GMO INTERNET

janog35発表資料

OpenStackと ネットワークエンジニア

GMOインターネット株式会社
中里　昌弘
masahiro-nakazato@gmo.jp

プログラム応募のきっかけ

１つめ

OpenStackに苦労させられているので、
経験談とこれから触る方の手助けが出来ればと思って

２つめ

従来のネットワークのお仕事と違っていて、刺激を受けて色々と思うことがあり皆様とディスカッションが出来たら嬉しいです

発表者情報

現在の職場での仕事履歴

昔　　：社内オフィスネットワーク、バックボーン

少し前　：クラウド系ネットワーク、プログラミング少々

現在　　：OpenStackのネットワーク周り(neutron)のサービス開発

その他　：OpenStack歴は半年程、中間管理職

GMOインターネットのOpenStack活用例

ソーシャルゲーム向け
クラウドサービス
「アプリクラウド」
OpenStack havana使用

自由度の高い
VPSサービス
「conoha」
OpenStack grizzly使用

OpenStack　何がおいしいですか？

クラウドサービスやVPSサービスのベースを**無料で構築**出来る

クラウドリソース（**コンピュート、ネットワーク、ストレージ**）を
APIで一括操作管理　automate all the things　がコンセプトのよう

gre,vxlanでのオーバレイ接続やアドレッシング、ロードバランサ等
のサービスをソフトウェア上で実現

ネットワーク機器も操作出来る

バグは自分でなおしたり、仕様はカスタマイズも出来る

進化・機能改善が早い

OpenStack　何が大変？

管理者が楽になるものではなく、改良を加えて
クラウドサービスを作る**ベース**となる**玄人向け**の**荒削り**なキット

複雑で理解して稼動までに時間が掛かる

複雑で重く小規模な環境で使うメリットは感じづらい

レイヤ３以上のネットワークサービスは冗長・拡張性が微妙
（今後解消される見込み）

開発と運用に継続的に**人的リソース**を割かれる

OpenStack　概要 -1

コンピュート(仮想マシン:VM)を中心とした
ネットワーク、ストレージのリソース・構成管理ツールの**集合体**

リソース内容は全て**データベース**に保存される

認証やリソース計測のコンポーネントも存在し商用サービス
提供のベースとなりうる

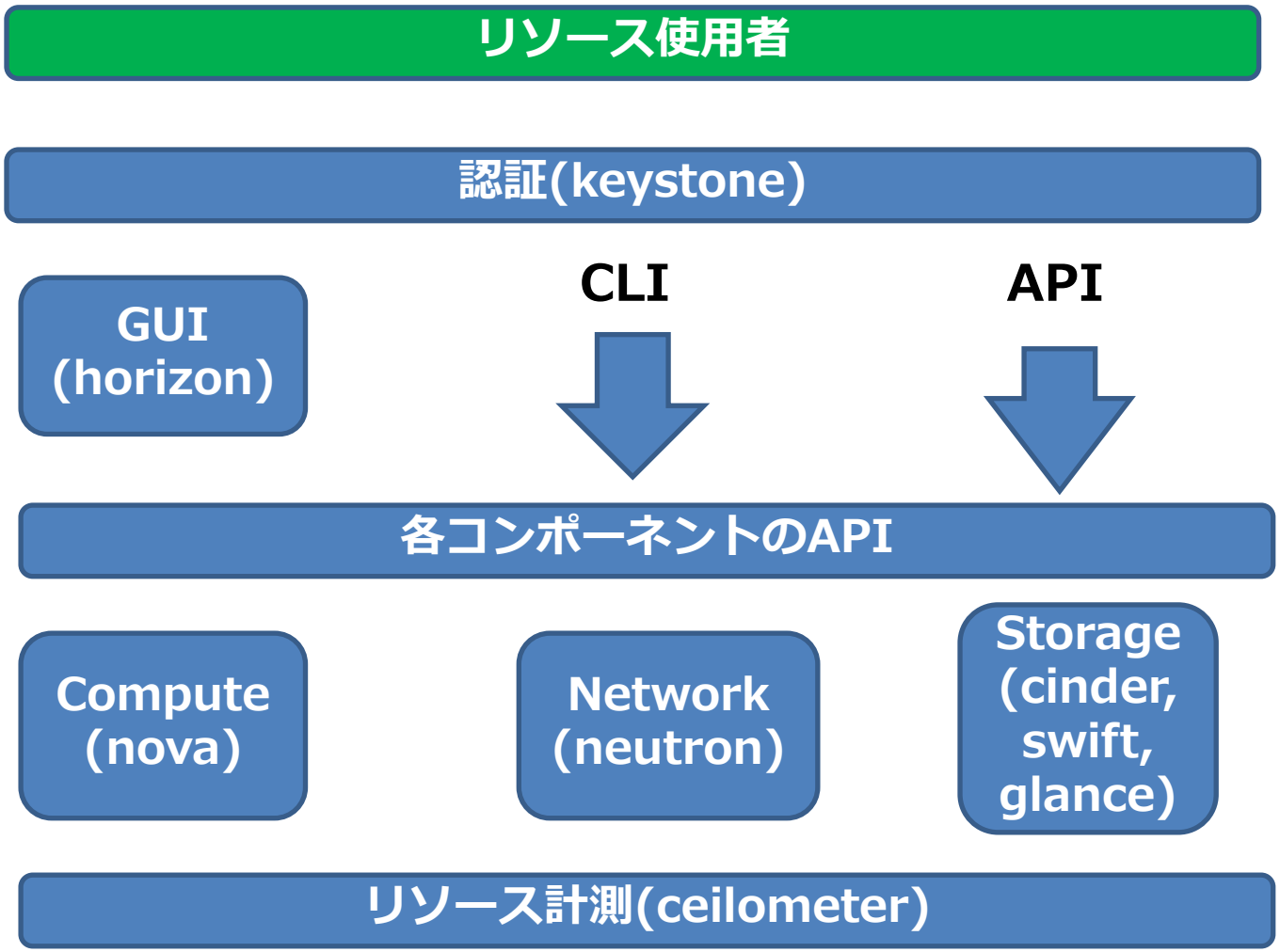
機能毎に**独立**したコンポーネントが**疎結合**でまとまって
全体としてサービス提供出来るようになっている

