クレデンシャルを Telemetry Service の設定ファイルに追加します。

/etc/ceilometer/ceilometer.conf ファイルを編集し、[keystone_authtoken] セクションを変更します。

```
[keystone_authtoken]
auth_host = controller
auth_port = 35357
auth_protocol = http
auth_uri = http://controller:5000
admin_tenant_name = service
admin_user = ceilometer
admin_password = CEILOMETER_PASS
```

[service_credentials] セクションも設定します。

```
[service_credentials]
os_username = ceilometer
os_tenant_name = service
os_password = CEILOMETER_PASS
```

12. Register the Telemetry service with the Identity Service so that other OpenStack services can locate it. Use the keystone command to register the service and specify the endpoint:

```
$ keystone service-create --name=ceilometer --type=metering ¥
--description="Telemetry"
$ keystone endpoint-create ¥
--service-id=$(keystone service-list | awk '/ metering / {print $2}') ¥
--publicurl=http://controller:8777 ¥
--internalurl=http://controller:8777 ¥
--adminurl=http://controller:8777
```

13. 新しい設定を用いてサービスを再起動します。

```
# service ceilometer-agent-central restart
# service ceilometer-api restart
# service ceilometer-collector restart
```

Telemetry 用 Compute エージェントのインストール

Telemetry は情報収集機能とさまざまな種類のエージェントを提供する API サービスを提供します。この手順はコンピュータノードで実行するエージェントをインストールする方法を詳細に説明します。

1. コンピュートノードに Telemetry Service をインストールします。

```
# apt-get install ceilometer-agent-compute
```

2. /etc/nova/nova.conf ファイルを編集し、[DEFAULT] セクションに以下の行を追加します。

```
[DEFAULT]
...
instance_usage_audit = True
instance_usage_audit_period = hour
notify_on_state_change = vm_and_task_state
notification_driver = nova.openstack.common.notifier.rpc_notifier
notification_driver = ceilometer.compute.nova_notifier
```

3. Compute Service を再起動します。

```
# service nova-compute restart
```

4. 前に設定したシークレットキーを設定する必要があります。Telemetry Service ノードは共有シークレットとしてこのキーを共有します。

/etc/ceilometer/ceilometer.conf ファイルを編集し、[DEFAULT] セクションでこれらの行を変更します。CEILOMETER_TOKEN を前に作成した ceilometer トークンに置き換えます。

```
[publisher_rpc]
# Secret value for signing metering messages (string value)
metering_secret = CEILOMETER_TOKEN
```

5. RabbitMQ のアクセス権を設定します。

/etc/ceilometer/ceilometer.conf ファイルを編集し、[DEFAULT] セクションを更新します。

```
[DEFAULT]
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

6. Identity Service のクレデンシャルを追加します。

/etc/ceilometer/ceilometer.conf ファイルを編集し、[keystone_authtoken] セクションを変更します。

```
[keystone_authtoken]
auth_host = controller
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = ceilometer
admin_password = CEILOMETER_PASS
```

[service_credentials] セクションも設定します。

```
[service_credentials]
os_auth_url = http://controller:5000/v2.0
os_username = ceilometer
os_tenant_name = service
os_password = CEILOMETER_PASS
```

7. ログディレクトリを設定します。

/etc/ceilometer/ceilometer.conf ファイルを編集し、[DEFAULT] セクションを更新します。

```
[DEFAULT]
log_dir = /var/log/ceilometer
```

8. 新しい設定を用いてサービスを再起動します。

```
# service ceilometer-agent-compute restart
```

Configure the Image Service for Telemetry

1. イメージのサンプルを取得するために、Image Service がバスに通知を送信するように設定する必要があります。

/etc/glance/glance-api.conf を編集し、[DEFAULT] セクションを変更します。

```
notification_driver = messaging
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
```

2. Restart the Image Services with their new settings:

```
# service glance-registry restart
# service glance-api restart
```

Add the Block Storage service agent for Telemetry

1. To retrieve volume samples, you must configure the Block Storage service to send notifications to the bus.

/etc/cinder/cinder.conf を編集し、[DEFAULT] セクションに追加します。

