OpenStack Ussuri構築手順書

1. 目次

2. VMの箱の作成

- それぞれの要件に合わせて作成していきます。
  また、OpenSuse 15.3を最小構成でインストールしていきます。

2.1 最低スペック

項目	contoroller	compute	Block Storage	Object Storage
vCPU	2[vCPU]	8[vCPU]	2[vCPU]	2[vCPU]
MEM	8[GB]	16[GB]	4[GB]	8[GB]
HDD	100[GB]	200[GB]	100[GB]	100[GB]
NIC	2 [NIC]	2 [NIC]	1 [NIC]	1 [NIC]

3. 事前準備

- OpenStackを構築するのに必要な、事前の準備をする章です。

3.1. パスワード

- 本章ではパスワードを決めていきます。
- 本環境では、以下のパスワードを設定します。本構築手順では、デフォルトのまま設定します。 必要に応じて、デフォルトのパスワードから変更してください。

パスワード名	説明
DB_PASS	データベースのルートパスワード
ADMIN_PASS	admin ユーザーのパスワード
CINDER_DBPASS	Block Storage サービスのデータベースのパスワード
CINDER_PASS	Block Storage サービスのユーザー cinder のパスワード
DASH_DBPASS	Dashboard のデータベースのパスワード
DEMO_PASS	demo ユーザーのパスワード
GLANCE_DBPASS	Image service のデータベースのパスワード
GLANCE_PASS	Image service のユーザー glance のパスワード
KEYSTONE_DBPASS	Identity サービスのデータベースのパスワード
METADATA_SECRET	メタデータプロキシー用のシークレット
NEUTRON_DBPASS	Networking サービスのデータベースのパスワード
NEUTRON_PASS	Networking サービスのユーザー neutron のパスワード
NOVA_DBPASS	Compute サービスのデータベースのパスワード
NOVA_PASS	Compute サービスのユーザー nova のパスワード
PLACEMENT_PASS	Password of the Placement service user placement
RABBIT_PASS	Password of RabbitMQ user openstack

3.2. ネットワーク

- 管理ネットワークのIPアドレスと名前解決の設定をしていきます。

3.2.1 コントローラーノード

1. 1番目のインターフェースを管理インターフェースとして以下の設定値を設定します。

    - IP アドレス: 192.168.1.11
      ネットマスク: 255.255.255.0 (または /24)
      デフォルトゲートウェイ: 192.168.1.1

2. プロバイダーインターフェースは、IP アドレスを割り当てない特別な設定を使用します。

   2 番目のインターフェースをプロバイダーインターフェースとして設定します。

   - 以下のファイルを編集し、以下の内容を追加します。

     ```
     vim /etc/sysconfig/network/ifcfg-INTERFACE_NAME
     ```

     ```
     STARTMODE='auto'
     BOOTPROTO='static'
     ```

3. 名前解決の設定をします。

   ```
   # controller
   192.168.1.11 controller
   # compute1
   192.168.1.31 compute1
   # block1
   192.168.1.41 block1
   # object1
   192.168.1.51 object1
   # object2
   192.168.1.52 object2
   ```

3.2.2 コンピュートノード

1. 1番目のインターフェースを管理インターフェースとして以下の設定値を設定します。

   - IP アドレス: 192.168.1.31
     ネットマスク: 255.255.255.0 (または /24)
     デフォルトゲートウェイ: 192.168.1.1

2. プロバイダーインターフェースは、IP アドレスを割り当てない特別な設定を使用します。 2 番目のインターフェースをプロバイダーインターフェースとして設定します。

   - ファイルを編集し、以下の内容を追加します。

     ```
     vim /etc/sysconfig/network/ifcfg-INTERFACE_NAME
     ```

     ```
     STARTMODE='auto'
     BOOTPROTO='static'
     ```

3. 名前解決の設定をします。

```
# controller
192.168.1.11 controller
# compute1
192.168.1.31 compute1
# block1
192.168.1.41 block1
# object1
192.168.1.51 object1
# object2
192.168.1.52 object2
```

> Note
> いくつかのディストリビューションでは、実際のホスト名を 127.0.1.1 といった別のループバックアドレスに名前解決する、本質的でない項目がファイル /etc/hosts に追加されます。名前解決の問題を防ぐために、この項目をコメントアウトするか削除してください。ただし 127.0.0.1 の項目は削除しないこと。

3.2.3 ブロックストレージノード

1. 1番目のインターフェースを管理インターフェースとして以下の設定値を設定します。
    - IP アドレス: 192.168.1.41
      ネットマスク: 255.255.255.0 (または /24)
      デフォルトゲートウェイ: 192.168.1.1
2. プロバイダーインターフェースは、IP アドレスを割り当てない特別な設定を使用します。
   2 番目のインターフェースをプロバイダーインターフェースとして設定します。
    - ファイル を編集し、以下の内容を追加します。

      ```
      vim /etc/sysconfig/network/ifcfg-INTERFACE_NAME
      ```

      ```
      STARTMODE='auto'
      BOOTPROTO='static'
      ```

3. 名前解決の設定をします。

```
# controller
192.168.1.11 controller
# compute1
192.168.1.31 compute1
# block1
192.168.1.41 block1
# object1
192.168.1.51 object1
# object2
192.168.1.52 object2
```

> Note
> いくつかのディストリビューションでは、実際のホスト名を 127.0.1.1 といった別のループバックアドレスに名前解決する、本質的でない項目がファイル /etc/hosts に追加されます。名前解決の問題を防ぐために、この項目をコメントアウトするか削除してください。ただし 127.0.0.1 の項目は削除しないこと。

3.2.4 オブジェクトノード

1. 1番目のインターフェースを管理インターフェースとして以下の設定値を設定します。
   - IP アドレス: 192.168.1.51
     ネットマスク: 255.255.255.0 (または /24)
     デフォルトゲートウェイ: 192.168.1.1
2. プロバイダーインターフェースは、IP アドレスを割り当てない特別な設定を使用します。
   2 番目のインターフェースをプロバイダーインターフェースとして設定します。
   - ファイル を編集し、以下の内容を追加します。

     > vim /etc/sysconfig/network/ifcfg-INTERFACE_NAME

     ```
     STARTMODE='auto'
     BOOTPROTO='static'
     ```
3. 名前解決の設定をします。

```
# controller
192.168.1.11 controller
# compute1
192.168.1.31 compute1
# block1
192.168.1.41 block1
# object1
192.168.1.51 object1
# object2
192.168.1.52 object2
```

> Note
> いくつかのディストリビューションでは、実際のホスト名を 127.0.1.1 といった別のループバックアドレスに名前解決する、本質的でない項目がファイル /etc/hosts に追加されます。名前解決の問題を防ぐために、この項目をコメントアウトするか削除してください。ただし 127.0.0.1 の項目は削除しないこと。

3.2.5 接続性の検証

1. コントローラーノードから、インターネットへのアクセスをテストします。

> ping -c 4 www.openstack.org

2. コントローラーノードから、コンピュートノードの管理インターフェースへのアクセスをテストします。

```
ping -c 4 compute1
```

3. コンピュートノードから、インターネットへのアクセスをテストします。

```
ping -c 4 www.openstack.org
```

4. コンピュートノードから、コントローラーノードの管理インターフェースへのアクセスをテストします。

```
ping -c 4 controller
```

3.3. NTP

サービスをノード間で正しく同期するために、NTP を実装している Chrony をインストールします。
コントローラーノードをできる限り正確な（低ストラタム値）参照サーバーに設定し、他のノードからコントローラーノードを参照するよう設定します。

3.3.1 コントローラーノード

1. パッケージをインストールします。

```
# zypper install chrony
```

2. ファイル /etc/chrony.conf を編集して、お使いの環境の必要に応じて、これらの項目の追加、変更、削除を行います。

```
# vim /etc/chrony.conf
```

```
server NTP_SERVER iburst
```

NTP_SERVER を適切なより正確な（ストラタムが小さい値の）NTP サーバーのホスト名か IP アドレスに置き換えます。複数の server キーを設定することもできます。

3. 他のノードがコントローラーノードの chrony デーモンに接続できるよう、これらの項目を /etc/chrony.conf に追加します。

```
allow 10.0.0.0/24
```

4. ファイアウォールを設定している場合は、以下のコマンドを実行します。

```
# firewall-cmd --add-service=ntp --permanent
# firewall-cmd --reload
```

5. NTP サービスを起動し、システム起動時に起動するよう設定します。

```
# systemctl enable chronyd.service
# systemctl restart chronyd.service
```

3.3.2 その他のノード

1. パッケージをインストールします。

```
# zypper install chrony
```

2. ファイル /etc/chrony.conf を編集して、server 項目を 1 つを除いてすべてコメントアウトまたは削除します。残した項目はコントローラーノードを参照するよう変更します。

```
server controller iburst
```

3. NTP サービスを起動し、システム起動時に起動するよう設定します。

```
# systemctl enable --now chronyd.service
```

3.3.3 動作検証

1. このコマンドを コントローラー ノードで実行します。

```
# chronyc sources
```

2. 同じコマンドを すべてのノード において実行します。

```
# chronyc sources
```

3.4. Openstackパッケージ

OpenStack パッケージのセットアップは、**コントローラーノード、コンピュートノード、ブロックストレージノードすべてで実行する必要があります。** OpenStack 環境に影響を与える可能性があるため、自動更新サービスを無効化または削除します。