OpenStack　概要 -2

主な機能とコンポーネント名の関連付け

機能	コンポーネント名
・認証	: keystone
・GUIのコントロールパネル	: horizon
・VMの管理	: nova
・**VMを繋ぐ為のネットワーク系サービス**	: **neutron**
・VMにマウント出来るブロックストレージ	: cinder
・VMのディスクイメージ	: glance
・オブジェクトストレージ	: swift
・リソース計測	: ceilometer

OpenStack　概要　イメージ

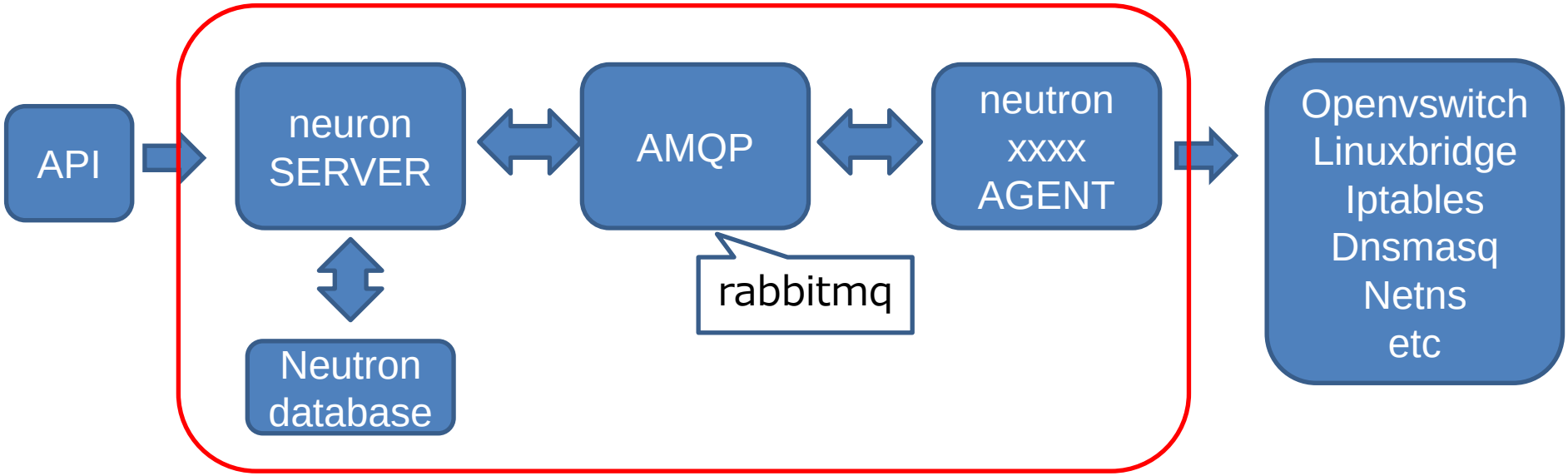


neutron　構造-1

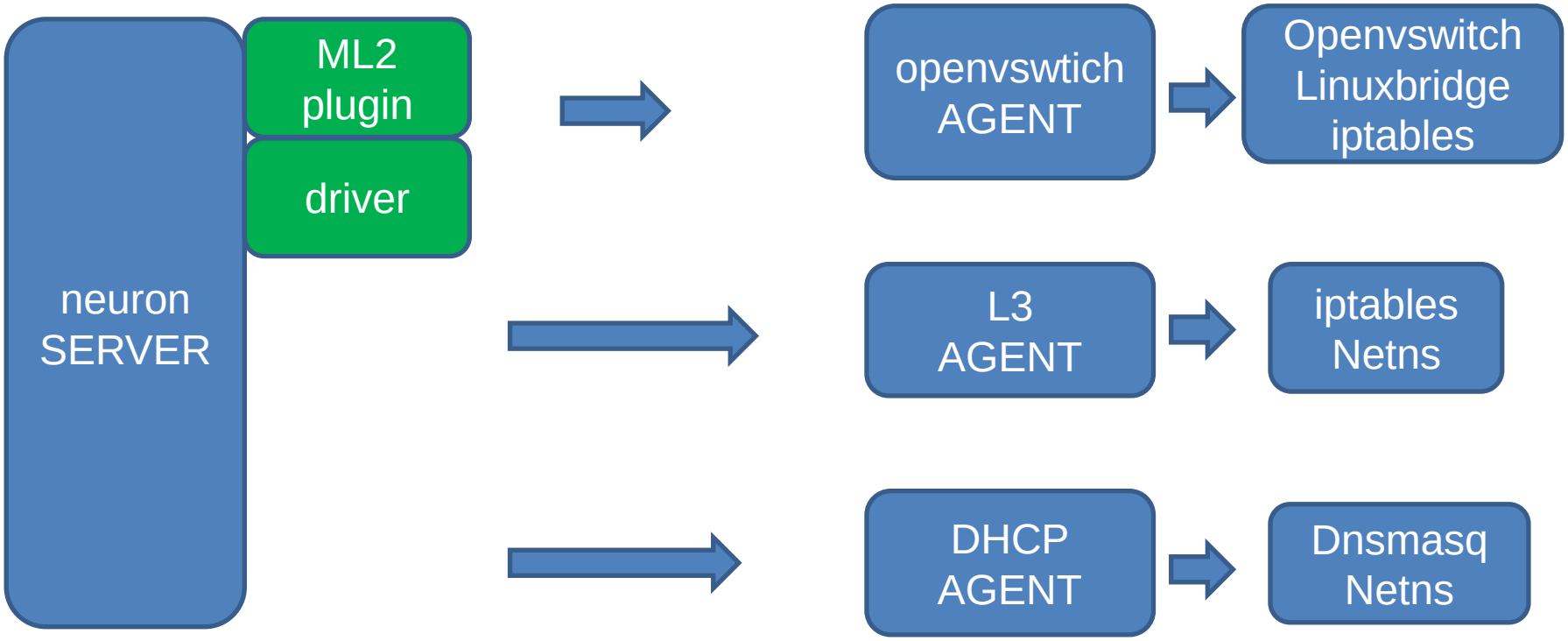
neutronは以下要素で動作する

- 設定を保存する**database**
- APIで命令を受けて処理を渡す**server**
- openvswitch等の設定を作成して起動させる**agent**
- Serverとagentの命令の橋渡しをする**AMQP**(neutronに限らず使用)