```
control_exchange = cinder
notification_driver = cinder.openstack.common.notifier.rpc_notifier
```

2. 新しい設定を用いて Block Storage サービスを再起動します。

```
# service cinder-volume restart
# service cinder-api restart
```

Configure the Object Storage service for Telemetry

1. オブジェクトストアの統計情報を取得するためには、Telemetry Service が ResellerAdmin ロールで Object Storage にアクセスする必要があります。このロールを os_tenant_name プロジェクトの os_username ユーザーに与えます。

```
$ keystone role-create --name=ResellerAdmin
+-----+-----+
+-----+-----+
+-----+-----+
```

Property	Value
id	462fa46c13fd4798a95a3bfbe27b5e54
name	ResellerAdmin

```
$ keystone user-role-add --tenant service --user ceilometer --role 462fa46c13fd4798a95a3bfbe27b5e54
```

2. 入力通信と出力通信を処理するために、Telemetry ミドルウェアを Object Storage に追加する必要があります。これらの行を `/etc/swift/proxy-server.conf` ファイルに追加します。

```
[filter:ceilometer]  
use = egg:ceilometer#swift
```

3. `ceilometer` を同じファイルの `pipeline` パラメーターに追加します。

```
[pipeline:main]  
pipeline = healthcheck cache authtoken keystoneauth ceilometer proxy-server
```

4. 新しい設定を用いてサービスを再起動します。

```
# service swift-proxy restart
```

Telemetry のインストールの検証

To test the Telemetry installation, download an image from the Image Service, and use the `ceilometer` command to display usage statistics.

1. Telemetry へのアクセスをテストするために `ceilometer meter-list` コマンドを使用します。

```
$ ceilometer meter-list
```

Name	Type	Unit	Resource ID	User ID	Project
image	gauge	image	9e5c2bee-0373-414c-b4af-b91b0246ad3b	None	
e66d97ac1b704897853412fc8450f7b9					
image.size	gauge	B	9e5c2bee-0373-414c-b4af-b91b0246ad3b	None	
e66d97ac1b704897853412fc8450f7b9					

2. Image Service からイメージをダウンロードします。

```
$ glance image-download "Cirros 0.3.1" > cirros.img
```

3. このダウンロードが Telemetry により検知され、保存されていることを検証するために `ceilometer meter-list` コマンドを呼び出します。

```
$ ceilometer meter-list
```

Name	Type	Unit	Resource ID	User ID	Project
image	gauge	image	9e5c2bee-0373-414c-b4af-b91b0246ad3b	None	
e66d97ac1b704897853412fc8450f7b9					
image.size	gauge	B	9e5c2bee-0373-414c-b4af-b91b0246ad3b	None	
e66d97ac1b704897853412fc8450f7b9					

Name	Type	Unit	Resource ID	User ID
Project ID				
image	gauge	image	9e5c2bee-0373-414c-b4af-b91b0246ad3b	None
e66d97ac1b704897853412fc8450f7b9				
image.download	delta	B	9e5c2bee-0373-414c-b4af-b91b0246ad3b	None
e66d97ac1b704897853412fc8450f7b9				
image.serve	delta	B	9e5c2bee-0373-414c-b4af-b91b0246ad3b	None
e66d97ac1b704897853412fc8450f7b9				
image.size	gauge	B	9e5c2bee-0373-414c-b4af-b91b0246ad3b	None
e66d97ac1b704897853412fc8450f7b9				

4. さまざまなメーターの使用量の統計情報を取得できるようになりました。