> note
> この先に進む前に、お使いのディストリビューションの基本インストールパッケージを最新バージョンに
> 更新する必要があります。

OpenStack リポジトリーの有効化

1. 以下のコマンドを実行して、リポジトリを有効化します。

```
# zypper addrepo -f obs://Cloud:/OpenStack:/Ussuri/openSUSE_Leap_15.2/ Ussuri
```

2. openSUSE ディストリビューションは、パッケージ群を表現するために、パターンという概念が使用します。初期インストール中に 'Minimal Server Selection (Text Mode)' を選択した場合、OpenStack パッケージをインストールしようとした際に、依存関係の競合が発生するかもしれません。これを避けるために、minimal_base-conflicts パッケージを削除します。

```
# zypper rm patterns-openSUSE-minimal_base-conflicts
```

3. すべてのノードでパッケージをアップグレードします。

```
# zypper refresh && zypper dist-upgrade
```

4. OpenStack クライアントのインストールします。

```
# zypper install -y python2-openstackclient
```

3.5. SQL

ディストリビューションに応じてMariaDBまたはMySQLを使用します。今回はMariaDBをコントローラーノードにインストールしていきます。

1. SQLのパッケージをインストールします

```
# zypper install -y mariadb-client mariadb python3-PyMySQL
```

2. ファイル /etc/my.cnf.d/openstack.cnf を新規作成、編集し、以下の作業をすべて行います。

   - [mysqld] セクションを作成して、bind-address にコントローラーノードの管理 IP アドレスを設定して、管理ネットワーク経由で他のノードよりアクセスできるようにします。その他、有用なオプションや UTF-8 文字セットを有効化するためのキーを設定します。

```
# vim /etc/my.cnf.d/openstack.cnf
```

```
[mysqld]
bind-address = 192.168.1.11
default-storage-engine = innodb
innodb_file_per_table = on
max_connections = 4096
collation-server = utf8_general_ci
character-set-server = utf8
```

3. データベースサービスを起動し、システム起動時に自動的に起動するよう設定します。

```
# systemctl enable --now mysql.service
```

4. mysql_secure_installation スクリプトを実行して、データベースサービスの安全性を向上します。特に、データベースの root アカウントに適切なパスワードを設定します。

```
# mysql_secure_installation
```

3.6. メッセージキュー

OpenStackは、メッセージキューを使用して、サービス間の操作とステータス情報を調整します。 Rabbimqをコントローラーノードにインストールしていきます。

1. パッケージをインストールします。

```
# zypper install -y rabbitmq-server
```

2. メッセージキューサービスを起動し、システム起動時に起動するよう設定します。

```
# systemctl enable rabbitmq-server.service
```

systemctl start rabbitmq-server.service

3. openstack ユーザーを追加します。RABBIT_PASS を適切なパスワードに置き換えます。

```
# rabbitmqctl add_user openstack RABBIT_PASS
```

4. openstack ユーザーに対して、設定、書き込み、読み出しアクセスを許可します。

```
# rabbitmqctl set_permissions openstack "." "." ".*"
```

3.7. Memcached

- コンポーネントのインストールと設定をしていきます。

1. パッケージをインストールします。

```
# zypper install -y memcached python2-python-memcached
```

2. コントローラノードの管理IPアドレスを使用するようにサービスを設定します。

```
# vim /etc/sysconfig/memcached
```

```
MEMCACHED_PARAMS="-l 192.168.1.11"
```

3. memcached サービスを起動し、システム起動時に起動するよう設定します。

```
# systemctl enable --now memcached.service
```

3.8 etcd

1. ユーザーを作成します。

```
# groupadd --system etcd
# useradd --home-dir "/var/lib/etcd" \
--system \
--shell /bin/false \
-g etcd \
etcd
```

2. etcd用のディレクトリを作成します。

```
# mkdir -p /etc/etcd
# chown etcd:etcd /etc/etcd
# mkdir -p /var/lib/etcd
# chown etcd:etcd /var/lib/etcd
```

3. CPUのアーキテクチャを確認します。

```
# uname -m
```

4. x86_64/amd64用のetcdtarballをダウンロードしてインストールします。

```
# ETCD_VER=v3.2.7
# rm -rf /tmp/etcd && mkdir -p /tmp/etcd
# GITHUB_URL=https://github.com/coreos/etcd/releases/download \
DOWNLOAD_URL=${GITHUB_URL}
# curl -L ${DOWNLOAD_URL}/${ETCD_VER}/etcd-${ETCD_VER}-linux-amd64.tar.gz -o \
/tmp/etcd-${ETCD_VER}-linux-amd64.tar.gz # tar xzvf /tmp/etcd-${ETCD_VER}-linux-
amd64.tar.gz \
-C /tmp/etcd --strip-components=1
```

```
# cp /tmp/etcd/etcd /usr/bin/etcd
# cp /tmp/etcd/etcdctl /usr/bin/etcdctl
```

5. 管理ネットワークを介した他のノードによるアクセスを有効にするには、/etc/etcd/etcd.conf.ymlファイルを作成および編集し、initial-cluster、initial-advertise-peer-urls、advertise-client-urls、listen-client-urlsをコントローラーノードの管理IPアドレスに設定します。

```
# vim /etc/etcd/etcd.conf.yml
```

```
name: controller
data-dir: /var/lib/etcd
initial-cluster-state: 'new'
initial-cluster-token: 'etcd-cluster-01'
initial-cluster: controller=http://192.168.1.11:2380
initial-advertise-peer-urls: http://192.168.1.11:2380
advertise-client-urls: http://192.168.1.11:2379
listen-peer-urls: http://0.0.0.0:2380
listen-client-urls: http://192.168.1.11:2379
```

6. /usr/lib/systemd/system/etcd.serviceファイルを作成し編集します。

```
# vim /usr/lib/systemd/system/etcd.service
```

```
[Unit]
After=network.target
Description=etcd - highly-available key value store
[Service]
# Uncomment this on ARM64.
# Environment="ETCD_UNSUPPORTED_ARCH=arm64"
LimitNOFILE=65536
Restart=on-failure
Type=notify
ExecStart=/usr/bin/etcd --config-file /etc/etcd/etcd.conf.yml
User=etcd
[Install]
WantedBy=multi-user.target
```

7. システムを再起動します。

```
# systemctl daemon-reload
```

8. etcdサービスを有効化し開始します。

```
# systemctl enable --now etcd
```

4. サービスのインストール

- 各種サービスをインストールしていきます。

4.1 Keystone

> 前提条件

・ Openstackインストールガイドに記載されている前提条件のインストール手順が完了していること。
・最新版のpython-pyasn1がインストールされていること。

確認コマンド

```
# zypper --no-refresh se -i -t package | grep pyasn1
```

4.1.1 準備

1. データベースアクセスクライアントを使用して、rootユーザーでデータベースサーバーに接続します。

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

2. Keysotne用のデータベースを作成します。

   ```
   MariaDB [(none)]> CREATE DATABASE keystone;
   ```

3. KeyStone用のデータベースに適切なアクセス権を付与する。

   ```
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' \
   IDENTIFIED BY 'KEYSTONE_DBPASS';
   MariaDB [(none)]> GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' \
   IDENTIFIED BY 'KEYSTONE_DBPASS';
   ```

4. データベースクライアントからログアウトします。

4.1.2 コンポーネントのインストールと設定

1. 以下のコマンドを実行し、パッケージをインストールします。

   パッケージ依存に関係する問題が発生したら、依存関係を解決する選択肢を選択します。

   ```
   # zypper install openstack-keystone apache2 apache2-mod_wsgi-python3
   ```

2. /etc/keystone/keystone.conf ファイルを編集し、以下の内容を記述します。

   ```
   # mv /etc/keystone/keystone.conf /etc/keystone/keystone.conf.org
   # vi /etc/keystone/keystone.conf
   ```

   ```
   [database]
   connection = mysql+pymysql://keystone:KEYSTONE_DBPASS@controller/keystone
   [token]
   provider = fernet
   ```

3. Identityサービスのデータベースを投入する。

   ```
   # su -s /bin/sh -c "keystone-manage db_sync" keystone
   ```

4. /etc/keystone/credential-keysフォルダの所有者を以下のコマンドで変更します。

   ```
   # chown keystone /etc/keystone/credential-keys/
   ```

5. Fernetキーリポジトリを初期化します。

```
# keystone-manage fernet_setup --keystone-user keystone --keystone-group keystone
# keystone-manage credential_setup --keystone-user keystone --keystone-group keystone
```

6. Identityサービスをブートストラップする。

```
# keystone-manage bootstrap --bootstrap-password ADMIN_PASS \
--bootstrap-admin-url http://controller:5000/v3/ \
--bootstrap-internal-url http://controller:5000/v3/ \
--bootstrap-public-url http://controller:5000/v3/ \
--bootstrap-region-id RegionOne
```

4.1.3 Apache HTTPサーバーを設定

1. /etc/sysconfig/apache2 ファイルを編集し、APACHE_SERVERNAME オプションを設定して、コントローラノードを参照するように設定します。APACHE_SERVERNAMEの項目がまだ存在しない場合は、追加する必要があります。

```
# vi /etc/sysconfig/apache2
```

```
APACHE_SERVERNAME="controller"
```

