OSC2014 Spring

OpenStackの概要および OpenStackによるクラウド活用法のご紹介

2014年2月28日

日本ヒューレット・パッカード株式会社
テクノロジーコンサルティング事業統括
デリバリー統括本部 オープンソース部
惣道　哲也

Agenda

1. OpenStackとは

2. OpenStackが動作する仕組み
   - OpenStackを構成するコンポーネント
   - ブロックストレージcinderとオブジェクトストレージswiftの違いと使い分け
   - neutronにより実現できるネットワーク構成
   - Havanaリリースの新機能「ceilometer」「heat」の概要

3. OpenStackによるクラウド活用法について

- Appendix
  - 検証時に発生したトラブルとその対処
  - プライベートクラウド検証のためのPoC環境の構成例

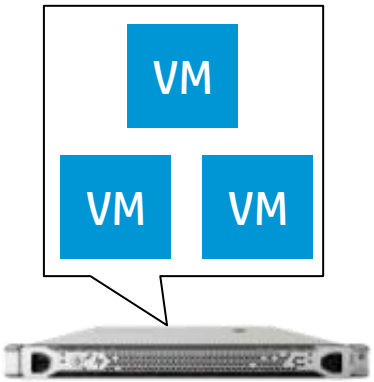
1. OpenStackとは

OpenStackが注目される背景

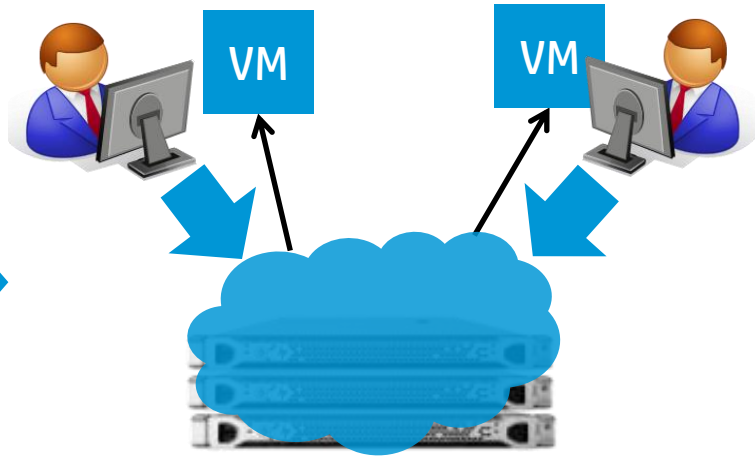
仮想化からクラウドへのIT基盤の進化

仮想化によるIT基盤統合 → IT基盤のクラウドサービス化

サイロ型IT基盤



仮想化技術を活用した
IT基盤統合と標準化



IT基盤のクラウドサービス化

得られる効果

- リソース稼働率の向上
- 運用作業の標準化
- システムコストの最適化

- ITリソースを、サービスメニュー化して迅速に提供
- セルフポータルの提供による管理業務の自動化

クラウドIT基盤を構築できるソフトウェアのニーズが拡大

クラウドIT基盤とは

- **ITリソースの「サービス化」＋「標準化」＋「自動化」**
  - 実装手段として「仮想化」技術を利用することが多いが「仮想化」は必須ではない

サービス化
- 利用者はIT基盤の内部構造を意識しない
- 使いたいときに使いたい分を利用する
- 使い終わった後に資産、在庫として残らない

標準化
- 次のような条件を共通メニューとして揃える
  - ✓ マシンリソース要件(OSイメージ、CPU、メモリ、ストレージ、ネットワーク等)
  - ✓ 利用条件(SLA、セキュリティ等)
  - ✓ 申請方法、運用管理等のプロセス

自動化
- 利用申請やリソース払い出しなどの管理タスクをポータルやAPIで自動化

IT基盤の利用者のメリット

- 要求に応じたスペックの仮想サーバやストレージをすぐに利用できる

IT基盤の管理者のメリット

- 利用者ごとの個別対応が不要
- 運用の効率化と管理の向上
- ヘルプデスクの負荷軽減
- 統合によるコスト削減効果

OpenStackとは

クラウドIT基盤の標準を目指しているオープンソースのクラウド基盤ソフトウェア

- クラウド基盤ソフトウェアを開発するOSSプロジェクト
  - http://www.openstack.org/
  - 運営体制
    - 非営利団体であるOpenStack Foundationが運営
    - HP、RedHat、SUSE、Canonical、AT&T、Cisco、IBM、DELL、RackSpace、NEC、Intel、VMware、EMC、Yahoo!などが参加
    - Linux Foundationモデルに類似

- ITインフラのライフサイクルを管理
  - サーバ、ストレージ、ネットワークリソースの生成、割当、返却、再利用
  - APIによるハードウェアのソフトウェア化
  - 異なる利用者の仮想マシンを同一物理サーバ上で利用できるマルチテナント対応

- 実装言語はpython
  - 内部でLinuxの各コマンドを呼び出すことで、環境構築・管理を実現
    - kvm/qemu, lvm, iscsi, iptables, openvswitch, ip netns, ...

OpenStack開発の経緯

NASA

Nebula
(IaaS基盤)

2009年
独自のクラウドプ
ラットフォームを
開発・運営

RackSpace

Cloud Files
(ファイルホスティング)

2008年
独自のクラウドファイル
ホスティングサービスを
開発・運営

2010/7

OpenStack

ロードマップ

Austin	Bexer	Cactus	Diable	Essex	Folsom	Grizzly	Havana	IceHouse
							現状 最新版	次期リリース予定
2010/10	2011/2	2011/4	2011/10	2012/4	2012/9	2013/4	2013/10	2014/4

OpenStack導入事例

国内外問わず導入事例が多く、商用サービスとしての利用も増えている

①パブリッククラウドサービス向け用途

- HP (HP Cloud Services – www.hpcloud.com)
  - 数千台の物理マシンとPByteクラスストレージシステムが複数DCにて稼働
- RackSpace
- GMO (お名前.com VPS)
- Korea Telecom (オブジェクトストレージswiftの商用サービス)

②商用サービス・社内サービス基盤向け用途

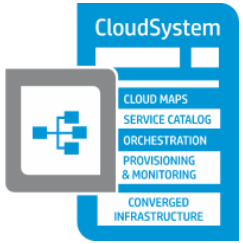
- PayPal
  - 「可用性に妥協することなく、すばやくスケールする能力」という要件を満たすため、それまで80,000台のVMWareで稼働していたプラットフォームをOpenStackにリプレースすると発表 (http://www.openstack.org/user-stories/paypal/agility-with-stability/)
- Yahoo!
  - 開発環境、Hadoop/Stormクラスタ、また商用サービス(ピーク時の突発対応)で利用
- サイバーエージェント
  - Ameba基盤をOpenStack上に構築

③研究・学術機関向け用途

- CERN
- 国立情報学研究所

HPのOpenStackへの取り組み

3パターンのご要望　それぞれにお応えしたい



HP CloudSystem Matrix

HP Cloud OS for HP Moonshot

1 OpenStack®を使いこなしたい

- HP Cloud Service Automation
- HP Operations Orchestration

OpenStack®対応オーケストレーター

- HP Cloud Services (Public)
- HP Enterprise Cloud Services

- HP Cloud OS 搭載製品 (HP Cloud Systemなど)

OpenStack®

Network Plug-in / Storage Driver

対応ハードウェア

OpenStack® API

OpenStack®ベースのクラウドサービス

- HP 3PAR
- HP Lefthand
- HP VAN SDN Controller

2 OpenStack ®でIaaSを作りたい

3 OpenStack ®のIaaSを使いたい

この章のまとめ

OpenStackとは

- クラウドIT基盤とは、ITリソースの「サービス化」「標準化」「自動化」を実現するもの
- クラウドIT基盤の標準を目指すオープンソースのクラウド基盤ソフトウェア ＝「OpenStack」
- すでに商用でも活用されており、国内外で導入事例も増えてきている
- HPでも、さまざまなお客様の要望に応じるために、パブリッククラウドのHP Cloud ServicesやHP Cloud OS搭載製品などを提供しており、いずれも基盤技術としてOpenStackを採用しています