※赤枠範囲がOpenStackのソフトウェア



neutron　構造-2



機能毎に**agent**は分かれている
pluginで機能を追加可能
driverでopenvswitchやvxlanの使用を指定する

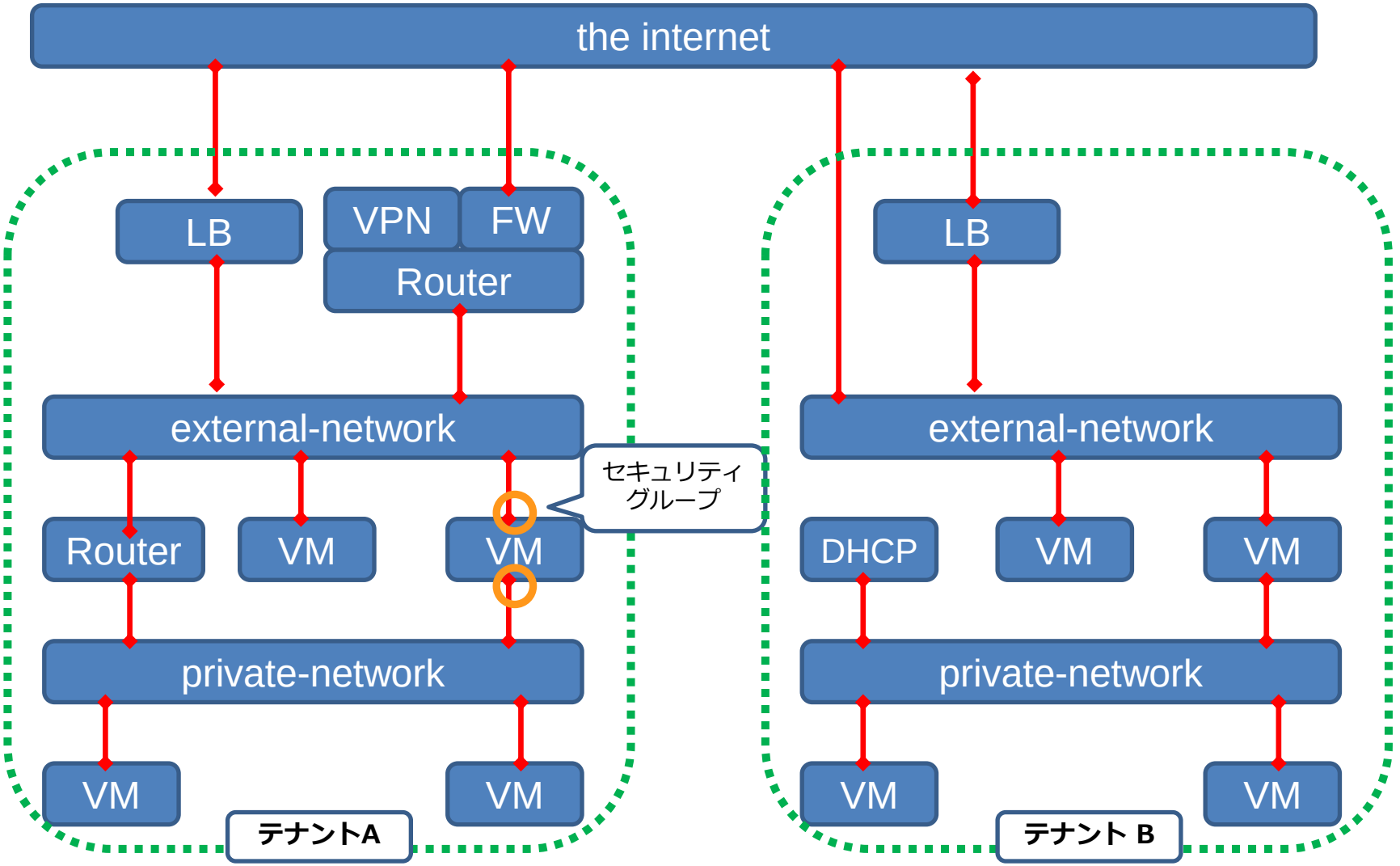
OpenStack　何が出来ますか？network

neutronはユーザ（テナントという単位）毎に**独立したネットワーク環境を作成して**以下のネットワークサービスを提供出来る

・ネットワーク接続(vlan,vxlan,greをサポート)
・DHCP
・セキュリティグループ(VMのポートに適用するフィルタ)
・ロードバランサ

・ルータ(セグメント間接続、nat、VPNゲートウェイ、ファイヤウォール用の限定された機能)
└VPNゲートウェイ
└ファイアウォール

OpenStack　何が出来ますか？networkイメージ



neutron　動き

OpenStackの各ソフトは**あくまで構成管理ツール**

- 主にlinuxに実装された機能を**設定・操作**する
- 従って設定した機能は例えOpenStackサービスが落ちても機能し続ける

ネットワークサービスを受け持つ**neutron** は標準的なインストールでは

- ネットワーク接続　: openvswitch&linuxbridge
- パケットフィルタ　: iptables&ipset
- ルーティング　: iptables&network namespace