2. /etc/apache2/conf.d/wsgi-keystone.conf ファイルを以下の内容で作成します。

```
# vi /etc/apache2/conf.d/wsgi-keystone.conf
```

```
Listen 5000
<VirtualHost *:5000>
WSGIDaemonProcess keystone-public processes=5 threads=1 user=keystone group=keystone
display-name=%{GROUP}
WSGIProcessGroup keystone-public
WSGIScriptAlias / /usr/bin/keystone-wsgi-public
WSGIApplicationGroup %{GLOBAL}
WSGIPassAuthorization On
ErrorLogFormat "%{cu}t %M"
ErrorLog /var/log/apache2/keystone.log
CustomLog /var/log/apache2/keystone_access.log combined
<Directory /usr/bin>
Require all granted
</Directory>
</VirtualHost>
```

3. etc/keystone ディレクトリの所有権を再帰的に変更します。

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

4.1.4 インストールを確定

1. Apache HTTP サービスを開始し、システム起動時に開始するように設定します

```
# systemctl enable apache2.service
# systemctl start apache2.service
```

2. 適切な環境変数を設定し、管理者アカウントを構成します。

   ADMIN_PASSは、コンポーネントのインストールと設定の手順5のkeystone-manage bootstrapコマンドで使用したパスワードに置き換えてください。

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

4.1.5 ドメイン、プロジェクト、ユーザー、ロールの作成

- このガイドのkeystone-manage bootstrapの手順で「デフォルト」ドメインがすでに存在していますが、新しいドメインを作成する正式な方法は次のとおりです。

```
$ openstack domain create --description "An Example Domain" example
```

1. 環境に追加する各サービスに一意のユーザーを含むサービスプロジェクトを使用します。

   以下のコマンドで、サービスプロジェクトを作成します。

```
$ openstack project create --domain default --description "Service Project" service
```

2. 通常のタスクは、非特権プロジェクトとユーザーを使用する必要があります。

例として、このガイドでは、myprojectプロジェクトとmyuserユーザーを作成します。

- [myproject]を作成します。:

```
$ openstack project create --domain default --description "Demo Project" myproject
```

- [myuser]を作成します。

  パスワードを聞かれるので、[DB_PASS]を指定します。

```
$ openstack user create --domain default --password DEMO_PASS myuser
```

- [myrole]を作成します。

```
$ openstack role create myrole
```

- [myproject]プロジェクトに[myrole]ロールを追加し、[myuser]ユーザーを追加します。

```
$ openstack role add --project myproject --user myuser myrole
```

4.1.6 動作確認

1. 一時的な[OS_AUTH_URL]と[OS_PASSWORD]の環境変数を解除します。

```
$ unset OS_AUTH_URL OS_PASSWORD
```

2. adminユーザーとして、認証トークンを要求する。
   パスワードが聞かれるので、[OS_PASSWORD]を入力します。

```
$ openstack --os-auth-url http://controller:5000/v3 \
--os-project-domain-name Default --os-user-domain-name Default \
--os-project-name admin --os-username admin token issue
```

3. 前項で作成したmyuserユーザーとして、認証トークンを要求します。

パスワードが聞かれるので、[DEMO_PASS]を入力します。

```
$ openstack --os-auth-url http://controller:5000/v3 \
--os-project-domain-name Default --os-user-domain-name Default \
--os-project-name myproject --os-username myuser token issue
```

4.1.7 OpenStackクライアント環境スクリプトの作成

スクリプトを作成します。admin と demo のプロジェクトとユーザーのためのクライアント環境スクリプトを作成します。このガイドの今後の部分では、これらのスクリプトを参照して、クライアント操作のための適切な認証情報をロードします。

1. admin-openrc ファイルを作成・編集し、以下の内容を追加します。

   [ADMIN_PASS]を、Identityサービスのadminユーザー用に選択したパスワードに置き換えます。

```
$ vim admin-openrc
```

```
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=ADMIN_PASS
export OS_AUTH_URL=http://controller:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
```

2. demo-openrc ファイルを作成・編集し、以下の内容を追加します。

   [DEMO_PASS]を、Identityサービスでデモユーザー用に選択したパスワードに置き換えます。

```
$ vim demo-openrc
```

```
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=myproject
export OS_USERNAME=myuser
export OS_PASSWORD=DEMO_PASS
export OS_AUTH_URL=http://controller:5000/v3
```

```
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
export PS1='\u@\h \W(keystone)$ '
```

3. スクリプトの使用方法

   クライアントを特定のプロジェクトやユーザーとして実行するには、実行前に関連するクライアント環境スクリプトを読み込むだけでよいです。

   admin-openrc ファイルを読み込んで、Identity サービスの場所、admin プロジェクトとユーザーの認証情報を環境変数に入力します。

   ```
   $ . admin-openrc
   ```

4. 認証トークンを要求する。

   ```
   $ openstack token issue
   ```

4.2 Glance

- Glanceサービスをインストールしていきます。

4.2.1 準備

- Imageサービスをインストール・設定する前に、データベース、サービス認証情報、APIエンドポイントを作成する必要があります。データベースを作成するには、次の手順を実行します。

1. データベースアクセスクライアントを使用して、rootユーザーでデータベースサーバーに接続します。

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

2. Glance用のデータベースを作成します。

```
MariaDB [(none)]> CREATE DATABASE glance;
```

3. Glance用のデータベースへの適切なアクセス権を付与する。 GLANCE_DBPASSを適切なパスワードに置き換えてください。

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'localhost' \
IDENTIFIED BY 'GLANCE_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%' \
IDENTIFIED BY 'GLANCE_DBPASS';
```

4. データベースアクセスクライアントを終了します。

```
MariaDB [(none)]> Exit;
```

5. 管理者専用のCLIコマンドにアクセスするために、管理者の認証情報をソースにします。

```
$ . admin-openrc
```

6. サービスの認証情報を作成するには、以下の手順を実行します。
    - glanceユーザーを作成します。

```
$ openstack user create --domain default --password GLANCE_PASS glance
```

    - glanceユーザーとサービスプロジェクトにadminロールを追加する。

```
$ openstack role add --project service --user glance admin
```

    - Glanceサービスエンティティを作成します。

```
$ openstack service create --name glance \
--description "OpenStack Image" image
```

7. GlanceサービスのAPIエンドポイントを作成します。

```
$ openstack endpoint create --region RegionOne \
image public http://controller:9292
```

```
$ openstack endpoint create --region RegionOne \
image internal http://controller:9292
```

```
$ openstack endpoint create --region RegionOne \
image admin http://controller:9292
```

4.2.2 コンポーネントのインストールと設定

1. Glanceパッケージをインストールします。

```
# zypper install openstack-glance openstack-glance-api
```

2. /etc/glance/glance-api.conf ファイルを編集し、以下を記入します。

```
# mv /etc/glance/glance-api.conf /etc/glance/glance-api.conf.org
```

vim /etc/glance/glance-api.conf

```
[database]
connection = mysql+pymysql://glance:GLANCE_DBPASS@controller/glance
[keystone_authtoken]
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = glance
password = GLANCE_PASS
[paste_deploy]
flavor = keystone
[glance_store]
stores = file,http
default_store = file
filesystem_store_datadir = /var/lib/glance/images/
[oslo_limit]
auth_url = http://controller:5000
auth_type = password
user_domain_id = default
username = MY_SERVICE
system_scope = all
password = MY_PASSWORD
endpoint_id = ENDPOINT_ID
region_name = RegionOne
```

3. イメージサービスを起動し、システム起動時に開始するように設定します。

```
# systemctl enable openstack-glance-api.service
# systemctl start openstack-glance-api.service
```

4.3 Placement

- Placementサービスをインストールしていきます。

> **※この章は5.4 Novaの途中に実施します。**

4.3.1 事前準備

- Placementサービスをインストールし設定する前に、データベース、サービス認証情報、およびAPIエンドポイントを作成する必要があります。データベースを作成するには、次の手順を実行します。

1. データベースアクセスクライアントを使用して、rootユーザーでデータベースサーバーに接続します。

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

2. Placement用のデータベースを作成します。

```
MariaDB [(none)]> CREATE DATABASE placement;
```

3. データベースへの適切なアクセス権を付与する。

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON placement.*TO 'placement'@'localhost' \
IDENTIFIED BY 'PLACEMENT_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON placement.* TO 'placement'@'%' \
IDENTIFIED BY 'PLACEMENT_DBPASS';
```

4. データベースアクセスクライアントを終了します。

```
MariaDB [(none)]> Exit;
```

5. 管理者専用のCLIコマンドにアクセスするために、管理者の認証情報をソースにします。

```
$ . admin-openrc
```

6. 選択したPLACEMENT_PASSを使用して、プレースメントサービスユーザーを作成します。

   パスワードを聞かれた場合は、"PasZw0rd"を入力。

```
$ openstack user create --domain default --password PLACEMENT_PASS placement
```

7. サービスプロジェクトにPlacementユーザーをadminの役割で追加します。

```
$ openstack role add --project service --user placement admin
```

8. サービスカタログにPlacement APIのエントリーを作成します。

```
$ openstack service create --name placement \
--description "Placement API" placement
```

9. Placement APIサービスのエンドポイントを作成します。

```
$ openstack endpoint create --region RegionOne \
placement public http://controller:8778 $ openstack endpoint create --region RegionOne \
placement internal http://controller:8778 $ openstack endpoint create --region RegionOne \
placement admin http://controller:8778
```

4.3.2 コンポーネントのインストールと設定

1. パッケージをインストールします。

```
# zypper install openstack-placement openstack-placement-api
```