2. OpenStackが動作する仕組み

OpenStackの構成

管理者

ポータルの管理画面から、事前に利用者が使うメニューやVMイメージを登録しておく

コントローラノード

ポータル画面へのアクセス

利用者

制御

ポータル画面またはREST API経由でのシステム利用要求を受け付けOpenStack全体の制御をおこなう。

- 認証処理
- VMの生成・起動・停止など）
- ネットワーク割当・管理
- ストレージの割当・管理

コンピュートノード

SSHなど通常のサーバアクセス

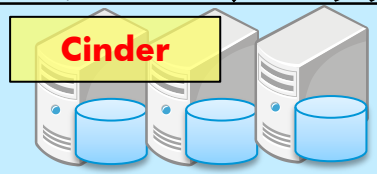
・・・

コントローラの指示により、VMの起動・停止を行う。VM起動後はシステム利用者はSSHなどを使い直接VMへアクセスすることができる。

利用者はシステム内部の仕組みを知らなくとも、使いたいときに欲しいスペックのマシンがすぐ利用できる

ディスクをマウント

ストレージノード（ブロックストレージ）

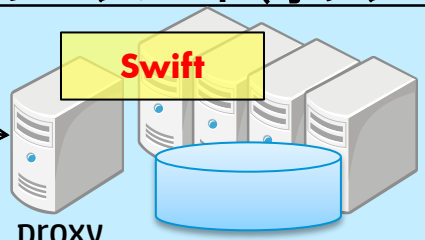


・・・

コントローラの指示により、ディスクボリュームを作成し、iSCSIディスクとしてVMに提供する。

REST API

ストレージノード（オブジェクトストレージ）



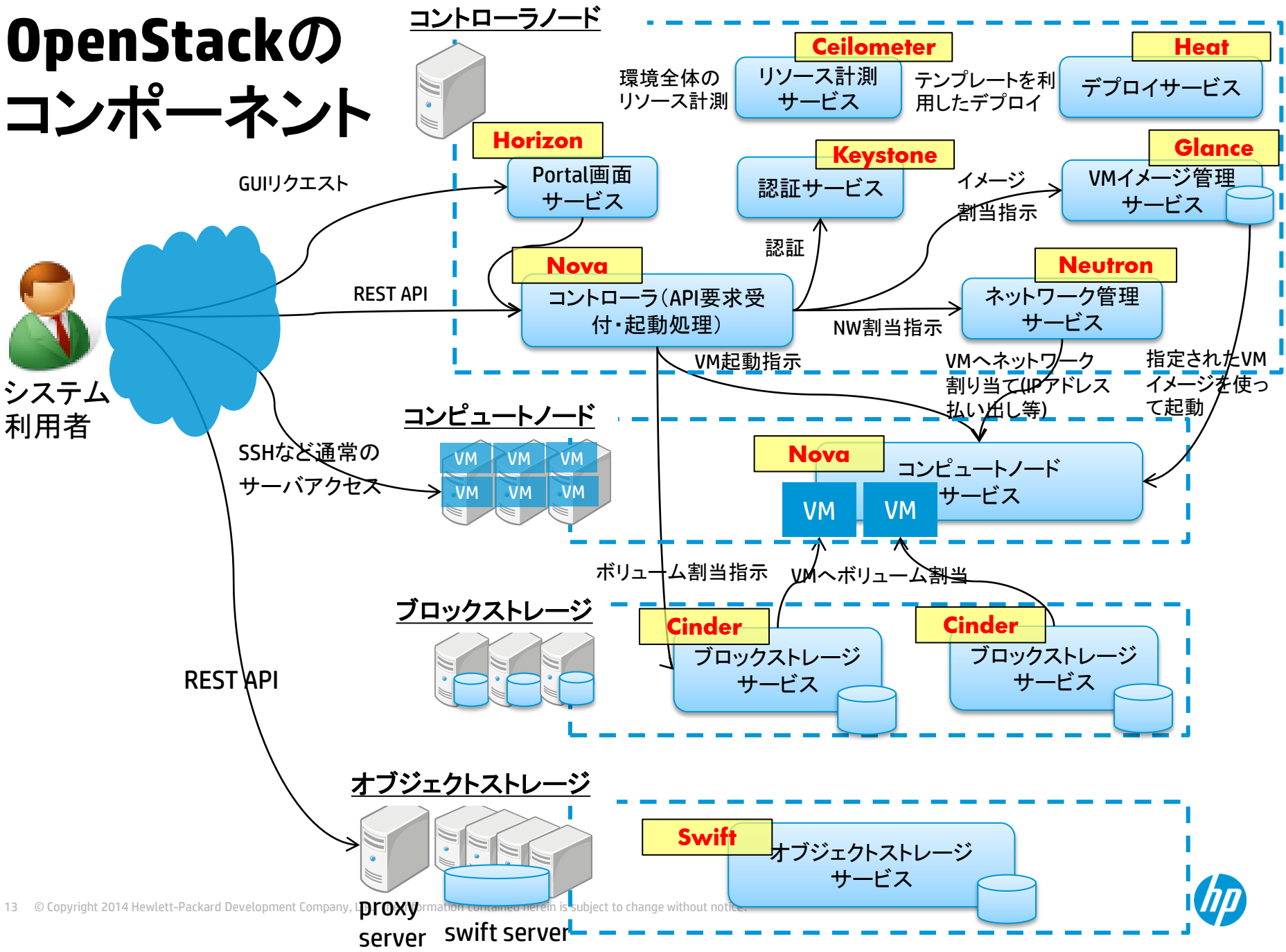
・・・

REST APIでアクセスして、画像ファイルやログファイルなどをファイル単位で格納して保存する。VMゲストが起動していなくても利用可能。

proxy server

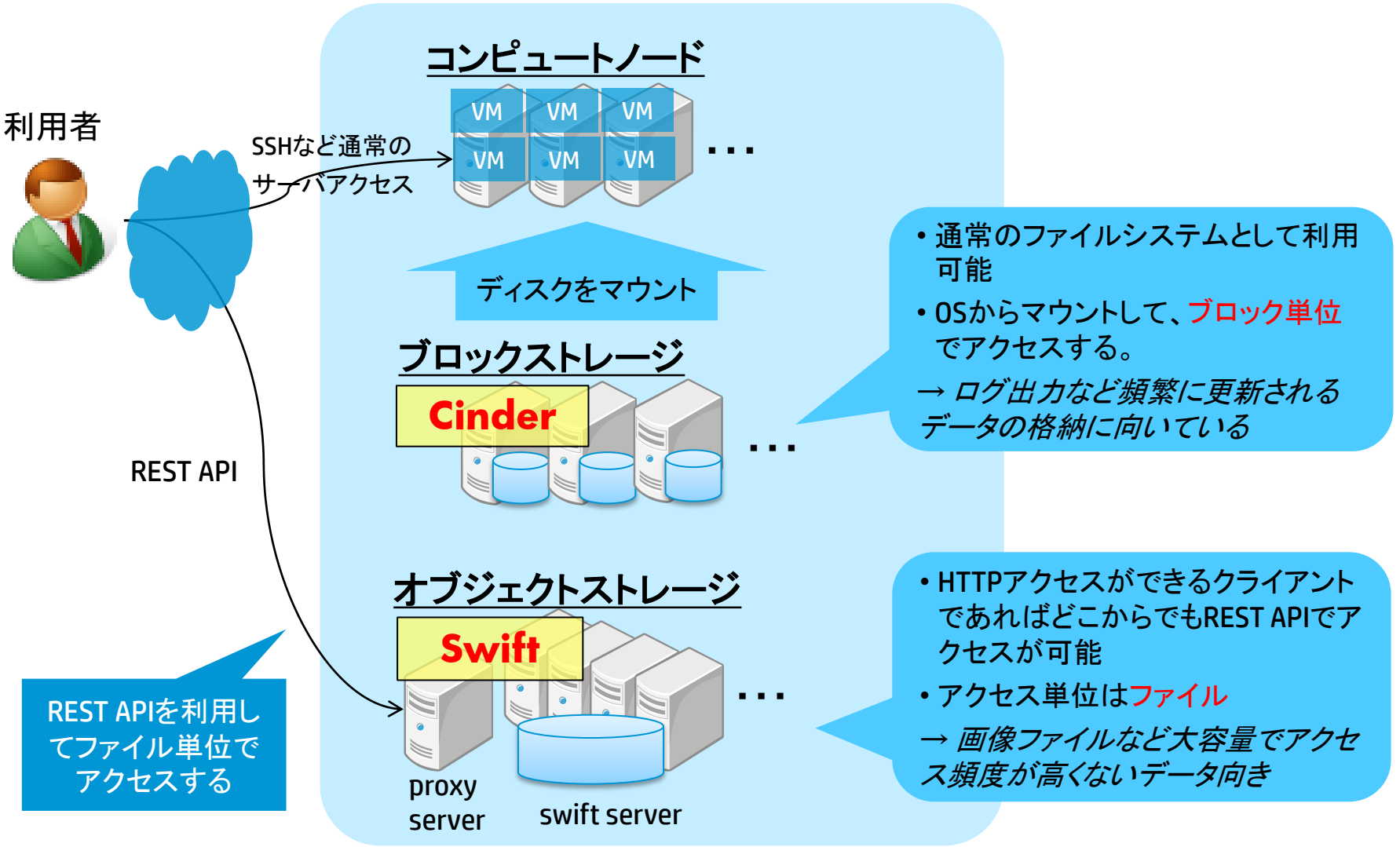
swift server

OpenStackのコンポーネント



ブロックストレージcinderとオブジェクトストレージswift

それぞれの違いと使い分け

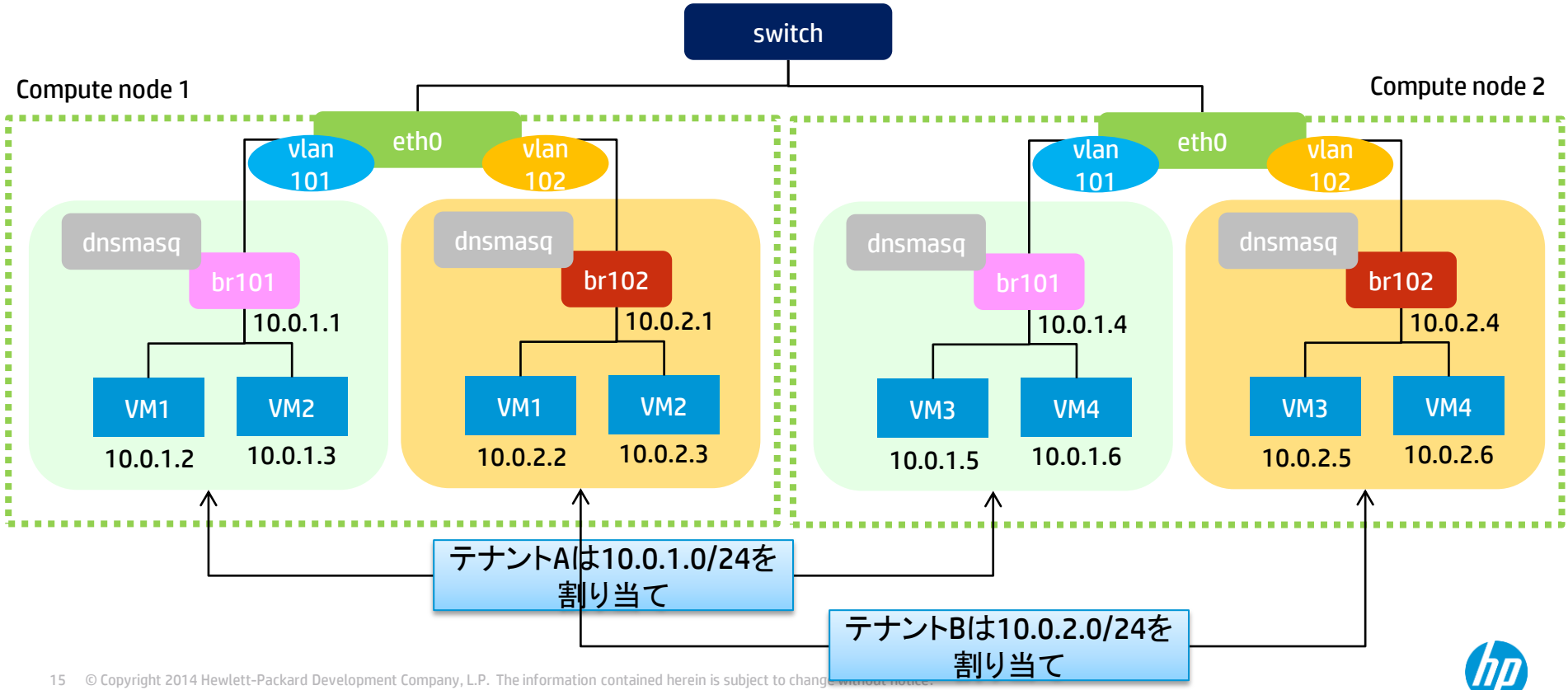


Neutron(quantum)の仕組み