```
$ ceilometer statistics -m image.download -p 60
```

Period	Period Start	Period End	Count	Min	Max
Sum	Avg	Duration	Duration Start	Duration End	
60	2013-11-18T18:08:50	2013-11-18T18:09:50	1	13147648.0	13147648.0
13147648.0	13147648.0	0.0	2013-11-18T18:09:05.334000	2013-11-18T18:09:05.334000	

付録A 予約済みユーザー ID

OpenStack では、特定のユーザー ID が特定の OpenStack サービスを実行し、特定の OpenStack ファイルを所有するために、予約され、使用されます。これらのユーザーはディストリビューションのパッケージにより設定されます。以下の表はその概要です。



注記

いくつかの OpenStack パッケージはインストール中にユーザー ID を自動的に生成し、割り当てます。これらのケースで、ユーザー ID 値が重要ではありません。ユーザー ID の存在が重要です。

表A.1 予約済みユーザー ID

名前	説明	ID
ceilometer	OpenStack Ceilometer デーモン	パッケージインストール中の割り当て
cinder	OpenStack Cinder デーモン	パッケージインストール中の割り当て
glance	OpenStack Glance デーモン	パッケージインストール中の割り当て
heat	OpenStack Heat デーモン	パッケージインストール中の割り当て
keystone	OpenStack Keystone デーモン	パッケージインストール中の割り当て
neutron	OpenStack Neutron デーモン	パッケージインストール中の割り当て
nova	OpenStack Nova デーモン	パッケージインストール中の割り当て
swift	OpenStack Swift デーモン	パッケージインストール中の割り当て

各ユーザーはユーザーと同じ名前のユーザーグループに所属します。

付録B コミュニティのサポート

目次

ドキュメント	103
ask.openstack.org	104
OpenStack メールングリスト	104
OpenStack wiki	104
Launchpad バグエリア	105
OpenStack IRC チャンネル	106
ドキュメントへのフィードバック	106
OpenStackディストリビューション	106

OpenStackの利用に役立つ、多くのリソースがあります。OpenStackコミュニティのメンバーはあなたの質問に回答するでしょうし、バグ調査のお手伝いもします。コミュニティはOpenStackを継続的に改善、機能追加していますが、もしあなたが何らかの問題に直面したら、遠慮せずに相談してください。下記のリソースをOpenStackのサポートとトラブルシューティングに活用して下さい。

ドキュメント

OpenStackのドキュメントは、 docs.openstack.orgを参照してください。

ドキュメントにフィードバックするには、 [OpenStack Documentation Mailing List](http://openstack-documentation mailing list)の <openstack-docs@lists.openstack.org>か、Launchpadの[report a bug](#)を活用してください。

OpenStackクラウドと関連コンポーネントの導入ガイド:

- [Installation Guide for Debian 7.0](#)
- [Installation Guide for openSUSE and SUSE Linux Enterprise Server](#)
- [Red Hat Enterprise Linux, CentOS, and Fedora向けインストールガイド](#)
- [Ubuntu 12.04 \(LTS\)向けインストールガイド](#)

OpenStackクラウドの構成と実行ガイド:

- [Cloud Administrator Guide](#)
- [Configuration Reference](#)
- [Operations Guide](#)
- [High Availability Guide](#)
- [Security Guide](#)
- [Virtual Machine Image Guide](#)

OpenStackダッシュボードとCLIクライアントガイド

- [API Quick Start](#)
- [End User Guide](#)
- [Admin User Guide](#)
- [コマンドラインインターフェースのリファレンス](#)

OpenStack APIのリファレンスガイド

- [OpenStack API Reference](#)
- [OpenStack Block Storage Service API v2 Reference](#)
- [OpenStack Compute API v2 and Extensions Reference](#)
- [OpenStack Identity Service API v2.0 Reference](#)
- [OpenStack Identity Service API v2.0 Reference](#)
- [OpenStack Networking API v2.0 Reference](#)
- [OpenStack Object Storage API v1 Reference](#)

[トレーニングガイド](#)はクラウド管理者向けのソフトウェアトレーニングを提供します。

ask.openstack.org

OpenStackの導入やテスト中、特定のタスクが完了したのか、うまく動いていないのかを質問したくなるかもしれません。その時は、[ask.openstack.org](#)が役に立ちます。[ask.openstack.org](#)で、すでに同様の質問に回答がないかを確認してみてください。もしなければ、質問しましょう。簡潔で明瞭なサマリーをタイトルにし、できるだけ詳細な情報を記入してください。コマンドの出力結果やスタックトレース、スクリーンショットへのリンクなどがいいでしょう。

OpenStack メーリングリスト

回答やヒントを得るとっておきの方法は、OpenStackメーリングリストへ質問や問題の状況を投稿することです。同様の問題に対処したことのある仲間が助けてくれることでしょう。購読の手続き、アーカイブの参照は<http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack>で行ってください。特定プロジェクトや環境についてのメーリングリストは、[on the wiki](#)で探してみましょう。すべてのメーリングリストは、<http://wiki.openstack.org/MailingLists>で参照できます。

OpenStack wiki

[OpenStack wiki](#)は広い範囲のトピックを扱っていますが、情報によっては、探すのが難しかったり、情報が少なかったりします。幸いなことに、wikiの検索機能にて、タイトルと内容で探せます。もし特定の情報、たとえばネットワークや novaについて探すのであれ