2. /etc/placement/placement.conf ファイルを編集し、以下の記入をします。

```
# mv /etc/placement/placement.conf /etc/placement/placement.conf.org
# vi /etc/placement/placement.conf
```

```
[placement_database]
connection = mysql+pymysql://placement:PLACEMENT_DBPASS@controller/placement
[api]
auth_strategy = keystone
[keystone_authtoken]
auth_url = http://controller:5000/v3
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = placement
password = PLACEMENT_PASS
```

3. placementデータベースを作成する。

```
# su -s /bin/sh -c "placement-manage db sync" placement
```

4. Placement APIのApache vhostを有効にします。

```
# cp /etc/apache2/vhosts.d/openstack-placement-api.conf.sample \
/etc/apache2/vhosts.d/openstack-placement-api.conf
```

5. Apache vhostのファイルのポート番号を"8778"に変更します。

```
# vim /etc/apache2/vhosts.d/openstack-placement-api.conf
```

以下は例になります。

```
# OpenStack placement-api Apache2 example configuration
Listen 8778
<Directory /srv/www/openstack-placement-api/>
Options FollowSymLinks MultiViews
AllowOverride None
Require all granted
</Directory>
<VirtualHost *:8778>
WSGIScriptAlias / /srv/www/openstack-placement-api/app.wsgi
WSGIDaemonProcess openstack-placement-api processes=2 threads=1 user=placement
group=placement
WSGIProcessGroup openstack-placement-api
ErrorLog /var/log/placement/placement-api.log
CustomLog /var/log/placement/placement-api.log combined
<Directory /usr/bin>
Require all granted
<Files "placement-api"> <RequireAll> Require all granted Require not env blockAccess
</RequireAll> </Files> </Directory>
</VirtualHost>
Alias /placement /srv/www/openstack-placement-api/app.wsgi
<Location /placement>
SetHandler wsgi-script
Options +ExecCGI
WSGIProcessGroup openstack-placement-api
</Location>
```

6. Apacheをリロードします。

```
# systemctl reload apache2.service
```

4.3.3 動作確認

1. 管理者専用のCLIコマンドにアクセスするために、管理者の認証情報をソースにします。

```
$ . admin-openrc
```

2. 状況確認を行い、異常がないことを確認する。コマンドの出力は、リリースによって異なります。

```
$ sudo placement-status upgrade check
```

3. placement APIに対していくつかのコマンドを実行します。

    - プラグイン「osc-placement」をインストールします。

```
$ pip3 install osc-placement
```

- 利用可能なリソースクラスと特性をリストアップします。

```
$ openstack --os-placement-api-version 1.2 resource class list --sort-column name
$ openstack --os-placement-api-version 1.6 trait list --sort-column name
```

4.4 Nova

- コントローラノードにComputeサービス（コードネーム：nova）をインストールし、設定する方法を説明します。

4.4.1 事前準備

- Computeサービスをインストールし、設定する前に、データベース、サービス認証情報、およびAPIエンドポイントを作成する必要があります。

1. データベースを作成するには、次の手順を実行します。
    - データベースアクセスクライアントを使用して、rootユーザーでデータベースサーバーに接続します。

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

    - nova_api、nova、nova_cell0 データベースを作成します。

      ```
      MariaDB [(none)]> CREATE DATABASE nova_api;
      MariaDB [(none)]> CREATE DATABASE nova;
      MariaDB [(none)]> CREATE DATABASE nova_cell0;
      ```

    - データベースへの適切なアクセス権を付与する。NOVA_DBPASSを適切なパスワードに置き換えてください。

      ```
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'localhost' \
      IDENTIFIED BY 'NOVA_DBPASS';
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_api.* TO 'nova'@'%' \
      IDENTIFIED BY 'NOVA_DBPASS';
      ```

      ```
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost' \
      IDENTIFIED BY 'NOVA_DBPASS';
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' \
      IDENTIFIED BY 'NOVA_DBPASS';
      ```

      ```
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_cell0.* TO 'nova'@'localhost' \
      IDENTIFIED BY 'NOVA_DBPASS';
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_cell0.* TO 'nova'@'%' \
      IDENTIFIED BY 'NOVA_DBPASS';
      ```

    - データベースアクセスクライアントを終了します。

2. 管理者専用のCLIコマンドにアクセスするために、管理者の認証情報をソースにします

   ```
   $ . admin-openrc
   ```

3. Computeサービスのクレデンシャルを作成します。
    - novaユーザーを作成します。

      ```
      $ openstack user create --domain default --password NOVA_PASS nova
      ```

- novaユーザーにadminロールを追加します。

```
$ openstack role add --project service --user nova admin
```

- novaサービスエンティティを作成します。

```
$ openstack service create --name nova \
--description "OpenStack Compute" compute
```

4. Compute APIのサービスエンドポイントを作成します。

```
$ openstack endpoint create --region RegionOne \
compute public http://controller:8774/v2.1
```

```
$ openstack endpoint create --region RegionOne \
compute internal http://controller:8774/v2.1
```

```
$ openstack endpoint create --region RegionOne \
compute admin http://controller:8774/v2.1
```

5. Placementサービスをインストールし、ユーザーとエンドポイントを設定します。

4.4.2 コントローラーノードへのコンポーネントのインストールと設定

1. パッケージをインストールします。

```
# zypper install \
openstack-nova-api \
openstack-nova-scheduler \
openstack-nova-conductor \
openstack-nova-novncproxy \
iptables
```

2. etc/nova/nova.conf ファイルを編集し、以下の操作を行います。

```
# mv /etc/nova/nova.conf /etc/nova/nova.conf.org
```

vim /etc/nova/nova.conf

```
[DEFAULT]
enabled_apis = osapi_compute,metadata
transport_url = rabbit://openstack:RABBIT_PASS@controller:5672
my_ip = 192.168.1.11
[api_database]
connection = mysql+pymysql://nova:NOVA_DBPASS@controller/nova_api
[database]
connection = mysql+pymysql://nova:NOVA_DBPASS@controller/nova
[api]
auth_strategy = keystone
[keystone_authtoken]
www_authenticate_uri = http://controller:5000/
auth_url = http://controller:5000/
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = nova
password = NOVA_PASS
[vnc]
enabled = true
server_listen = $my_ip
server_proxyclient_address = $my_ip
[glance]
api_servers = http://controller:9292
[oslo_concurrency]
lock_path = /var/run/nova
[placement]
region_name = RegionOne
project_domain_name = Default
project_name = service
auth_type = password
user_domain_name = Default
auth_url = http://controller:5000/v3
username = placement
password = PLACEMENT_PASS
```

3. nova-apiのデータベースにデータを入れる。

```
# su -s /bin/sh -c "nova-manage api_db sync" nova
```

4. cell0 データベースを登録する。

```
# su -s /bin/sh -c "nova-manage cell_v2 map_cell0" nova
```

5. cell1 セルを作成します。

```
# su -s /bin/sh -c "nova-manage cell_v2 create_cell --name=cell1 --verbose" nova
```

```
以下のメッセージが出力されますが無視して大丈夫です。正常な出力です。
--transport-url not provided in the command line, using the value [DEFAULT]/transport_url from the configuration file
--database_connection not provided in the command line, using the value [database]/connection from the configuration file
```

6. novaデータベースにデータを登録する。

```
# su -s /bin/sh -c "nova-manage db sync" nova
```

7. nova cell0とcell1が正しく登録されていることを確認する。

```
# su -s /bin/sh -c "nova-manage cell_v2 list_cells" nova
```

8. Computeサービスを起動し、システム起動時に起動するように設定します。

```
# systemctl enable \
openstack-nova-api.service \
openstack-nova-scheduler.service \
openstack-nova-conductor.service \
openstack-nova-novncproxy.service
```

```
# systemctl start \
openstack-nova-api.service \
openstack-nova-scheduler.service \
openstack-nova-conductor.service \
openstack-nova-novncproxy.service
```

9. コンピュートノードがコントローラーノードへアクセスをするために、コントローラーノードのファイアウォールのポートを以下のコマンドで開けます。

```
# firewall-cmd --add-port=5000/tcp
# firewall-cmd --add-port=5672/tcp
```

4.4.3 コンピュートノードへのコンポーネントのインストールと設定

- 事前準備

  以下のリポジトリを追加します。

```
# zypper addrepo \
https://download.opensuse.org/repositories/home:cabelo/15.3/home:cabelo.repo
# zypper refresh
```

1. パッケージをインストールします。

```
# zypper install openstack-nova-compute genisoimage qemu-kvm libvirt
```

2. /etc/nova/nova.conf ファイルを編集し、以下の操作を行います。

```
# mv /etc/nova/nova.conf /etc/nova/nova.conf.org
```

vim /etc/nova/nova.conf

```
[DEFAULT]
enabled_apis = osapi_compute,metadata
compute_driver = libvirt.LibvirtDriver
transport_url = rabbit://openstack:RABBIT_PASS@controller
my_ip = 192.168.1.31
[api]
auth_strategy = keystone
[keystone_authtoken]
www_authenticate_uri = http://controller:5000/
auth_url = http://controller:5000/
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = nova
password = NOVA_PASS
[vnc]
enabled = true
server_listen = 0.0.0.0
server_proxyclient_address = $my_ip
novncproxy_base_url = http://controller:6080/vnc_auto.html
[api_database]
connection = mysql+pymysql://nova:NOVA_DBPASS@controller/nova_api
[database]
connection = mysql+pymysql://nova:NOVA_DBPASS@controller/nova
[glance]
api_servers = http://controller:9292
[oslo_concurrency]
lock_path = /var/run/nova
[placement]
region_name = RegionOne
project_domain_name = Default
project_name = service
auth_type = password
user_domain_name = Default
auth_url = http://controller:5000/v3
```

```
username = placement
password = PLACEMENT_PASS
```