Neutron以前のネットワーク構成

テナント = 利用者をグループ化した概念。
VMやネットワークの管理単位となる。
ポータルメニューでは「プロジェクト」という
用語も出てくるが全く同じ意味。

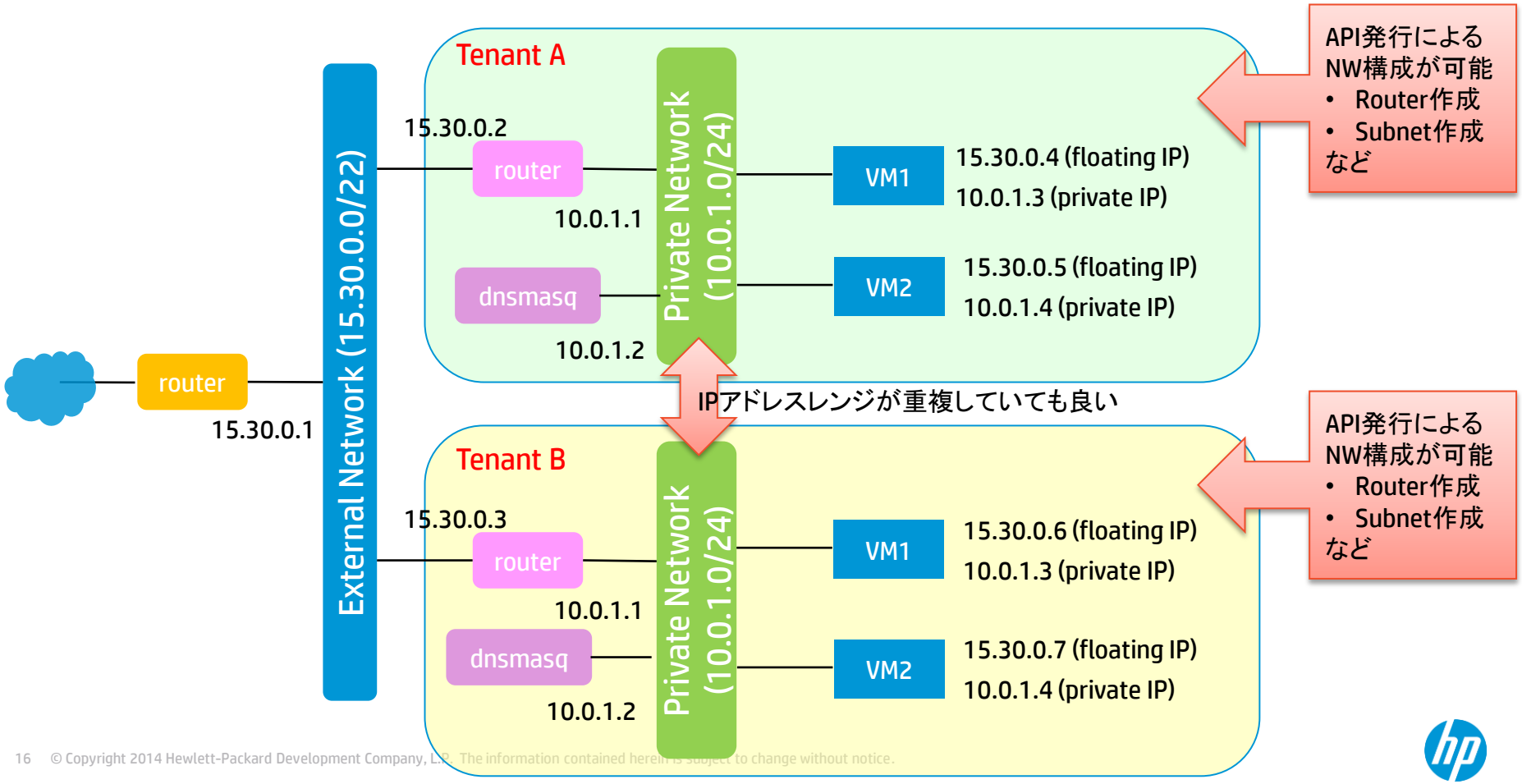
- クラウド全体で重複したIPの使用はできない
  - テナントごとに利用可能なサブネットが割り当てられる
  - テナントごとの通信トラフィックはVLANにより別テナントとは分離される
- ネットワーク構成の制御はテナントユーザにはできない（APIがサポートされていない）



Neutron(quantum)の仕組み

Neutronから可能になったネットワーク構成

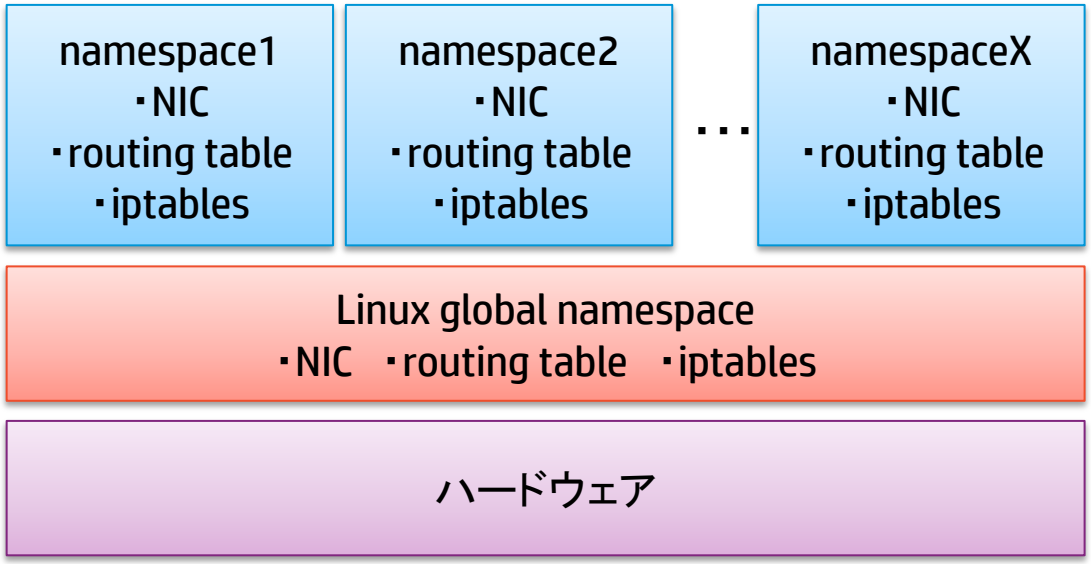
- 各テナントは専用の内部ルータを持ち、任意のネットワークアドレスをアサインすることができる
  - 他テナントと内部ネットワークのIPアドレスが重複していても良い（network namespaceで実現）
- 各テナントはneutron APIを使い、テナント利用者が内部ネットワーク構成を自由に設定可能



Neutron(quantum)の仕組み

Network namespaceとは？

- Linux kernelで実装されているネットワーク仮想化の機能
  - 1つのhost内で独立したネットワーク環境を作成できます
  - 各network namespaceはそれぞれ以下のリソースを持ちます
    - ネットワークインターフェース
    - ルーティングテーブル
    - iptables
  - Neutronでは1つのhost内での仮想マシン、仮想ネットワークを分離する目的で利用されます
  - 各namespace間でIPアドレスが重複していても良い



【注意】 各nsへのアクセス方法
global namespace（host OS)から
pingなどを打っても届かないため、
以下のようにns名を指定して実行
する必要があります。