ば、多くの関連情報を見つけられます。日々追加されているため、こまめに確認してみてください。OpenStack wikiページの右上に、その検索窓があります。

Launchpad バグエリア

OpenStackコミュニティはあなたのセットアップ、テストの取り組みに価値を感じており、フィードバックを求めています。バグを登録するには、<https://launchpad.net/+login>でLaunchpadのアカウントを作成してください。Launchpadバグエリアにて、既知のバグの確認と報告ができます。すでにそのバグが報告、解決されていないかを判断するため、検索機能を活用してください。もしそのバグが報告されていないければ、バグレポートを入力しましょう。

使いこなすヒント:

- 明瞭で簡潔なまとめを!
- できるだけ詳細な情報を記入してください。コマンドの出力結果やスタックトレース、スクリーンショットへのリンクなどがいいでしょう。
- ソフトウェアとパッケージのバージョンを含めることを忘れずに。特に開発ブランチは”Grizzly release” vs git commit bc79c3ecc55929bac585d04a03475b72e06a3208のよう
に明記しましょう。
- 環境固有のお役立ち情報、たとえばUbuntu 12.04や複数ノードインストール

Launchpadバグエリアは下記リンクを参照してください。

- [Bugs: OpenStack Block Storage \(cinder\)](#)
- [Bugs: OpenStack Compute \(nova\)](#)
- [Bugs : OpenStack Dashboard \(horizon\)](#)
- [Bugs : OpenStack Identity \(keystone\)](#)
- [Bugs : OpenStack Image Service \(glance\)](#)
- [Bugs : OpenStack Networking \(neutron\)](#)
- [Bugs : OpenStack Object Storage \(swift\)](#)
- [Bugs: Bare Metal \(ironic\)](#)
- [Bugs: Data Processing Service \(sahara\)](#)
- [Bugs: Database Service \(trove\)](#)
- [Bugs: Orchestration \(heat\)](#)
- [Bugs: Telemetry \(ceilometer\)](#)
- [Bugs: Queue Service \(marconi\)](#)
- [Bugs: OpenStack API Documentation \(api.openstack.org\)](#)

- [Bugs: OpenStack Documentation \(docs.openstack.org\)](http://docs.openstack.org)

OpenStack IRC チャネル

OpenStackコミュニティはFreenode上の#openstack IRCチャネルを活用しています。あなたはそこに訪れ、質問することで、差し迫った問題へのフィードバックを迅速に得られます。IRCクライアントをインストール、もしくはブラウザベースのクライアントを使うには、<http://webchat.freenode.net/>にアクセスしてください。また、Colloquy (Mac OS X, <http://colloquy.info/>), mIRC (Windows, <http://www.mirc.com/>), or XChat (Linux)なども使えます。IRCチャネル上でコードやコマンド出力結果を共有したい時には、Paste Binが多く使われています。OpenStackプロジェクトのPaste Binは<http://paste.openstack.org>です。長めのテキストやログであっても、webフォームに貼り付けてURLを得るだけです。OpenStack IRCチャネルは、#openstack on irc.freenode.netです。OpenStack関連IRCチャネルは、<https://wiki.openstack.org/wiki/IRC>にリストがあります。

ドキュメントへのフィードバック

ドキュメントにフィードバックするには、[OpenStack Documentation Mailing List](mailto:openstack-docs@lists.openstack.org)の<openstack-docs@lists.openstack.org>か、Launchpadの[report a bug](#)を活用してください。

OpenStackディストリビューション

OpenStackのコミュニティサポート版を提供しているディストリビューション

- Debian: <http://wiki.debian.org/OpenStack>
- CentOS, Fedora, およびRed Hat Enterprise Linux: <http://openstack.redhat.com/>
- openSUSEとSUSE Linux Enterprise Server: <http://en.opensuse.org/Portal:OpenStack>
- Ubuntu: <https://wiki.ubuntu.com/ServerTeam/CloudArchive>

用語集

API

アプリケーションプログラミングインターフェース。

認証

ユーザー、プロセスまたはクライアントが、秘密鍵、秘密トークン、パスワード、指紋または同様の方式により示されている主体と本当に同じであることを確認するプロセス。

コントローラーノード

クラウドコントローラーノードの別名。

クレデンシャル

Data that is only known to or accessible by a user and used to verify that the user is who they say they are. Credentials are presented to the server during authentication. Examples include a password, secret key, digital certificate, fingerprint, and so on.

DHCP

Dynamic Host Configuration Protocol. A network protocol that configures devices that are connected to a network so that they can communicate on that network by using the Internet Protocol (IP). The protocol is implemented in a client-server model where DHCP clients request configuration data such as, an IP address, a default route, and one or more DNS server addresses from a DHCP server.

エンドポイント

API エンドポイントを参照。

IaaS