- DHCP　: dnsmasq&network namespace
- ロードバランサ　: ha proxy

を**設定・操作**するツールとして動作する

neutronで設定可能なこと

CLIで　　#Neutron help で表示される内容が全て、
と思ってしまえばいい

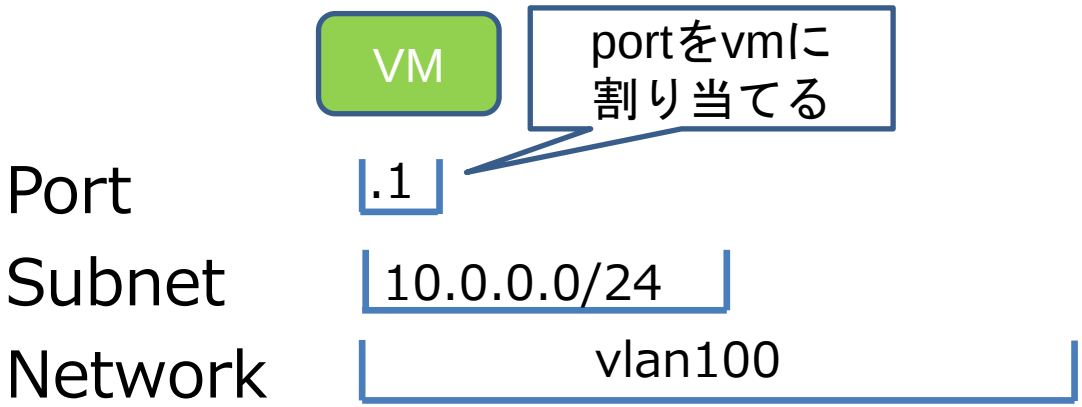
ex)
(keystone_admin)]# neutron net-create
(keystone_admin)]# neutron net-show
(keystone_admin)]# neutron net-delete
(keystone_admin)]# neutron net-update
(keystone_admin)]# neutron net-list
のようにずらりと出てきます　nova等も同様に確認出来ます

参考
http://docs.OpenStack.org/cli-reference/content/neutronclient_commands.html
http://developer.OpenStack.org/api-ref-networking-v2.html

リソースの概念　ネットワーク接続の例

Network　　：vlan ,vxlan,gre等の**入れもの**
Subnet　　：**入れもの**の中で使えるIPアドレスの範囲指定
Port　　：VM等が使用可能な**IPアドレス**と**ポート**