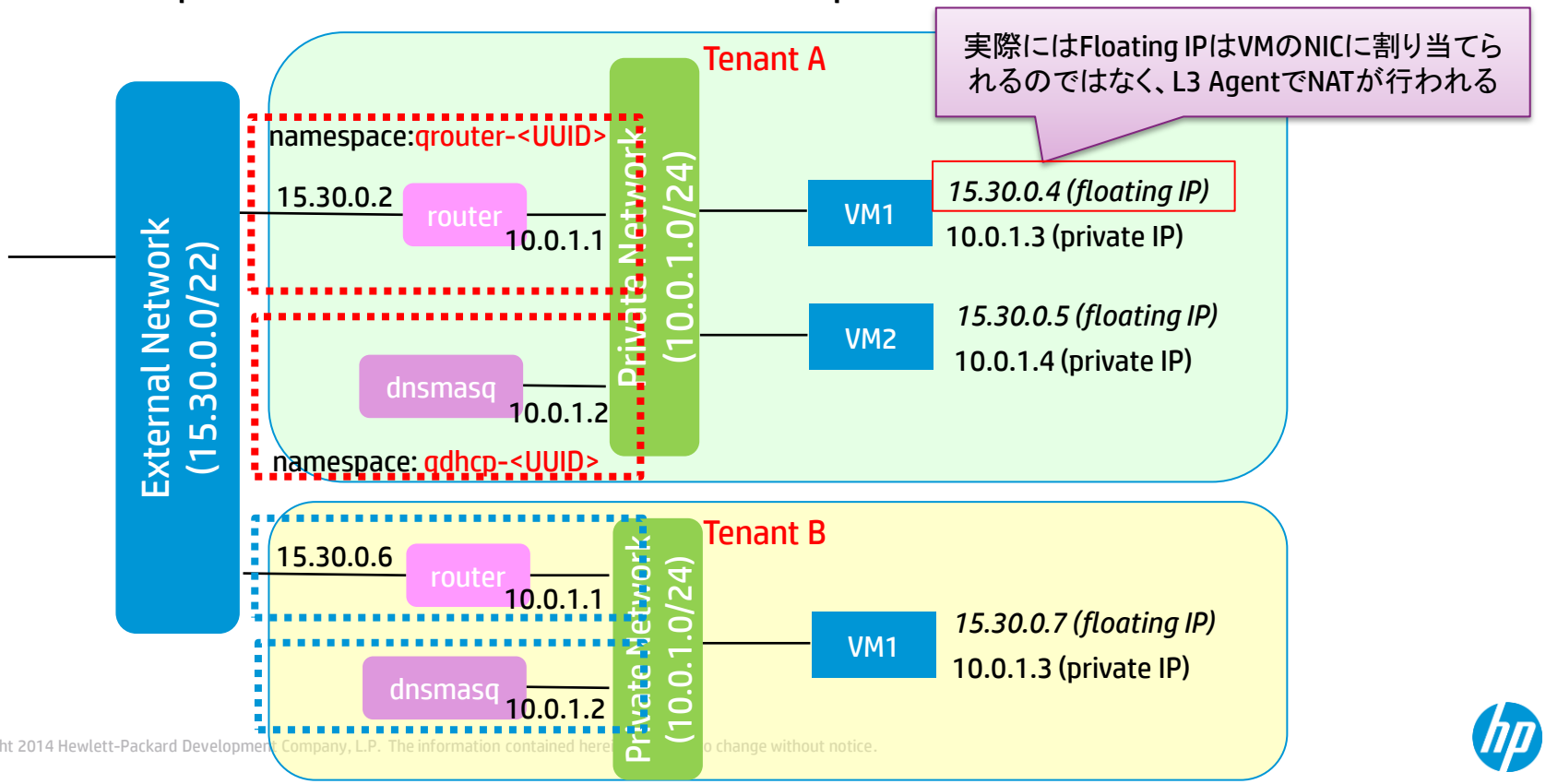
ip netns exec <ns名> command

例：
ip netns exec namespace1 ping ¥
10.0.1.1

Neutron(quantum)の仕組み

Network namespaceとは？

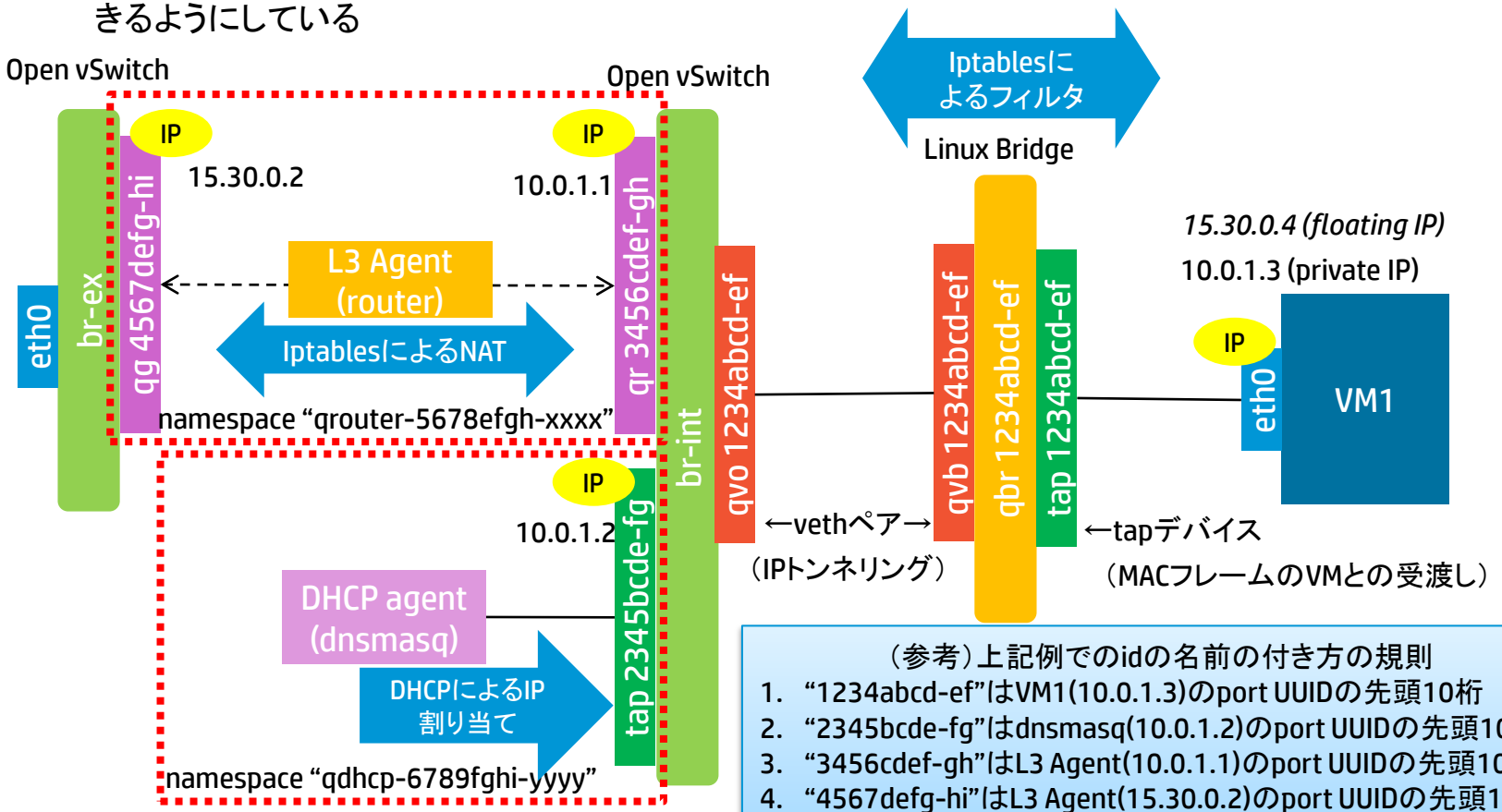
- Neutronでは以下の2種類のnamespaceが自動的に作られます
  - qrouter-<UUID> (UUIDは作成したrouterのID)
    - L3レベルのrouter1つにつき、1つのnamespaceが作られる
  - qdhcp-<UUID> (UUIDは作成したprivate networkのID)
    - L2レベルのprivate network1つにつき、1つのnamespaceが作られる



Neutron(quantum)の仕組み

Neutron構成の実装方法

- vethペアやtapデバイスを使い、ネットワークを構成しますが、若干理解しづらい点があります
  - 現在KVM/QEMUはvethが非サポート、TAPのみサポートであるため、VM接続用にtapデバイスが必要
  - Security groupsの実装にiptablesが使われるが、現在Open vSwitchに直接接続されたtapデバイスをiptablesで制御することはサポートされておらず、workaroundとして、間にLinux Bridgeを挟むことでiptablesを利用できるようにしている



（参考）上記例でのidの名前の付き方の規則

1. “1234abcd-ef”はVM1(10.0.1.3)のport UUIDの先頭10桁
2. “2345bcde-fg”はdnsmasq(10.0.1.2)のport UUIDの先頭10桁
3. “3456cdef-gh”はL3 Agent(10.0.1.1)のport UUIDの先頭10桁
4. “4567defg-hi”はL3 Agent(15.30.0.2)のport UUIDの先頭10桁
5. “5678efgh-xxxx”はrouterのUUID（全桁）
6. “6789fghi-yyyy”はprivate networkのUUID（全桁）