3. カーネルモジュールnbdがロードされていることを確認する。

```
# modprobe nbd
```

4. /etc/modules-load.d/nbd.conf ファイルに nbd を追加して、起動時に必ずモジュールをロードするようにします。

```
# vim /etc/modules-load.d/nbd.conf
```

```
nbd
```

4.4.4 インストールの確認

1. 計算ノードが仮想マシンのハードウェアアクセラレーションをサポートしているかどうかを確認します。

   このコマンドで1以上の値が返された場合、お使いの計算ノードはハードウェア・アクセラレーションをサポートしており、通常、追加の設定は必要ありません。

   このコマンドが 0 の値を返す場合、ご使用の計算ノードはハードウェアアクセラレーションをサポートしておらず、 libvirt が KVM の代わりに QEMU を使用するように設定する必要があります。

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

   - /etc/nova/nova.conf ファイル内に以下を追記します。

```
# vim /etc/nova/nova.conf
```

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

2. Computeサービスをその依存関係を含めて起動し、システム起動時に自動的に起動するように設定します。

```
# systemctl enable libvirtd.service openstack-nova-compute.service
# systemctl start libvirtd.service openstack-nova-compute.service
```

> nova-compute サービスの起動に失敗した場合は、/var/log/nova/nova-compute.log を確認してください。AMQP server on controller:5672 is unreachableというエラーメッセージは、コントローラノード上のファイアウォールがポート5672へのアクセスを妨げている可能性があります。コントローラノードでポート5672を開くようにファイアウォールを設定し、計算ノードでnova-computeサービスを再起動します。

4.4.5 セルデータベースにコンピュートノードを追加する

- コントローラノードで以下のコマンドを実行します。

1. 管理者権限をソースとして、管理者のみのCLIコマンドを有効にし、データベースにコンピュートホストが存在することを確認します。

> $. admin-openrc

> $ openstack compute service list --service nova-compute

2. コンピュートホストを発見する。

> # su -s /bin/sh -c "nova-manage cell_v2 discover_hosts --verbose" nova

> 新しいコンピュートノードを追加する際には、コントローラノード上でnova-manage cell_v2 discover_hostsを実行して、それらの新しいコンピュートノードを登録しなければなりません。また、/etc/nova/nova.confで適切な間隔を設定することも可能です。

[scheduler]
discover_hosts_in_cells_interval = 300

4.4.6 インストールの確認

- Computeサービスの動作を確認します。

1. 管理者専用のCLIコマンドにアクセスするために、管理者の認証情報をソースにします。

> $. admin-openrc

2. 各プロセスの起動と登録が正常に行われたことを確認するために、サービスコンポーネントをリストアップします。

> $ openstack compute service list

> この出力は、コントローラノードで2つのサービスコンポーネントが有効で、コンピュートノードで1つのサービスコンポーネントが有効であることを示します。

3. Identity サービスの API エンドポイントをリストアップし、Identity サービスとの接続を確認します。

   エンドポイントリストは、OpenStackコンポーネントのインストール状況によって異なる場合があります。

> $ openstack catalog list

> この出力に含まれる警告は無視すること。

4. Imageサービスの画像を一覧表示し、Imageサービスとの接続を確認します。

> $ openstack image list

5. セルとPlacement APIが正常に動作していること、その他必要な前提条件が整っていることを確認します。

> # nova-status upgrade check

4.5 Neutron

- 初めにコントローラーノードに対してインストールをしていきます。

4.5.1 コントローラーノード

4.5.1.1 事前準備

1. データベースを作成するには、次の手順を実行します。
    - データベースアクセスクライアントを使用して、rootユーザーでデータベースサーバーに接続します。

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

    - neutronデータベースを作成します。

      ```
      MariaDB [(none)]> CREATE DATABASE neutron;
      ```

    - NEUTRON_DBPASSを適切なパスワードに置き換えて、中性子データベースへの適切なアクセスを許可してください。

      ```
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.*TO 'neutron'@'localhost' \
      IDENTIFIED BY 'NEUTRON_DBPASS';
      MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO 'neutron'@'%' \
      IDENTIFIED BY 'NEUTRON_DBPASS';
      ```

    - データベースアクセスクライアントを終了します。
2. 管理者専用のCLIコマンドにアクセスするために、管理者の認証情報をソースにします。

   ```
   $ . admin-openrc
   ```

3. サービスの認証情報を作成するには、以下の手順を実行します。
    - neutronユーザーを作成します。

      ```
      $ openstack user create --domain default --password NEUTRON_PASS neutron
      ```

    - neutronユーザーにadminロールを追加します。

      ```
      $ openstack role add --project service --user neutron admin
      ```

    - neutronサービスエンティティを作成します。

      ```
      $ openstack service create --name neutron \
      --description "OpenStack Networking" network
      ```

4. NetworkingサービスのAPIエンドポイントを作成します。

   ```
   $ openstack endpoint create --region RegionOne \
   network public http://controller:9696
   ```

   ```
   $ openstack endpoint create --region RegionOne \
   network internal http://controller:9696
   ```

   ```
   $ openstack endpoint create --region RegionOne \
   network admin http://controller:9696
   ```

4.5.1.2 ネットワーク構築オプション2：セルフサービス型ネットワークのインストール

1. コンポーネントをインストールします。

```
# zypper install --no-recommends openstack-neutron \
openstack-neutron-server openstack-neutron-linuxbridge-agent \
openstack-neutron-l3-agent openstack-neutron-dhcp-agent \
openstack-neutron-metadata-agent bridge-utils dnsmasq
```

2. /etc/neutron/neutron.conf ファイルを編集し、以下を記述します。

```
# mv /etc/neutron/neutron.conf /etc/neutron/neutron.conf.org
```

vim /etc/neutron/neutron.conf

```
[database]
connection = mysql+pymysql://neutron:NEUTRON_DBPASS@controller/neutron
[DEFAULT]
core_plugin = ml2
service_plugins = router
allow_overlapping_ips = true
transport_url = rabbit://openstack:RABBIT_PASS@controller
auth_strategy = keystone
notify_nova_on_port_status_changes = true
notify_nova_on_port_data_changes = true
[keystone_authtoken]
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = NEUTRON_PASS
[nova]
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = nova
passwo rd = NOVA_PASS
[oslo_concurrency]
lock_path = /var/lib/neutron/tmp
```

4.5.1.3 モジュールレイヤー2（ML2）プラグインを設定する

- ML2プラグインは、Linuxのブリッジ機構を利用して、インスタンスのレイヤ2（ブリッジとスイッチング）仮想ネットワーク基盤を構築します。

1. /etc/neutron/plugins/ml2/ml2_conf.ini ファイルを編集し、以下の操作を行います。

```
# mv /etc/neutron/plugins/ml2/ml2_conf.ini \
/etc/neutron/plugins/ml2/ml2_conf.ini.org
```

```
# vim /etc/neutron/plugins/ml2/ml2_conf.ini
```

```
[ml2]
type_drivers = flat,vlan,vxlan
tenant_network_types = vxlan
mechanism_drivers = linuxbridge,l2population
extension_drivers = port_security
[ml2_type_flat]
flat_networks = provider
[ml2_type_vxlan]
vni_ranges = 1:1000
[securitygroup]
enable_ipset = true
```

4.5.1.4 Linuxブリッジエージェントの設定

- Linuxブリッジエージェントは、インスタンスのレイヤ2（ブリッジとスイッチング）仮想ネットワークインフラを構築し、セキュリティを処理します。

1. /etc/neutron/plugins/ml2/linuxbridge_agent.ini ファイルを編集し、以下の操作を行います。

```
# mv /etc/neutron/plugins/ml2/linuxbridge_agent.ini \
```

/etc/neutron/plugins/ml2/linuxbridge_agent.ini.org

```
# vim /etc/neutron/plugins/ml2/linuxbridge_agent.ini
```

   - プロバイダ仮想ネットワークをプロバイダ物理ネットワークインターフェイスにマッピングします。

     PROVIDER_INTERFACE_NAMEは、基盤となるプロバイダの物理ネットワーク・インターフェイスの名前に置き換えてください。詳細については、「ホスト・ネットワーク」を参照してください。

```
[linux_bridge]
```

```
physical_interface_mappings = provider:<PROVIDER_INTERFACE_NAME>
```

- VXLANオーバーレイネットワークを有効にし、オーバーレイネットワークを扱う物理ネットワークインタフェースのIPアドレスを設定し、レイヤ2ポピュレーションを有効にします。

  OVERLAY_INTERFACE_IP_ADDRESS を、オーバーレイ・ネットワークを処理する基礎となる物理ネットワーク・インターフェイスの IP アドレスに置き換えてください。このアーキテクチャ例では、管理インターフェイスを使用して他のノードにトラフィックをトンネリングしています。したがって、OVERLAY_INTERFACE_IP_ADDRESSをコントローラノードの管理IPアドレスに置き換えます。詳細は、「ホストネットワーキング」を参照してください。

  ```
  [vxlan]
  enable_vxlan = true
  local_ip = <OVERLAY_INTERFACE_IP_ADDRESS>
  l2_population = true
  ```

- セキュリティグループを有効にし、Linux bridge iptables firewall driverを設定します。

  ```
  [securitygroup]
  # ...
  enable_security_group = true
  firewall_driver = neutron.agent.linux.iptables_firewall.IptablesFirewallDriver
  ```