ex)Portを割り当てることでVMやルータは通信可能になる



操作機能

OpenStackのリソースはmysql ,MariaDB等のデータベースに格納されている　従ってOpenStackの操作は**ほぼデータベースの操作**

操作は「作る、読む、更新する、消す」**(CRUD)**を意識すればよい

ex)ネットワークリソースに対して以下の操作が可能

Create　：作成
Show　：詳細表示
Delete　：削除
Update：修正
List　　：一覧表示

操作パターン

GUIとCLIはAPIをたたいている　動作に違いは無い

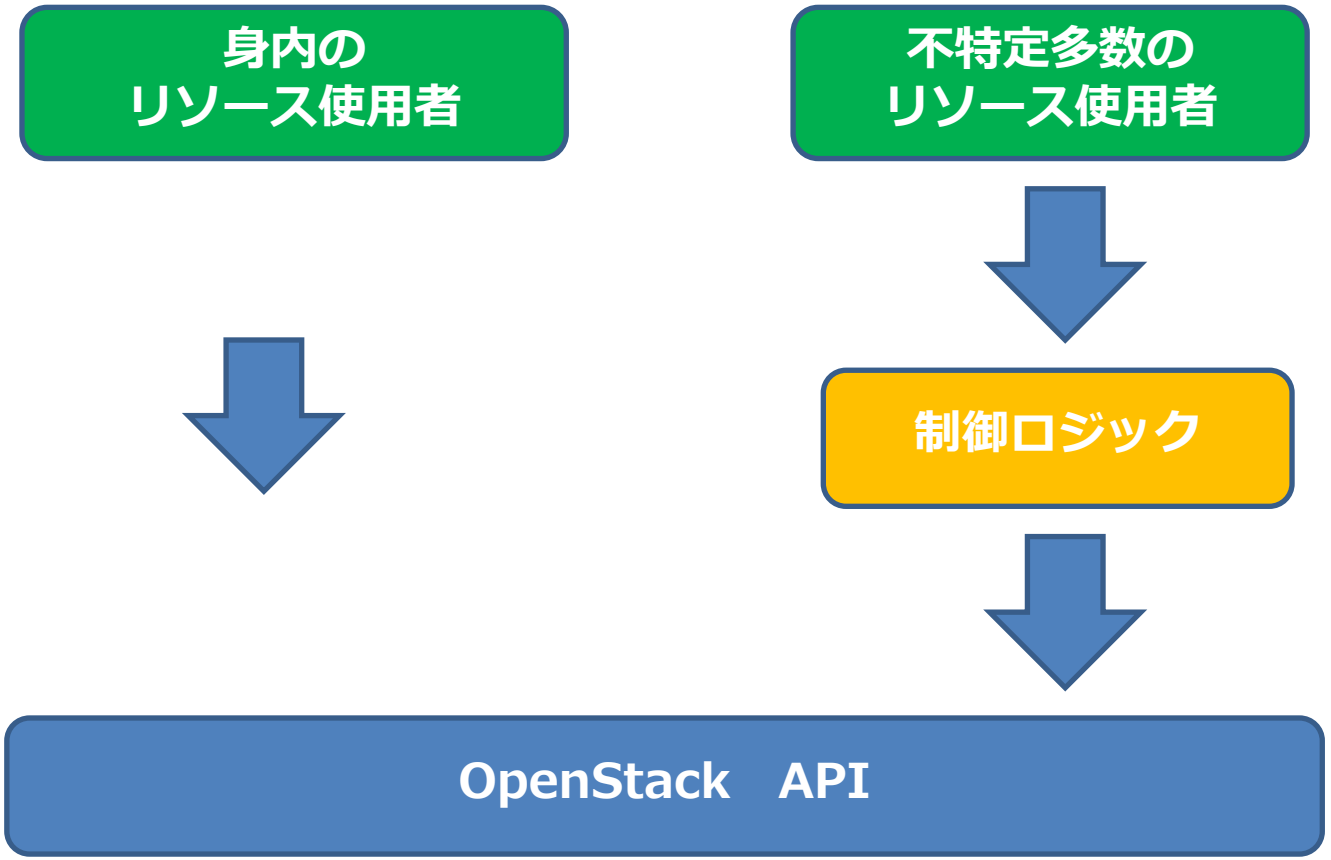
・GUI(horizon):設定出来ない機能もある。主に確認用

・CLI：ほぼ全ての設定が可能。最初にこれで動作を見るとよい

・API：プログラムからの操作用に提供

※操作したらOpenStackが設定の動作確認すると理解が深まる
ex)ネットワークやポートを作成後に
#ovs-vsctl show等 でopenvswitchの状態を確認

リソース使用者が誰かによって作りは異なる



サービス開発の流れのポイント

1. OpenStackでサポートされている機能をドキュメントで確認する

2. CLI,APIで動作確認

3. 実際の機能の動作確認

4. コードの修正や制御ロジックでの機能制限や拡張

5. 負荷試験

#まともに動かない前提で確認をすること

開発に必要なリソース

プログラムの知識
バグにあたったときに修正できないと詰んでしまう
コードの追加修正がサービスの可能性を広げられる

サーバ（linux系）の知識
OpenStack自体がlinuxの機能を使うことが多いため

時間と人
情報収集と動作確認とバグつぶしでたっぷりと

周囲の理解
オープンソースを使用したサービス開発はつまづきながらやっていく

OpenStackな人達と接して

OpenStackに先行して携わっていた人達はサーバやネットワーク
というジャンルの垣根を意識していない

むしろ積極的にVMを繋ぐ必要性があったりリソース管理の点で
ネットワーク（機器も含めて）もまとめて管理したがっている

不完全なものを扱っている為か
うまく動かないのは当たり前という認識で
どうやったら出来るか、を考える傾向が強い

クラウドサービスの開発の現場

ネットワーク機器はケーブルの延長のものと考えられるか
構成管理ツールから操作される存在という位置付け

そういう環境で一歩踏み込んでいくと
ネットワーク機器の設定だけしている訳にはいかなかったり

通常のネットワーク構築についても

ネットワークを使いたい人が使いたい
だけリソースを利用出来るようにして対価をいただくコンセプトで
ネットワークシステムを作っていかないと手作業運用が増えて苦しい

OpenStackを触った後に、その意識はより強く

そのシステム作りの為にはプログラミングや構成管理ツールの
知識が必要になってくる

最後に

ネットワークオペレーションのマンネリ化
BGPからL2スイッチの設定まで、ここ10年位大きな変化が無い

クラウドサービスの現場では製品・機器だけ触る
ネットワークエンジニアはあまり力になれなくなってきている

他のネットワークジャンルにも同じ波はいずれくるのではないか
波が来てから動くと苦しい

新しいことを学べてなかったり、面倒な作業をしていると
感じている人は違う分野に進出すると吉
OpenStackは色々と学べるのでよい教材なのでは...!?