Havana新機能のご紹介: Ceilometer

リソース使用量の計測

概要

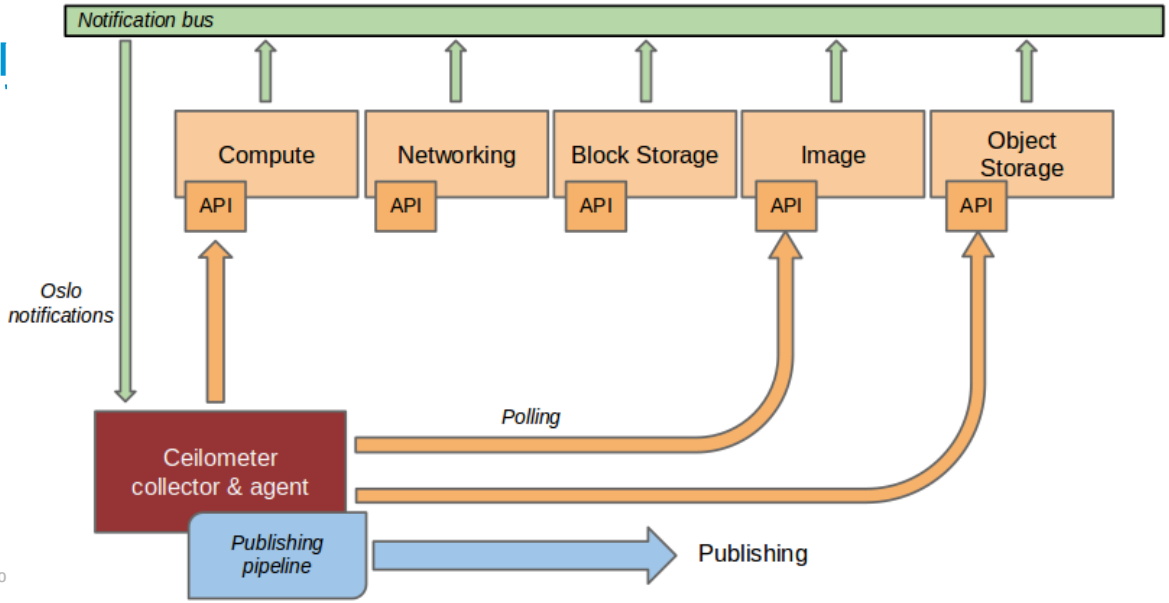
OpenStackの各コンポーネント(Nova,Swift,Glance, Cinder 等)のリソース計測を実行する。

使い方の例

- 従量課金に必要なリソース消費量情報の取得
- リソース消費量が設定した閾値を超えた際に、ユーザが設定したアクションを実行
- Heatと連携して、設定したリソース消費量を超えた際にインスタンスを追加で起動する(オートスケール)

取得できるリソースの例

- vCPU数
- メモリ量
- HDDの使用量
- ネットワークの流量
- イメージの利用サイズ



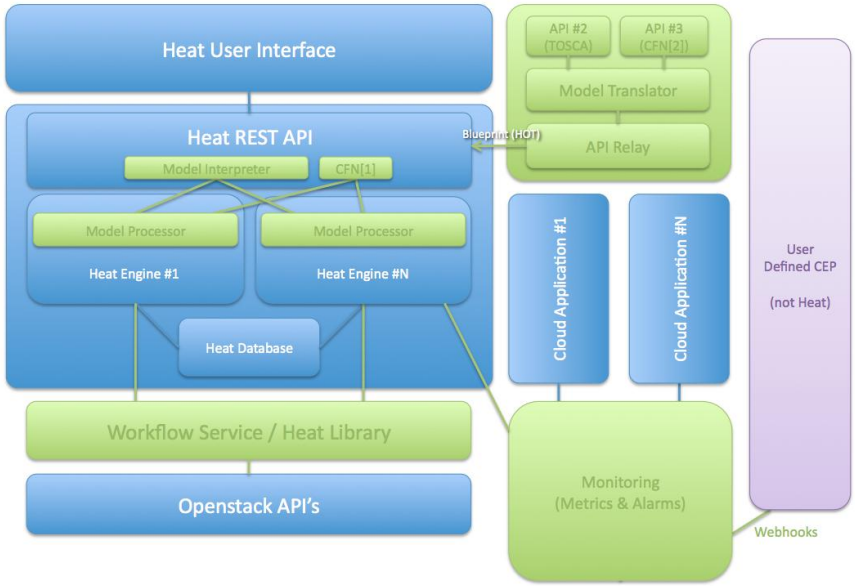
Havana新機能のご紹介: Heat

VM/ネットワーク等各種リソースの一括設定・設定

概要

VMの作成やネットワークの設定等の情報をまとめてテンプレートに記述しておく。それを利用してクラウド上に自動で一括デプロイを行う。

テンプレートはAWS CloudFormationと互換性がある。



使い方の例

- 開発環境のシステム構成をテンプレートに記述しておき、それとまったく同じシステム構成を別環境に再現する
- Chef/Puppetと連携してミドルウェアの導入・設定までを自動化する
  - インスタンス生成、NW設定：Heatテンプレートで記述
  - ミドルウェアの導入・設定：Chef/Puppetレシピに記述

この章のまとめ

OpenStackが動作する仕組み

- OpenStackは、制御用のノードと、VM・ストレージ・NW等の各リソースを提供するノードから構成

- Cinderとswiftの違いと使い分け
  - ブロックストレージcinderは、通常のファイルシステムとしてOSからマウントしてアクセスする
    - ログ出力など参照・更新のアクセス頻度が高いデータに向いている
  - オブジェクトストレージswiftは、HTTPアクセスができるクライアントであればどこからでもアクセスが可能
    - 画像データなど大容量でアクセス頻度が高くないデータに向いている

- Neutronで実現できるようになったことと仕組みについて
  - network namespaceを使用して、複数テナントでお互いに干渉しないネットワークを構成できる
  - DHCP agent(dnsmasq)はprivate network内でのIPアドレスを割り当てる
  - L3 agent(内部ルータ)は外部ネットワーク(FloatingIP)とprivate network間のをNATを行う
  - 各テナントごとに2つ（DHCP agent/L3 agent）のnamespaceが自動的に生成される
  - テナント内部ではUUID名のついたさまざまなtap/vethデバイスが生成されている

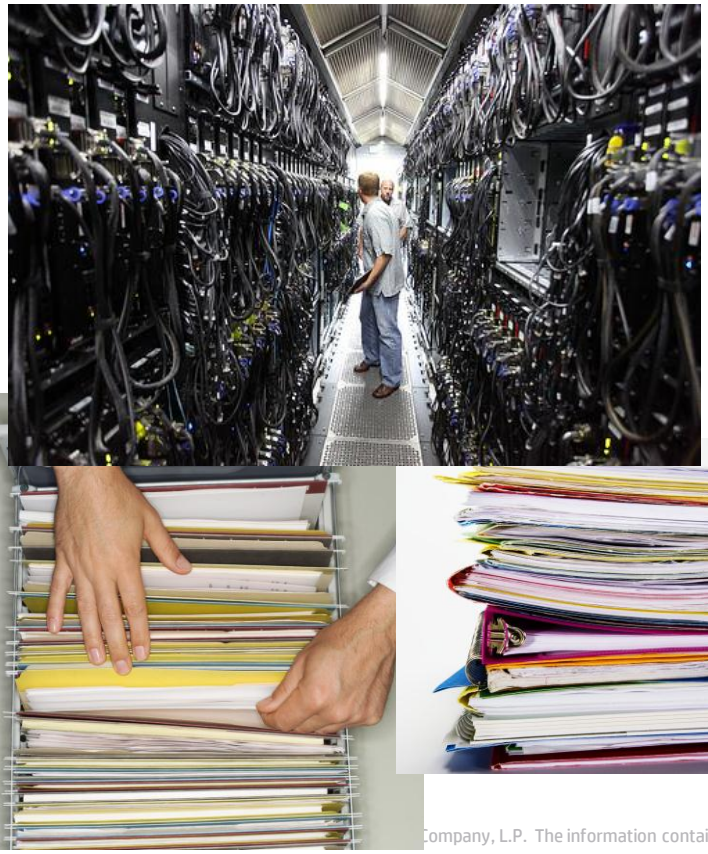
- Havanaリリースでは、リソース使用量の計測を行うCeilometerと、VM/NWの一括設定・生成を行うHeatが利用できるようになった

3. OpenStackの活用例

OpenStackの活用によるインフラ設計・構築・運用の変化

クラウド時代の新しい考え方へ

- オンプレを前提とした方法とは異なる考え方の登場
  - 例：CPUやメモリのサイジングは事前に行わず、必要に応じて容易にスケールアップが可能
- いくつかの例をOpenStackでの実装方法と合わせてご紹介します



①カスタマイズしたOSイメージによるサーバ複製

「起動可能ボリューム」の利用

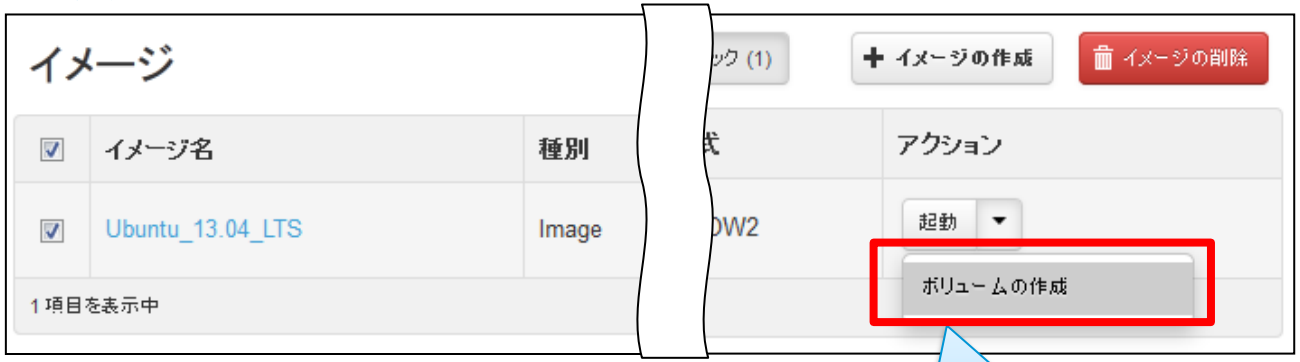
- 概要
  - 自分たちの用途にあわせたOSやミドルウェア、アプリケーションの導入、設定を行って、その状態をボリュームとして保存する

- メリット
  - スケールアウトするときなど、同じ状態のサーバが一度に大量に必要な場合に、一からセットアップする必要がなくすぐに利用することができる
  - Webサーバ用ボリューム、DBサーバ用ボリュームなど用途ごとに用意しておくと楽
  - CPU数、メモリサイズ等のフレーバーは起動時に自由に選択できる