Infrastructure-as-a-Service. IaaS is a provisioning model in which an organization outsources physical components of a data center such as storage, hardware, servers and networking components. A service provider owns the equipment and is responsible for housing, operating and maintaining it. The client typically pays on a per-use basis. IaaS is a model for providing cloud services.

Icehouse

OpenStack の 9 回目リリースのプロジェクト名。

Image Service

An OpenStack core project that provides discovery, registration, and delivery services for disk and server images. The project name of the Image Service is glance.

interface

A physical or virtual device that provides connectivity to another device or medium.

kernel-based VM (KVM)

OpenStack がサポートするハイパーバイザーの 1 つ。

multi-host

High-availability mode for legacy (nova) networking. Each compute node handles NAT and DHCP and acts as a gateway for all of the VMs on it. A networking failure on one compute node doesn't affect VMs on other compute nodes.

Network Address Translation (NAT)

The process of modifying IP address information while in-transit. Supported by Compute and Networking.

Network Time Protocol (NTP)

A method of keeping a clock for a host or node correct through communications with a trusted, accurate time source.

OpenStack

OpenStack is a cloud operating system that controls large pools of compute, storage, and networking resources throughout a data center, all managed through a dashboard that gives administrators control while empowering their users to provision resources through a web interface. OpenStack is an open source project licensed under the Apache License 2.0.

plug-in

Software component providing the actual implementation for Networking APIs, or for Compute APIs, depending on the context.

ポート

A virtual network port within Networking; VIFs / vNICs are connected to a port.

プロジェクト

A logical grouping of users within Compute, used to define quotas and access to VM images.

Qpid

Message queue software supported by OpenStack; an alternative to RabbitMQ.

RabbitMQ

OpenStackでデフォルトで採用されているメッセージキューのソフトウェア。

RESTful

A kind of web service API that uses REST, or Representational State Transfer. REST is the style of architecture for hypermedia systems that is used for the World Wide Web.

role

ユーザーが特定の操作の組を実行できると仮定する人格。ロールは一組の権利と権限を含みます。そのロールを仮定しているユーザーは、それらの権利と権限を継承します。

router

A physical or virtual network device that passes network traffic between different networks.

セキュリティグループ

A set of network traffic filtering rules that are applied to a Compute instance.

サービスカタログ

Alternative term for the Identity Service catalog.

静的 IP アドレス

固定 IP アドレスの別名。

subnet

Logical subdivision of an IP network.

Telemetry

An integrated project that provides metering and measuring facilities for OpenStack. The project name of Telemetry is ceilometer.

テナント

A group of users, used to isolate access to Compute resources. An alternative term for a project.

トークン

OpenStack API やリソースへのアクセスに使用される英数字文字列。

ユーザー

In Identity Service, each user is associated with one or more tenants, and in Compute can be associated with roles, projects, or both.

virtual networking

A generic term for virtualization of network functions such as switching, routing, load balancing, and security using a combination of VMs and overlays on physical network infrastructure.

ZeroMQ

OpenStack によりサポートされるメッセージキューソフトウェア。RabbitMQ の代替。0MQ とも表記。