苦しい部分はjanogのようなコミュニティで助け合い楽しくやりたい

宣伝

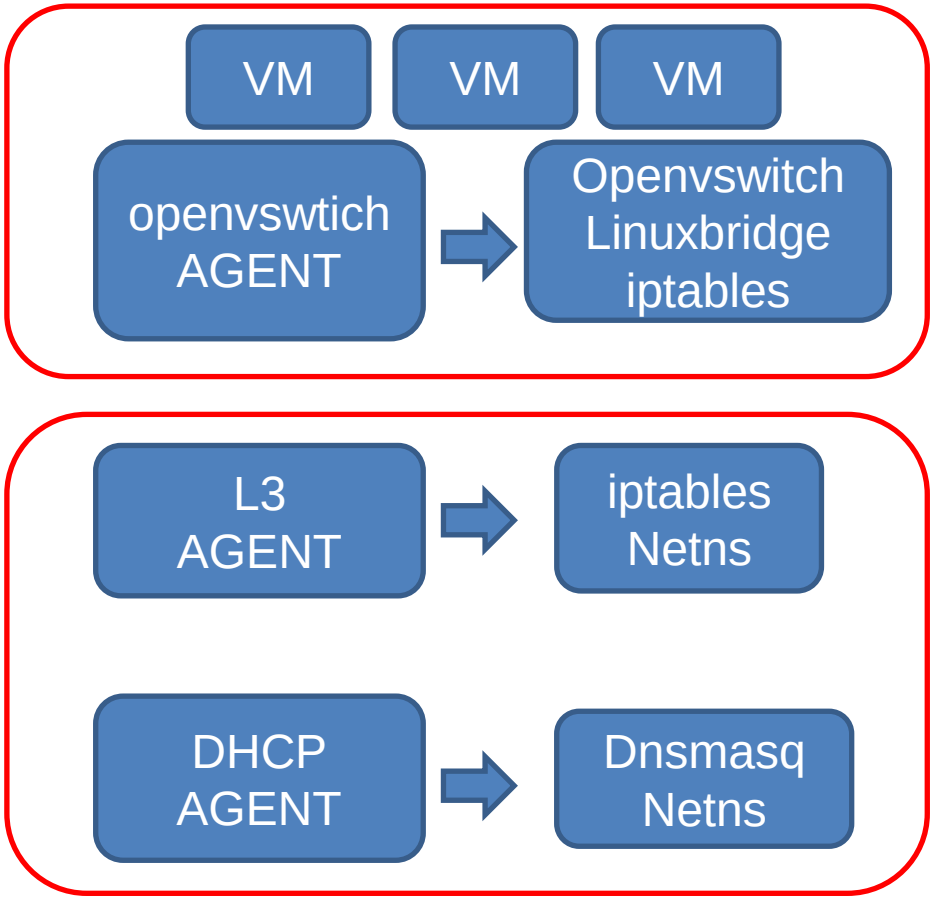
http://openstackdays.com/index.html

2015,2/3-4 グランドプリンス高輪B1F

以下参考資料

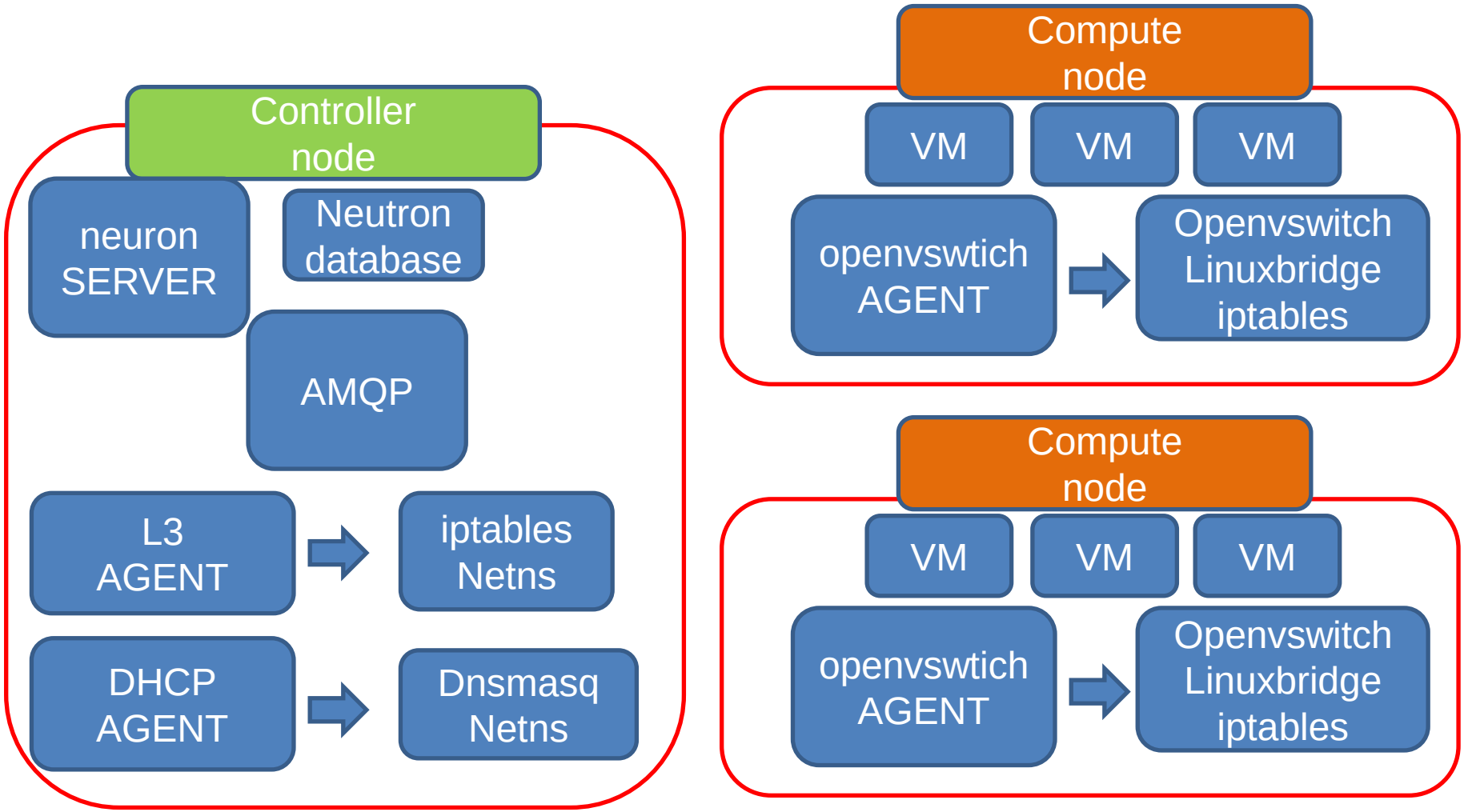
実践　インストール構成

実際にはVMを乗せるホスト
(**compute node**）に
openvswtich agentを入れて
L3 agentやDHCP agent等は
管理用ホスト(**controller node**)
に分けてインストールが可能