2. 以下のコマンドで、必要なモジュールがインストールされているか確認してください。

   ```
   # lsmod|grep br_netfilter
   ```

   ない場合は、以下のコマンドでインストールとモジュールの永続化をしてください。

   ある場合は、当該コマンドを実行せず次の"sysctl -w"コマンドを実行してください。

   ```
   # modprobe br_netfilter
   # sh -c 'echo "br_netfilter" > /etc/modules-load.d/br_netfilter.conf'
   ```

   最後に以下のコマンドで値を1に設定してください。

   ```
   # sysctl -w net.bridge.bridge-nf-call-iptables=1
   # sysctl -w net.bridge.bridge-nf-call-ip6tables=1
   ```

4.5.1.5 レイヤー3エージェントを設定する

- レイヤー3（L3）エージェントは、セルフサービスの仮想ネットワークにルーティングとNATのサービスを提供します。

1. /etc/neutron/l3_agent.ini ファイルを編集し、以下の操作を行います。

   ```
   # mv /etc/neutron/l3_agent.ini /etc/neutron/l3_agent.ini.org
   ```

   ```
   # vim /etc/neutron/l3_agent.ini
   ```

   - Linuxブリッジインターフェースドライバを設定する。

     ```
     [DEFAULT]
     interface_driver = linuxbridge
     ```

4.5.1.6 DHCPエージェントを設定する

- DHCPエージェントは、仮想ネットワークにDHCPサービスを提供します。

1. /etc/neutron/dhcp_agent.ini ファイルを編集し、以下の操作を行います。

```
# mv /etc/neutron/dhcp_agent.ini /etc/neutron/dhcp_agent.ini.org
```

```
# vim /etc/neutron/dhcp_agent.ini
```

   - [DEFAULT]セクションで、Linuxブリッジインターフェースドライバ、Dnsmasq DHCPドライバを設定し、プロバイダネットワーク上のインスタンスがネットワーク経由でメタデータにアクセスできるように、分離メタデータを有効にします。

```
[DEFAULT]
interface_driver = linuxbridge
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = true
```

4.5.1.7 メタデータエージェントを設定する

- メタデータエージェントは、インスタンスにクレデンシャルなどの設定情報を提供する。

1. /etc/neutron/metadata_agent.ini ファイルを編集し、以下の操作を行います。METADATA_SECRETをメタデータプロキシに適したシークレットに置き換える。

```
# mv /etc/neutron/metadata_agent.ini /etc/neutron/metadata_agent.ini.org
```

```
# vim /etc/neutron/metadata_agent.ini
```

   - メタデータホストと共有シークレットを設定します。

```
[DEFAULT]
nova_metadata_host = controller
metadata_proxy_shared_secret = METADATA_SECRET
```

4.5.1.8 ComputeサービスがNetworkingサービスを使用するように設定する

1. /etc/nova/nova.conf ファイルを編集し、以下の操作を行います。

```
# vim /etc/nova/nova.conf
```

   - アクセスパラメータを設定し、メタデータプロキシを有効化し、シークレットを設定します。

NEUTRON_PASSをIdentityサービスでNeutronユーザー用に選んだパスワードに置き換えます。

METADATA_SECRETを、メタデータプロキシ用に選択したシークレットに置き換えてください。

```
[neutron]
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = NEUTRON_PASS
service_metadata_proxy = true
metadata_proxy_shared_secret = METADATA_SECRET
```

4.5.1.9 インストールを確定する

- SLESはデフォルトでapparmorを有効にし、dnsmasqを制限しています。apparmorを完全に無効にするか、dnsmasqプロファイルのみを無効にする必要があります。

```
# ln -s /etc/apparmor.d/usr.sbin.dnsmasq /etc/apparmor.d/disable/
# systemctl restart apparmor
```

1. Compute APIサービスを再起動します。

```
# systemctl restart openstack-nova-api.service
```

2. Networkingサービスを起動し、システム起動時に起動するように設定します。

```
# systemctl enable openstack-neutron.service \
openstack-neutron-linuxbridge-agent.service \
openstack-neutron-dhcp-agent.service \
openstack-neutron-metadata-agent.service
# systemctl start openstack-neutron.service \
openstack-neutron-linuxbridge-agent.service \
openstack-neutron-dhcp-agent.service \
openstack-neutron-metadata-agent.service
```

3. ネットワークオプション2の場合、レイヤー3サービスも有効にして起動します。

```
# systemctl enable openstack-neutron-l3-agent.service
# systemctl start openstack-neutron-l3-agent.service
```

4.5.2 コンピュートノード

4.5.2.1 コンポーネントをインストールする

```
# zypper install --no-recommends \
openstack-neutron-linuxbridge-agent bridge-utils
```

4.5.2.2 共通のコンポーネントを設定する

- Networkingの共通コンポーネント構成は、認証機構、メッセージキュー、プラグインである。

1. /etc/neutron/neutron.conf ファイルを編集し、以下の操作を行います。

```
# mv /etc/neutron/neutron.conf /etc/neutron/neutron.conf.org
# vim /etc/neutron/neutron.conf
```

   - [database] セクションでは、computeノードは直接データベースにアクセスしないため、接続オプションをコメントアウトします。
   - RabbitMQメッセージキューアクセスを設定します。

```
[DEFAULT]
transport_url = rabbit://openstack:RABBIT_PASS@controller
```

   - Identity サービスへのアクセスを設定します。NEUTRON_PASSをIdentityサービスで中性子ユーザー用に選んだパスワードに置き換えます。

```
[DEFAULT]
auth_strategy = keystone
[keystone_authtoken]
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = NEUTRON_PASS
```

   - ロックパスを設定する。

```
[oslo_concurrency]
lock_path = /var/lib/neutron/tmp
```

4.5.2.3 ネットワーク構築オプション2：セルフサービス型ネットワーク

- コンピュート・ノードでネットワーキング・コンポーネントを設定します。
- Linux ブリッジエージェントは、インスタンスのレイヤ2（ブリッジとスイッチング）の仮想ネットワークインフラを構築し、セキュリティグループを処理します。

1. /etc/neutron/plugins/ml2/linuxbridge_agent.ini ファイルを編集し、以下の操作を行います。

```
# vim /etc/neutron/plugins/ml2/linuxbridge_agent.ini
```

   - プロバイダ仮想ネットワークをプロバイダ物理ネットワークインターフェイスにマッピングします。PROVIDER_INTERFACE_NAMEは、基盤となるプロバイダの物理ネットワーク・インターフェイスの名前に置き換えてください。

```
[linux_bridge]
physical_interface_mappings = provider:<PROVIDER_INTERFACE_NAME>
```

   - VXLANオーバーレイネットワークを有効にし、オーバーレイネットワークを扱う物理ネットワークインタフェースのIPアドレスを設定し、レイヤ2ポピュレーションを有効にします。

OVERLAY_INTERFACE_IP_ADDRESS を、オーバーレイ・ネットワークを処理する基礎となる物理ネットワーク・インターフェイスの IP アドレスに置き換えてください。このアーキテクチャ例では、管理インターフェイスを使用して他のノードにトラフィックをトンネリングしています。したがって、OVERLAY_INTERFACE_IP_ADDRESSをコンピュート・ノードの管理IPアドレスに置き換えます。

```
[vxlan]
enable_vxlan = true
local_ip = <OVERLAY_INTERFACE_IP_ADDRESS>
l2_population = true
```

   - セキュリティグループを有効にし、Linuxブリッジのiptablesファイアウォールドライバーを設定します。

```
[securitygroup]
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.IptablesFirewallDriver
```

   - 以下のsysctlの値がすべて1に設定されていることを確認し、Linuxオペレーティングシステムのカーネルがネットワークブリッジフィルタをサポートしていることを確認してください。

ネットワークブリッジのサポートを有効にするには、通常、br_netfilter カーネルモジュールがロードされる必要があります。このモジュールを有効にするための詳細については、お使いのオペレーティングシステムのドキュメントを確認してください。

[参考]

以下のコマンドでモジュールを有効化させます

```
modprobe br_netfilter
net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-ip6tables
```

4.5.2.4 ComputeサービスがNetworkingサービスを使用するように設定する

1. /etc/nova/nova.conf ファイルを編集し、以下の操作を行います。

   NEUTRON_PASSをIdentityサービスで中性子ユーザー用に選んだパスワードに置き換えます。

```
[neutron]
auth_url = http://controller:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = NEUTRON_PASS
```

4.5.2.5 インストールを確定する

1. Networkingサービスの初期化スクリプトは、/etc/sysconfig/neutronファイル内の変数NEUTRON_PLUGIN_CONFがML2プラグイン設定ファイルを参照することを想定しているため、/etc/sysconfig/neutronファイルに以下の内容が記述されていることを確認してください。/etc/sysconfig/neutron ファイルが以下を含んでいることを確認してください。

```
# vim /etc/sysconfig/neutron
NEUTRON_PLUGIN_CONF="/etc/neutron/plugins/ml2/ml2_conf.ini"
```

2. Computeサービスを再起動します。

```
# systemctl restart openstack-nova-compute.service
```

3. Linux Bridgeエージェントを起動し、システム起動時に起動するように設定します。

```
# systemctl enable openstack-neutron-linuxbridge-agent.service
# systemctl start openstack-neutron-linuxbridge-agent.service
```