- 実装方法
  1. Imageメニューから「ボリューム作成」で起動可能ボリュームを作成
     - カスタマイズ前の状態
  2. Instanceの起動メニューで「ボリュームから起動」を選び、上記ボリュームを指定する
     - このとき、インスタンスのルートディスクにこのボリューム(/dev/vda)がアタッチされている状態
  3. OS設定やミドルウェアの導入・設定などを自分たちの用途に合わせてカスタマイズする
  4. インスタンスを終了(Terminate)する
     - カスタマイズが完了した状態

①カスタマイズしたOSイメージによるサーバ複製

利用方法の例



イメージから「ボリュームの作成」を実行して、起動可能ボリュームを作成しておく

インスタンス起動時に「ボリュームから起動」を選んで起動する

②スナップショットを使ったデータ領域のバックアップ

迅速で安全なバックアップの実現

- 概要
  - ある時点でのデータ領域用ファイルシステムのスナップショットを作成する

- メリット
  - スナップショットの作成自体は短時間で完了する
    - 例としてデータベースのデータ領域全体のオンラインバックアップ等が可能

- 実装方法
  1. バックアップ対象となるファイルシステムの一貫性を確保するため、スナップショット取得前にMySQLの書き込みロックおよびファイルシステムのフリーズを行う
  2. 「cinder snapshot-create」コマンドを使って、対象ボリュームのスナップショットを取得
  3. スナップショット取得完了後、ファイルシステムのアンフリーズおよびMySQLのアンロックを行う

MySQLのデータ領域をバックアップする手順の例

```
# ssh vm1 mysql –uroot -ppassword-e “FLUSH TABLES WITH READ LOCK; "
# ssh vm1 xfs_freeze –f /mnt/data
# cinder  snapshot-create  --force True  snap-vol  data-vol
# ssh vm1 xfs_freeze –u /mnt/data
# ssh vm1 mysql –uroot -ppassword-e “UNLOCK TABLES; "
```

②スナップショットを使ったデータ領域のバックアップ

迅速で安全なバックアップの実現

- ファイルシステムのフリーズとは
  - ゲストのファイルシステムのdirtyバッファの内容はメモリ上にありバックアップされない
    - データ整合性を確保するため、ファイルシステムのフリーズを行ってからスナップショットを取得する
    - フリーズを行えばI/Oが止まり、メモリ上のdirtyバッファがすべて書き出されることが保証される
  - フリーズをサポートするファイルシステムはXFS（Linux カーネル 2.6.29 以降であればext3,ext4も可）。RHELであれば6.5以降からはext3,ext4でも利用できる

- バックアップにおける注意点
  - インスタンスにボリュームがアタッチされている場合は、cinderコマンドに「--force True」オプションをつける必要がある
  - 取得済スナップショットはそのままにせず、これを元にしたボリュームを作成しておく
    - 取得済スナップショットはバックアップ元のボリュームに依存しており、このボリュームが破損するとデータが参照できなくなるため
  - 環境による差異等もあるため、事前に十分な検証を実施してください

③Heatテンプレートを使ったサーバ群の自動起動

複雑な環境もミスなく効率よく構築

- 概要
  - Heatテンプレートに立ち上げるサーバやネットワークなどのリソースを記述しておき、Heatコマンドもしくは GUIからシステム環境全体を一括で自動起動する
  - 終了時も一度にシステム環境全体を破棄できる

- メリット
  - 手順書ベースの運用と比較して、操作ミスがなく、短時間で環境構築ができる
  - 検証環境で確認した環境と同一の環境が容易に本番環境にも構築できる
  - テンプレートはテキストベースのため、gitなどを使って共同開発・バージョン管理が可能

- 実装方法
  1. Heatテンプレートのフォーマットを記載する
     - 新規作成、もしくは、既存環境のフォーマットがあれば再利用する
  2. テンプレート名を指定してHeat起動コマンドを実行
     - このときパラメータを渡すことができるので、状況により、CPU数を変えたりすることも可能
  3. 利用後は、テンプレート名を指定してHeat終了コマンドを実行

③Heatテンプレートを使ったサーバ群の自動起動

複雑な環境もミスなく効率よく構築

- Heatテンプレートのみを使うか、他のツールと連携するか？
  - 単純な構成であれば「1.Heatテンプレートのみ」でも良いが、ある程度複雑な構成になると、テンプレート記載量が増え管理が煩雑になってくるため「2.他ツールとの連携」がおすすめ

1. Heatテンプレートだけを使い、すべての設定情報を記載
   - Heatテンプレートファイルがメンテナンス対象
     - 記載内容の例
       - CPU/メモリ等のHWリソース
       - NW構成
       - インストールパッケージ
       - 設定ファイル
       - その他カスタムスクリプトの実行

2. ミドルウェア、アプリケーション領域はChef,Puppetなどのツールと連携
   - Chefと連携した場合の例
     - Chefテンプレートに、導入アプリ・ミドルウェアの情報、設定を記載
     - Heatテンプレートに、HWリソース、NW構成およびchef-soloのインストール・実行までを記載
     - Heat起動時のオプションとして、Chefテンプレートファイル名を渡す
     - （さらにserverspecなどを利用すれば構築環境の正しさを自動テストすることも可能）

③Heatテンプレートを使ったサーバ群の自動起動

複雑な環境もミスなく効率よく構築

- 「2.他ツールとの連携」の例：Chef(Chef-solo)との連携を記載したHeatテンプレートの例

```
...
resources:
 server1:
  type: OS::Nova::Server
  properties:
   name: Server1
   image: { get_param: image }
   flavor: { get_param: flavor }
   key_name: { get_param: key_name }
   networks:
    - port: { get_resource: server1_port }
   user_data: { "Fn::Base64" : { "Fn::Join" : ["", [
       "#!/bin/bash -v¥n",
       "curl -L https://www.opscode.com/chef/install.sh | bash¥n",
       "wget -O /tmp/chef-repo.tar.gz http://10.0.0.1/chef/web-chef-repo.tar.gz¥n",
       "tar zxvf chef-repo.tar.gz -C /home/sodo/¥n",
       “cd /home/sodo/chef-repo¥n”
       "chef-solo -c .chef/solo.rb -j.chef/node.json¥n"
   ]]}}
...
```

chef-soloインストーラファイルの取得
およびインストール

このサーバの設定を記載した
Chefレシピファイルセット(後述)

Chef-soloの実行例
これによりレシピに記載した設定が反映される

Heatで任意のコードを実行できる。ここではbashスクリプトでchefインストールおよびchefレシピを実行

③Heatテンプレートを使ったサーバ群の自動起動

複雑な環境もミスなく効率よく構築

- Chef-soloでパッケージインストールを記載する例
  - 最低で3ファイル記載すればOK

```
$ cat /home/sodo/chef-repo/.chef/solo.rb
file_cache_path "/home/sodo/chef-repo"
cookbook_path "/home/sodo/chef-repo/cookbooks“
#http_proxy http://10.0.0.1 :8080/

$ cat /home/sodo/chef-repo/.chef/node.json
{
 "run_list": [ "recipe[install_packages]" ]
}

$ cat /home/sodo/chef-repo/cookbooks/install_packages/recipes/default.rb
#
# Cookbook Name:: install_packages
# Recipe:: default
#
%w(apache2 php5 libapache2-mod-php5).each do |pkg|
 package pkg do
  action :install
 end
end
```

Chef-Soloの設定を記述
(cookbookのパスなど)

このノードで実行する内容を記述
ここではレシピの実行を指示

レシピの内容を記述
OS設定やパッケージ導入などが可能
サービス自動起動なども指示できる

この章のまとめ

OpenStackを活用したインフラ設計・構築・運用の考え方

- 本章ではOpenStackを活用したいくつかの例を紹介した

1. 同じ構成のサーバを再利用、または、スケールアウト型配備をする場合、カスタマイズした起動可能ボリュームを作っておくと良い(glanceの利用)

2. データ格納用ボリュームのスナップショットを利用することで、アプリケーションをほぼ止めずにバックアップを取得(cinderの利用)

3. 複数のサーバやネットワーク構成などをすべてテンプレートに記載して、迅速でオペミスのない構築が可能になる。「環境」の共同開発やバージョン管理が実現できる。(heatの利用)

Thank you

Appendix.
検証時に発生したトラブルとその対処

1. インスタンス起動時にメタデータの取得に失敗する

- 発生事象の内容
  - インスタンスを起動し、起動に成功
  - ただしインスタンス起動時のログに以下のようなエラーメッセージが出力される
    - util.py[WARNING]: 'http://169.254.169.254/2009-04-04/metadata/instanceid' failed [0/120s]: http error [500]
  - Pingの疎通確認において以下の状況となった
    - VNCコンソール経由でインスタンスにログインし、インスタンスから"169.254.169.254"へpingを発行する：疎通NG

- インスタンス起動時のメタデータ取得について
  - "169.254.169.254"は各インスタンスのホスト名やFloatingIPの情報などのメタデータを提供するホストのIPアドレス
    - 各インスタンスは起動時にこのIPアドレスへアクセスしてHTTP GETで情報を取得する

- 原因
  - Computeノードの以下のkernel parameterの設定ミス
    - "ip_forward=0"

- 対処
  - "ip_forward=1"と設定を変更

2. Cinderボリュームのアタッチに失敗する①

- 発生事象の内容
  - Cinderでボリュームを作成する
  - 作成したボリュームをインスタンスに接続するが、接続エラーが発生

- 原因
  - iSCSIに関連する以下２種類のソフトウェアを両方インストールしていた
    - tgtd
    - Iscsid
  - docs.openStack.orgのインストールマニュアルには両方インストールする旨の記述があるが、実際にはどちらか一方のみインストールする必要がある