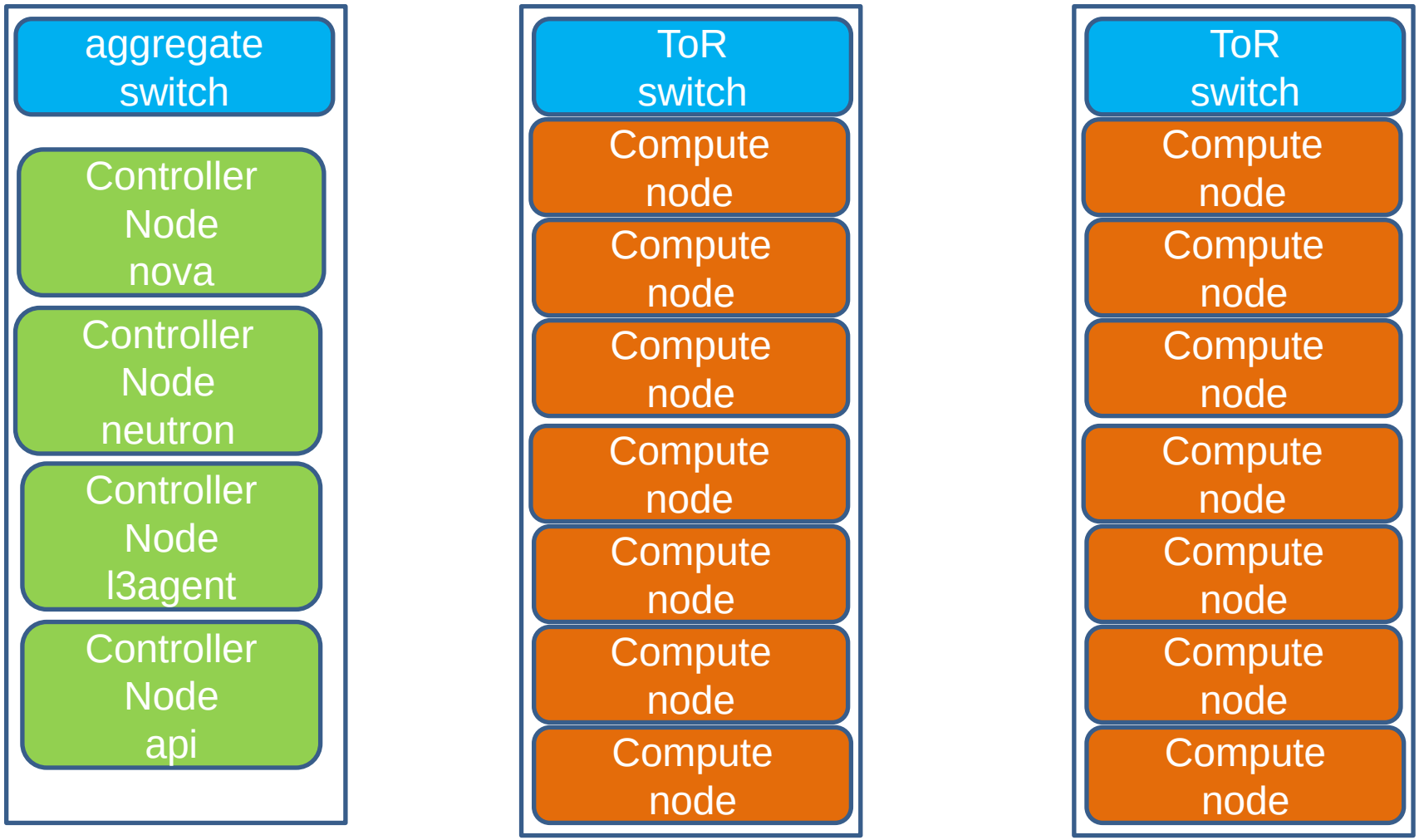


赤枠の範囲が１台のサーバ

実践　neutron関連の構成例



実践　物理構成例


aggregate switch
Controller Node nova
Controller Node neutron
Controller Node l3agent
Controller Node api
ToR switch
Compute node
Compute node
Compute node
Compute node
Compute node
Compute node
Compute node
ToR switch
Compute node
Compute node
Compute node
Compute node
Compute node
Compute node
Compute node

実践　環境を用意する

現在はRDOとpackstackを使えば比較的簡単に環境構築可能

https://OpenStack.redhat.com/Main_Page

1つのサーバでcontrollerとcomputeの役割を兼ねさせることも可能だが
Network系の検証をするならcontrollerとcomputeをnode分けて
ノード間通信の動作確認が出来るようにするのがお勧め