4.5.2.6 動作確認

- コントローラノードで以下のコマンドを実行します。

1. 管理者専用のCLIコマンドにアクセスするために、管理者の認証情報をソースにします。

```
$ . admin-openrc
```

2. neutron-serverプロセスの正常な起動を確認するために、ロードされた拡張機能をリストアップします。

```
$ openstack extension list --network
```

3. neutron-sanity-checkコマンドラインクライアントを使用して、ネットワークのさらなるテストを実行することができます。

出力は、コントローラノードに4つのエージェント、各コンピュートノードに1つのエージェントがあることを示しているはずです。

```
$ openstack network agent list
```

4.6 Dashboard

4.6.1 コンポーネントのインストールと設定

1. コンポーネントのインストールと設定をする。

```
# zypper install openstack-dashboard
```

2. Webサーバーを設定する。

```
# cp /etc/apache2/conf.d/openstack-dashboard.conf.sample \
/etc/apache2/conf.d/openstack-dashboard.conf
# a2enmod rewrite
```

3. /srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py ファイルを編集し、次の操作を行います。

```
# vim /srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py
```

- コントローラノードでOpenStackサービスを使用するようにダッシュボードを設定します。

```
OPENSTACK_HOST = "controller"
```

- ホストがダッシュボードにアクセスできるようにします。

  ALLOWED_HOSTS に ['*'] を指定すると、すべてのホストを受け入れることもできます。これは開発作業には便利かもしれませんが、安全でない可能性があるので、実稼働環境 では使わないで下さい。

```
ALLOWED_HOSTS = ['one.example.com', 'two.example.com']
```

- memcachedセッションストレージサービスを設定します。その他のセッションストレージの構成はコメントアウトしてください。

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

- Identity API バージョン 3 を有効にする。

```
OPENSTACK_KEYSTONE_URL = "http://controller:5000/v3"
```

- ドメインのサポートを有効にする。

```
OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT = True
```

- APIのバージョンを設定する。

```
OPENSTACK_API_VERSIONS = {
"identity": 3,
"image": 2,
"volume": 3,
}
```

- ダッシュボードから作成するユーザーのデフォルトドメインとしてDefaultを設定します。

```
OPENSTACK_KEYSTONE_DEFAULT_DOMAIN = "Default"
```

- ダッシュボードで作成したユーザーのデフォルトロールとして、ユーザーを設定します。

```
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "user"
```

- オプションで、タイムゾーンを設定する。

TIME_ZONEを適切なタイムゾーン識別子で置き換えてください。

```
TIME_ZONE = "TIME_ZONE"
```

4.6.2 インストールを確定する

1. Webサーバーとセッションストレージサービスを再起動します。

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

4.6.3 動作確認

1. 以下のURLにアクセスする。

```
<<http://<controller_IPAddress>>>
```

2. ログイン画面が表示されたら、こちらで作成したユーザー名とパスワードでログインします。

   adminでログインする場合のパスワードは[ADMIN_PASS]で指定した値になります。

3. トップページが表示されたらログイン成功です。

4.7 Cinder

4.7.1 コントローラノードのインストールと設定

1. データベースアクセスクライアントを使用して、root ユーザーとしてデータベースサーバーに接続します。

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

2. cinderデータベースを作成します。

```
\MariaDB [(none)]> CREATE DATABASE cinder;
```

3. cinderデータベースへの適切なアクセス権を付与する。
   CINDER_DBPASSを適切なパスワードに置き換えてください。

```
MariaDB [(none)]> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'localhost' \
IDENTIFIED BY 'CINDER_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%' \
IDENTIFIED BY 'CINDER_DBPASS';
```

4. データベースアクセスクライアントを終了します。

5. コントローラーノードで、以下のコマンドを実行して、ユーザー専用のCLIコマンドにアクセスします。

```
$ . admin-openrc
```

6. cinderユーザーを作成します。

```
$ openstack user create --domain default --password-prompt cinder
```

7. cinderユーザーにadminロールを追加します。

```
$ openstack role add --project service --user cinder admin
```

8. cinder v2およびcinder v3サービスエンティティを作成します。

```
$ openstack service create --name cinderv2 \
--description "OpenStack Block Storage" volumev2
```

```
$ openstack service create --name cinderv3 \
--description "OpenStack Block Storage" volumev3
```

9. Block StorageサービスのAPIエンドポイントを作成します。

```
$ openstack endpoint create --region RegionOne \
volumev2 public http://controller:8776/v2/%\(project_id\)s
```

```
$ openstack endpoint create --region RegionOne \
volumev2 internal http://controller:8776/v2/%\(project_id\)s
```

```
$ openstack endpoint create --region RegionOne \
volumev2 admin http://controller:8776/v2/%\(project_id\)s
```

```
$ openstack endpoint create --region RegionOne \
volumev3 public http://controller:8776/v3/%\(project_id\)s
```

```
$ openstack endpoint create --region RegionOne \
volumev3 internal http://controller:8776/v3/%\(project_id\)s
```

```
$ openstack endpoint create --region RegionOne \
volumev3 admin http://controller:8776/v3/%\(project_id\)s
```

4.7.2 コンポーネントのインストールと設定

1. パッケージをインストールします。

```
# zypper install openstack-cinder-api openstack-cinder-scheduler
```

2. /etc/cinder/cinder.conf ファイルを編集し、以下の操作を完了します。

```
# mv /etc/cinder/cinder.conf /etc/cinder/cinder.conf.org
```

```
# vim /etc/cinder/cinder.conf
```

```
[database]
connection = mysql+pymysql://cinder:CINDER_DBPASS@controller/cinder
[DEFAULT]
transport_url = rabbit://openstack:RABBIT_PASS@controller
auth_strategy = keystone
my_ip = 192.168.1.31
[keystone_authtoken]
www_authenticate_uri = http://controller:5000
```

```
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = cinder
password = CINDER_PASS
[oslo_concurrency]
lock_path = /var/lib/cinder/tmp
```

4.7.3 ブロックストレージを使用するためのComputeの設定

1. /etc/nova/nova.conf ファイルを編集し、以下を追加してください。

```
# vim /etc/nova/nova.conf
```

```
[cinder]
os_region_name = RegionOne
```

4.7.4 インストールの確認

1. Compute APIサービスを再起動します。

```
# systemctl restart openstack-nova-api.service
```

2. Block Storageサービスを起動し、システム起動時に起動するように設定します。

```
# systemctl enable openstack-cinder-api.service openstack-cinder-scheduler.service
# systemctl start openstack-cinder-api.service openstack-cinder-scheduler.service
```

4.7.5 ストレージノードのインストールと設定

4.7.5.1 前提条件

ストレージノードにブロックストレージサービスをインストールし、設定する前に、ストレージデバイスを準備する必要があります。

1. サポートするユーティリティパッケージをインストールします。

2. LVM パッケージをインストールします。
   ディストリビューションによっては、デフォルトでLVMを搭載しているものもあります。

```
# zypper install lvm2
```

3. (オプション)QCOW2やVMDKのような非ローイメージタイプを使用する場合は、QEMUパッケージをインストールしてください。

```
# zypper install qemu
```

4. LVM物理ボリューム/dev/sdbを作成します。

```
# pvcreate /dev/sdb
```

5. LVMボリュームグループcinder-volumesを作成します。
   ブロックストレージサービスは、このボリュームグループに論理ボリュームを作成します。

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

6. ブロックストレージのボリュームにアクセスできるのは、インスタンスのみです。しかし、基盤となるオペレーティング・システムは、ボリュームに関連するデバイスを管理します。デフォルトでは、LVM ボリューム スキャン ツールが /dev ディレクトリをスキャンして、ボリュームを含むブロック ストレージ デバイスを探します。プロジェクトでボリュームに LVM を使用している場合、スキャンツールはこれらのボリュームを検出してキャッシュしようとするため、基盤となるオペレーティングシステムとプロジェクトのボリュームの両方でさまざまな問題が発生する可能性があります。cinder-volumes ボリュームグループを含むデバイスのみをスキャンするように、LVM を再設定する必要があります。/etc/lvm/lvm.conf ファイルを編集して、次の操作を完了します。
   devicesセクションに、/dev/sdbデバイスを受け入れ、それ以外のデバイスを拒否するフィルタを追加します。
   フィルター配列の各項目は、acceptならa、rejectならrで始まり、デバイス名の正規表現が含まれています。残りのデバイスを拒否するには、配列の最後を r/.*/ で終わらせる必要があります。vgs -vvvv コマンドを使用すると、フィルタをテストすることができます。

```
# vim /etc/lvm/lvm.conf
```

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

```
[警告] ストレージノードがオペレーティングシステムディスクで LVM を使用する場合、関連するデバイスもフィルターに追加する必要があります。例えば、/dev/sda デバイスにオペレーティングシステムが含まれている場合、以下のようになります。
filter = [ "a/sda/", "a/sdb/", "r/./"].
同様に、計算ノードがオペレーティング・システム・ディスク上でLVMを使用する場合、 それらのノード上の/etc/lvm/lvm.confファイル内のフィルターも、オペレーティング・システム・ディスクのみを含むように変更する必要があります。例えば、/dev/sdaデバイスにオペレーティングシステムが含まれている場合、以下のようになります。
filter = [ "a/sda/", "r/./" ]
```