- 対処
  - tgtdのみをインストールしたところ、正常に接続できた

2. Cinderボリュームのアタッチに失敗する②

- 発生事象の内容
  - Cinderでボリュームを作成する
  - 作成したボリュームをインスタンスに接続するが、接続エラーが発生

- 原因
  - Ubuntu13.10ではdefaultでapparmorが有効化されており、iSCSIに関する通信がapparmorによりrejectされていた
    - ログメッセージの例
      - Nov 6 19:29:40 havana01 kernel: [2038.003136] type=1400 audit(1383733780.198:54): apparmor="DENIED" operation="open" parent=1 profile="libvirt-c3468db6-1397-4c82-a2f4-0e54462bd5fd" name="/dev/sdb" pid=13362 comm="qemu-system-x86" requested_mask="r" denied_mask="r" fsuid=113 ouid=113

- 対処
  - 以下のようにapparmorを無効化することで接続が成功した

```
$ sudo ln -s /etc/apparmor.d/usr.sbin.libvirtd /etc/apparmor.d/disable/
$ sudo ln -s /etc/apparmor.d/usr.lib.libvirt.virt-aa-helper /etc/apparmor.d/disable/
$ sudo apparmor_parser -R /etc/apparmor.d/usr.sbin.libvirtd
$ sudo apparmor_parser -R /etc/apparmor.d/usr.lib.libvirt.virt-aa-helper
$ sudo service apparmor restart
```

3. HorizonのGUIから外部ネットワークが作成できない

- 発生事象の内容
  - HorizonのGUIを使い、プライベートネットワーク、ルータは作成できたが、外部ネットワークを作成する画面が見つからなかった

- 原因
  - テナントの”Member”権限では外部ネットワークの作成はできない
  - “Admin”権限のユーザでログインすると、外部ネットワークの作成が可能になる

- 対処
  - まず”Admin”権限のユーザでログインし、外部ネットワークを作成する
  - 次に”Member”権限のユーザでログインし、残りのタスクを実施する
    - 内部ネットワーク、サブネットの作成
    - ルータの作成
      - インターフェース設定
      - 外部接続用ゲートウェイ設定

4. 複数台構成でComputeノード/Networkノード間のtunnel作成に失敗する

- 発生事象の内容
  - Contollerノード、Computeノード、Networkノードからなる3台構成を構築
  - VMインスタンスからアクセスを行った際にtunnel作成に失敗した旨のエラーが発生
    - ovsctl showコマンドで確認したところ、”br-tun”ブリッジが存在していない

- 原因
  - Networkノード、Computeノードの/etc/neutron/neutron.confにて”local_ip”の設定漏れ

- 対処
  - Networkノード、Computeノードの/etc/neutron/neutron.confにて”local_ip”の設定を追記
  - なお、ContollerノードはGRE tunnelを利用しないため設定は不要

5. インスタンスから外部ネットワークへ接続できない

- 発生事象の内容
  - インスタンスを作成する
  - FloatingIPをインスタンスへ付与する
  - インスタンス上から外部インターネット上のURLへアクセスするが、接続に失敗する
    - FQDNでアクセスすると接続に失敗する
    - IPアドレスを指定してアクセスした場合は接続に成功する

- 原因
  - 名前解決するためのDNSサーバの設定を行っていない

- 対処
  - private subnetの設定オプションとして、DNSサーバを指定する必要がある
    - さらに、社内のイントラネットなどに構築した場合、HTTPプロキシの設定も必要

Appendix.
プライベートクラウド検証のための PoC環境の構成例

プライベートクラウド検証のためのPoC環境の構成例

PoC環境の構成例

- 利用したい機能を選択、組み合わせてプライベートクラウドの検証を行うことができます

- 弊社で検証したPoC環境の構成の例と、それぞれの主な検証目的

No	構成例	主な検証目的
1	1台のノードによるPoc環境(Swiftなし)	• OpenStackの基本的な機能検証 • 特にHavana新機能の検証

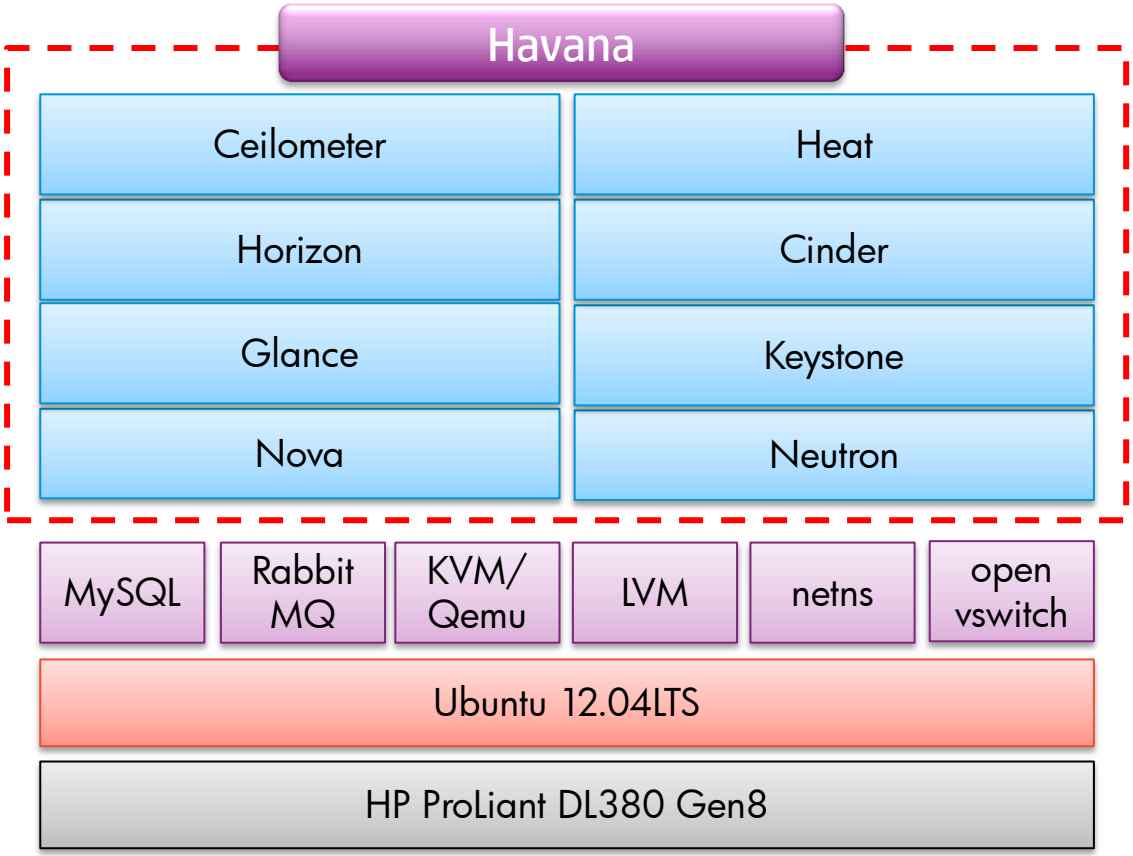
2	3台のノードによるPoC環境(Swiftなし)	• 将来的な可用性、拡張性を考慮した追加構成のための基礎検証
		• 追加構成の例
		  • Computeノードの追加(4台目)
		  • ControllerノードからDBの外だし

3	Swift検証用Poc環境	• オブジェクトストレージもを利用するための検証

①1台のノードによるPoC環境の構成例(Swiftなし)

ソフトウェアスタック

- OS上に追加パッケージを導入し、さらにHavanaリリースのパッケージを導入する
- Swift以外を全て構成（Swiftも同居可能だが、今回は別構成とした）
- 各機能コンポーネントの基本機能検証用



・・・OpenStackコンポーネント（Havanaリリース）

・・・OSに追加するパッケージ

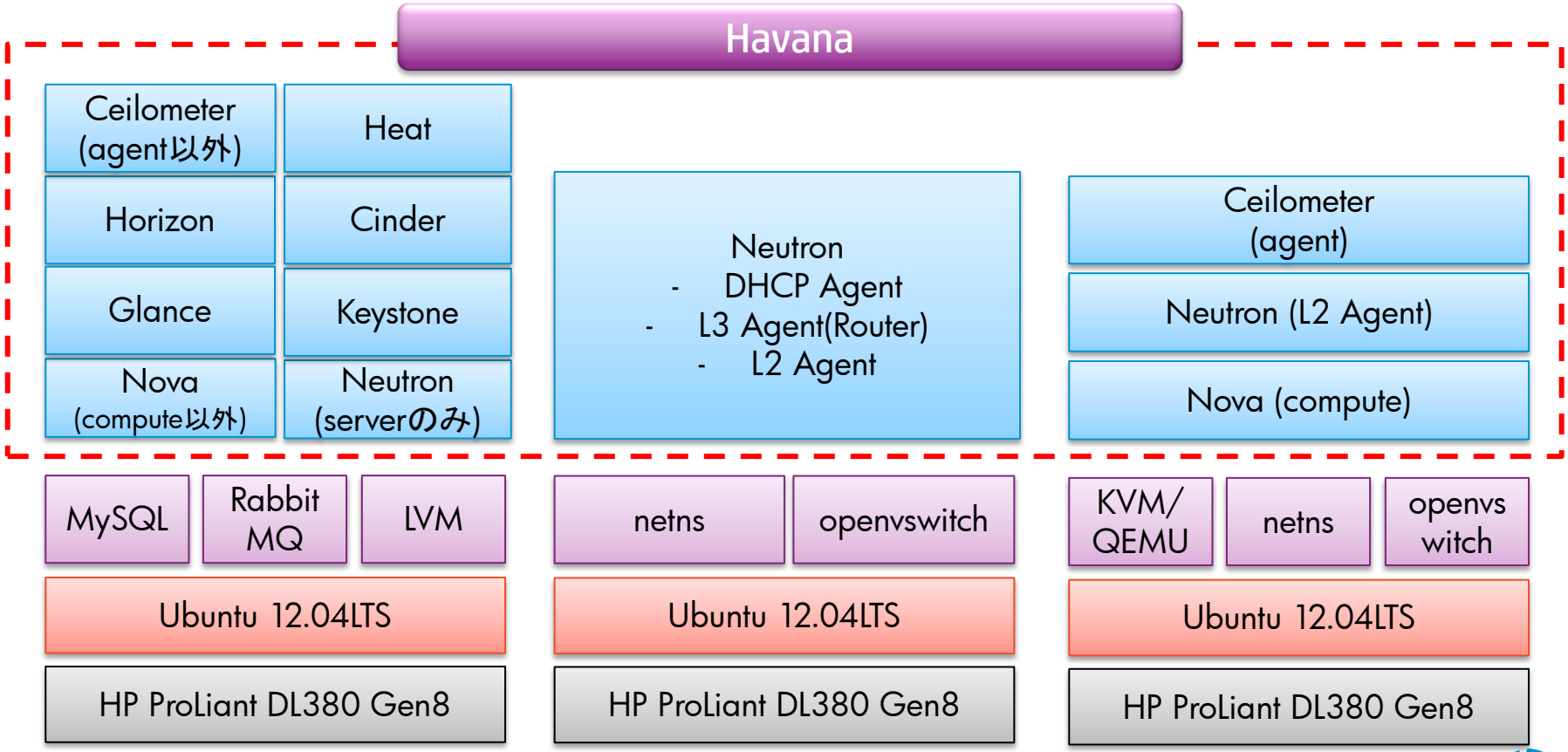
・・・OS

・・・ハードウェア

②3台のノードによるPoC環境の構成例(Swiftなし)

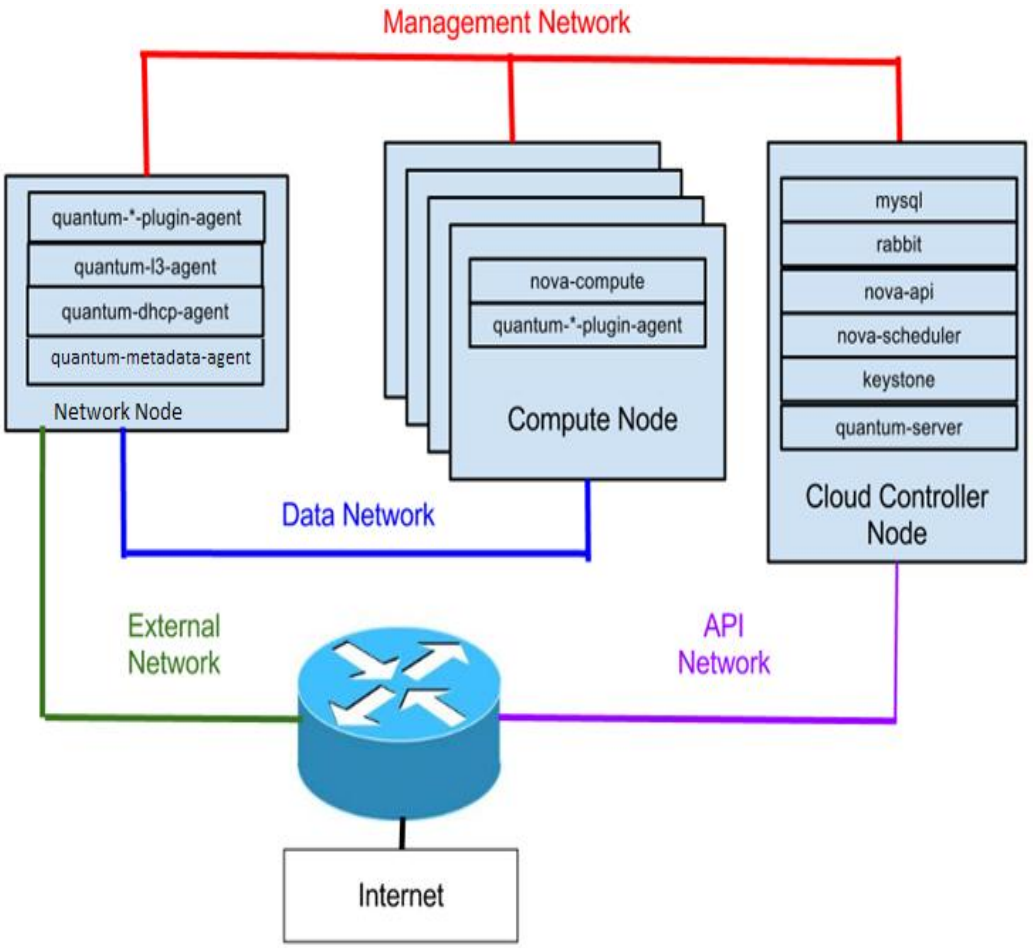
ソフトウェアスタック

- 管理用、ネットワーク、VM　Hostの役割で分離する
- ネットワーク構成にも注意が必要（後述）
- Computeノードを1台追加する、ControllerノードからDBを外だしするなどの追加検証も行える



②3台のノードによるPoC環境の構成例(Swiftなし)

Neutron構成で使われる4種類のネットワーク

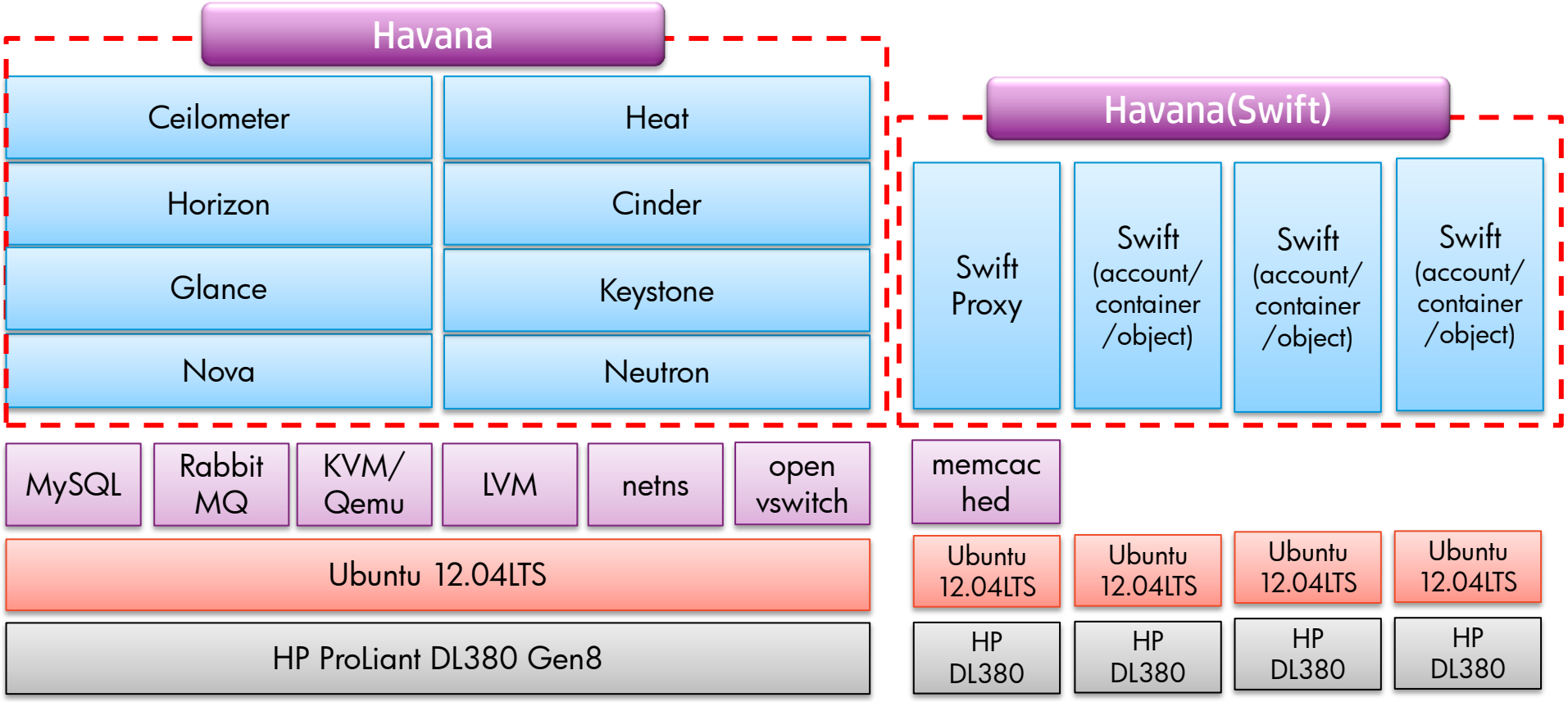


- Management Network
  - クラウド内部で全ノードが通信するための管理ネットワーク(DB、queue、認証などで利用)
- Data Network
  - Compute Node/Network Node間でVM同士が通信をするためのネットワーク(neutronのプラグイン実装に応じてVLANやGRE tunnelなどの通信を行う)
- External Network
  - インスタンスがインターネットなど外部環境と通信するための外部ネットワーク(floating IPはこのレンジからアサインされる)
- API Network
  - Nova APIやneutron APIなどのOpenStack APIへアクセスするためのネットワーク

③Swift検証用PoC環境の構成例

ソフトウェアスタック

- 別ノードにSwiftを導入
- Swiftノードは4台構成（proxy1台＋ストレージノード3台（レプリカ数:3））



Thank you