4.7.5.2 コンポーネントのインストールと設定

1. パッケージをインストールします。

```
# zypper install openstack-cinder-volume tgt
```

2. /etc/cinder/cinder.conf ファイルを編集し、以下の操作を完了します。

```
# mv /etc/cinder/cinder.conf /etc/cinder/cinder.conf.org
# vim /etc/cinder/cinder.conf
```

```
[database]
connection = mysql+pymysql://cinder:CINDER_DBPASS@controller/cinder
[DEFAULT]
transport_url = rabbit://openstack:RABBIT_PASS@controller
auth_strategy = keystone
my_ip = 192.168.1.41
enabled_backends = lvm
glance_api_servers = http://controller:9292
[keystone_authtoken]
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = cinder
password = CINDER_PASS
[lvm]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_group = cinder-volumes
target_protocol = iscsi
target_helper = tgtadm
[oslo_concurrency]
lock_path = /var/lib/cinder/tmp
```

3. /etc/tgt/conf.d/cinder.conf ファイルを編集し、以下を記述します。

```
# vim /etc/tgt/conf.d/cinder.conf
```

```
include /var/lib/cinder/volumes/*
```

4.7.5.3 インストールを確定する

1. Block Storageボリュームサービスをその依存関係を含めて起動し、システム起動時に起動するように設定します。

```
# systemctl enable openstack-cinder-volume.service tgtd.service
# systemctl start openstack-cinder-volume.service tgtd.service
```

4.8 その他サービス

5. 環境の動作確認

5.1 仮想ネットワークの作成

- この章では作成した仮想化基盤の上に仮想ネットワークを作成していきます。

5.1.1 プロバイダーネットワークの作成

1. コントローラーノードで、以下のコマンドを実行して、ユーザー専用のCLIコマンドにアクセスします。

```
$ . admin-openrc
```

2. 以下のコマンドでネットワークを作成します。

```
$ openstack network create --share --external \
--provider-physical-network provider \
--provider-network-type flat provider
```

3. 以下のコマンドを参考にし、サブネットを作成します。

```
$ openstack subnet create --network provider \
--allocation-pool start=<START_IP_ADDRESS>,end=<END_IP_ADDRESS> \
--dns-nameserver <DNS_RESOLVER> --gateway <PROVIDER_NETWORK_GATEWAY> \
--subnet-range <PROVIDER_NETWORK_CIDR> provider
```

[各値の説明]

- <START_IP_ADDRESS>：DHCPで割り当てる始まりのIPアドレス
- <END_IP_ADDRESS>：DHCPで割り当てる最後のIPアドレス
- <DNS_RESOLVER>：使用するDNSリゾルバのIPアドレス
- <PROVIDER_NETWORK_GATEWAY>：プロバイダーネットワークのゲートウェイIPアドレス
- <PROVIDER_NETWORK_CIDR>：CIDR表記の物理ネットワーク上のサブネット

[Example]

```
$ openstack subnet create --network provider \
--allocation-pool start=203.0.113.101,end=203.0.113.250 \
--dns-nameserver 8.8.4.4 --gateway 203.0.113.1 \
--subnet-range 203.0.113.0/24 provider
```

5.1.2 セルフサービス(プライベート)ネットワークの作成

1. コントローラーノードで、以下のコマンドを実行して、ユーザー専用のCLIコマンドにアクセスします。

```
$ . demo-openrc
```

2. 以下のコマンドでネットワークを作成します。

```
$ openstack network create selfservice
```

3. 以下のコマンドを参考にし、サブネットを作成します。

```
$ openstack subnet create --network selfservice \
--dns-nameserver <DNS_RESOLVER> --gateway <SELFSERVICE_NETWORK_GATEWAY> \
--subnet-range <SELFSERVICE_NETWORK_CIDR> selfservice
```

[各値の説明]

- <DNS_RESOLVER>：使用するDNSリゾルバのIPアドレス
- <SELFSERVICE_NETWORK_GATEWAY>：セルフサービスネットワーク上で使用するゲートウェイのIPアドレス
- <SELFSERVICE_NETWORK_CIDR>：セルフサービスネットワークで使用するサブネット

[example]

```
$ openstack subnet create --network selfservice \
--dns-nameserver 8.8.4.4 --gateway 172.16.1.1 \
--subnet-range 172.16.1.0/24 selfservice
```

5.1.3 ルーターの作成

- セルフサービスネットワークは、通常双方向NATを実行する仮想ルーターを使用してプロバイダーネットワークに接続します。 各ルーターには、少なくとも1つのセルフサービスネットワーク上のインターフェイスと、プロバイダーネットワーク上のゲートウェイが含まれます。
- プロバイダーネットワークには[router:external]オプションを含めて、セルフサービスルーターがインターネットなどの外部ネットワークへの接続に使用できるようにする必要があります。 管理者またはその他の特権ユーザーは、ネットワークの作成時にこのオプションを含めるか、後で追加する必要があります。 この場合、プロバイダーネットワークの作成時に[--external] パラメーターを使用して[router:external]オプションを設定します。

1. コントローラーノードで、以下のコマンドを実行して、ユーザー専用のCLIコマンドにアクセスします。

```
$ . demo-openrc
```

2. 以下のコマンドでルーターを作成します。

```
$ openstack router create router
```

3. セルフサービスネットワークのサブネットをルーターのインターフェイスとして追加します。

```
$ openstack router add subnet router selfservice
```

4. ルーターのゲートウェイとしてプロバイダーネットワークを設定します。

```
$ openstack router set router --external-gateway provider
```

5.1.4 動作確認

- 続きの作業を進める前に動作を確認し、問題がある場合は修正することをお勧めします。

1. コントローラーノードで、以下のコマンドを実行して、ユーザー専用のCLIコマンドにアクセスします。

```
$ . admin-openrc
```

2. ネットワーク名前空間の一覧を表示します。 1つのqrouter名前空間と2つのqdhcp名前空間が表示されるのを確認します。

```
$ ip netns
```

3. ルーターのポートを一覧表示して、プロバイダーネットワークのゲートウェイIPアドレスを確認します。

```
$ openstack port list --router router
```

4. コントローラーノードまたは物理プロバイダーネットワーク上の任意のホストから、手順3で確認したIPアドレスに pingを実行します。

   pingが成功したら仮想ネットワークの作成は完了です。次の手順に進んでください。

5.2 フレーバー m1.nano の作成

- 最小のデフォルトフレーバーは、インスタンスごとに512MBのメモリを消費します。メモリが4GB未満のComputeノードがある環境では、インスタンスごとに64MBしか必要としないm1.nanoフレーバーを作成することをお勧めします。このフレーバーは、テスト目的のCirrOSイメージでのみ使用してください。

1. 以下のコマンドでフレーバーを作成します。

```
$ openstack flavor create --id 0 --vcpus 1 --ram 64 --disk 1 m1.nano
```

5.3 キーペアの生成

- ほとんどのクラウドイメージは、従来のパスワード認証ではなく、公開鍵認証をサポートしています。インスタンスを起動する前に、Computeサービスに公開鍵を追加する必要があります。

1. コントローラーノードで、以下のコマンドを実行して、ユーザー専用のCLIコマンドにアクセスします。

```
$ . demo-openrc
```

2. 以下のおコマンドでキーペアを生成し、公開鍵を追加します。

```
$ ssh-keygen -q -N ""
```

```
$ openstack keypair create --public-key ~/.ssh/id_rsa.pub mykey
```

3. キーペアが追加されたら確認します。

```
$ openstack keypair list
```

5.4 セキュリティーグループルールの追加

- デフォルトでは、デフォルトのセキュリティグループがすべてのインスタンスに適用され、インスタンスへのリモートアクセスを拒否するファイアウォールルールが含まれます。 CirrOSなどのLinuxイメージの場合、少なくともICMP(ping)とセキュアシェル(SSH)を許可することをお勧めします。

1. 以下のコマンドでデフォルトのセキュリティグループにICMPとSSHを許可するルールを追加します。

```
$ openstack security group rule create --proto icmp default
$ openstack security group rule create --proto tcp --dst-port 22 default
```

5.5 インスタンスの起動

- ネットワーク オプション1を選択した場合は、プロバイダーネットワークでのみインスタンスを起動できます。 ネットワークオプション2を選択した場合は、プロバイダーネットワークとセルフサービスネットワークでインスタンスを起動できます。

- インスタンスを起動するには、少なくともフレーバー、イメージ名、ネットワーク、セキュリティ グループ、キー、およびインスタンス名を指定する必要があります。

1. コントローラーノードで、以下のコマンドを実行して、ユーザー専用のCLIコマンドにアクセスします。

```
$ . demo-openrc
```

2. 以下のコマンドで利用可能なフレーバーの一覧を表示します。

```
$ openstack flavor list
```

3. 以下のコマンドで利用可能なイメージの一覧を表示します。

```
$ openstack image list
```

4. 以下のコマンドで利用可能なネットワークの一覧を表示します。

   このインスタンスは、セルフサービスネットワークを使用します。ただし、名前の代わりにIDを使用してこのネットワークを参照する必要があります。

```
$ openstack network list
```

5. 以下のコマンドで利用可能なセキュリティグループの一覧を表示します。

```
$ openstack security group list
```

6. 以下のコマンドでインスタンスを起動します。

```
$ openstack server create --flavor m1.nano --image cirros \
--nic net-id=<SELFSERVICE_NET_ID> --security-group default \
--key-name mykey selfservice-instance
```

7. 以下のコマンドでインスタンスの状態を確認します。

   ビルドプロセスが正常に完了すると、ステータスが[BUILD]から[ACTIVE]に変わります。

```
$ openstack server list
```

8.