※メモリ16GB以上、nic2ポート以上のマシンでの構築を推奨
　　安定動作の為、サービスと管理用の通信を分ける為

実践　環境を用意する-流れ

1.RDO等でインストールファイルを用意

2.answerファイルでインストール構成を決める

3.各nodeで**/etc/neutron** 配下**の neutron.conf** や**plugin.ini**を修正

4. 各nodeでサービスを再起動する

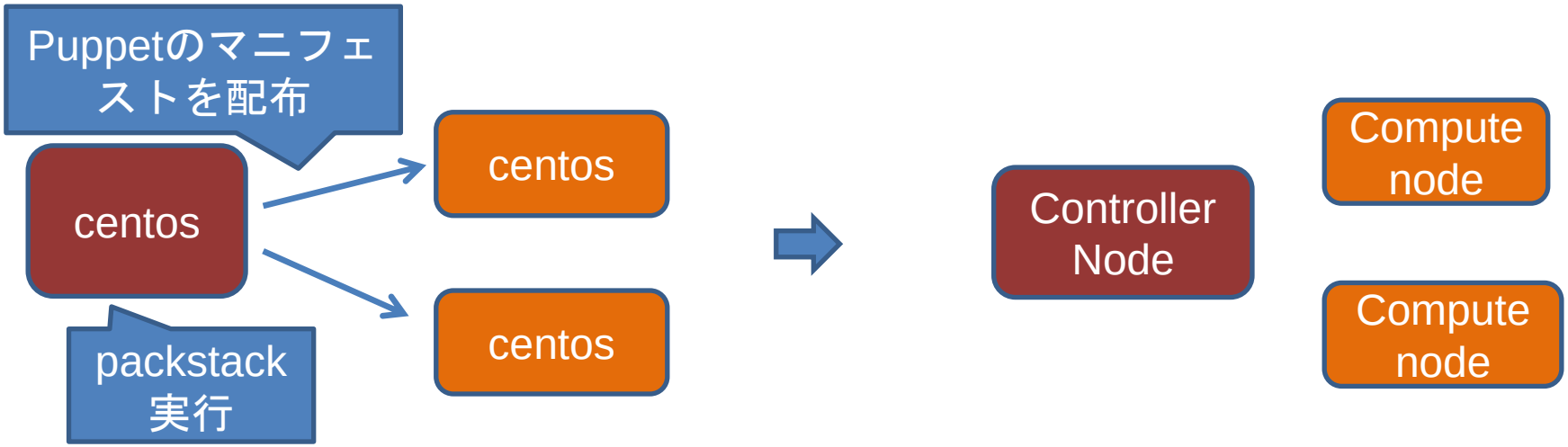
ex)systemctl restart neutron-openvswitch-agent.service　など

実践　環境を用意する-例

3台のマシンでマルチノード環境を用意する場合

1. Centos等をインストールして3台がお互いに通信可能な環境を用意する

2. answerファイルでcompute nodeのIPを指定すると構成管理ツールを利用してcomputeが作成される



実践 よく使うコンポーネント

ネットワーク関係の開発だと以下のコンポーネントを主に使用する

keystone　：認証を通過させて操作を可能にする
nova　：vmを作成管理する
neutron　：ネットワークサービスの動作確認を行う

実践 CLIでの設定例

ネットワークを作成してサブネットを割当てポートを作りVMにアタッチする
順番通り作成しないといけないという例

①# source keystonerc_test

②(keystone_admin)# neutron net-create ***testnet*** --provider:network_type ***vxlan*** --provider:segmentation_id ***100***

③(keystone_admin)# neutron subnet-create ***testnet 10.0.0.0/24*** –name ***testnet_subnet***

④(keystone_admin)# neutron port-create ***testnet***

⑤(keystone_admin)#nova boot ***testvm*** –image {**image uuid**} –flavor {**フレーバ uuid**}

⑥(keystone_admin)# nova interface-attach --port-id {**port uuid**} ***test-vm***

実践 CLIでの設定例の説明

入力コマンドの説明

①ユーザ：testの認証情報をファイルから読み込み、CLI経由でOpenStackのapiを操作出来るようにする

②***testnet***という名前で***vni100***の***vxlan***の**ネットワーク**を作成する

③ネットワーク***testnet***にsubnet名 ***testnet_subnet*** サブネット***10.0.0.0/24***を割り当てる

④***testnet***上で**ポート**を作成する　IPを指定しないと空いているIPが割り当てられる

⑤vm:***testvm***を作成

⑥作成したポートをvm:***testvm***に割り当てる

実践　見たほうがいいもの

更新頻度が高いので定期的に確認するとよいです

launchpad
https://launchpad.net/neutron

今後の予定(blueprint)やバグ情報

github
https://github.com/OpenStack/neutron

ソースコード plugin/agentのメソッドを見ると何が出来るかだいたい分かる

実践　よく出てくる単語集

uuid　:vmやportやnetworkなどOpenStackが管理する
オブジェクトに割り当てられる重複しない長い識別子

json　:apiでOpenStackに命令を渡すフォーマット (似:xml)

token　:user/pass認証すると渡される鍵　jsonに埋め込んで
apiを叩くときに使う

endpoint　:api操作の為にアクセスするurl

curl　:cliでapiを叩ける便利なツール　確認でよく使う
linux系OSにほぼデフォルトで入っている

実践　知っておくといいこと

phpMyAdmin等でOpenStackのデータベースをweb管理出来るようにすると切り分けの際に便利

AMQP（RabbitMQ)は処理が重くなりがちで、トラブル多発地帯

vmの通信が出来ないトラブルの盲点でデフォルト設定される
Spoofing防止用のパケットフィルタがある port-updateで変更可能

ソースファイルは以下に存在するので適宜確認修正することになる
/usr/lib/python2.7/site-packages/neutron

/var/log/neutron　配下のメッセージはトラブル時にまず確認

実践　依存関係に注意

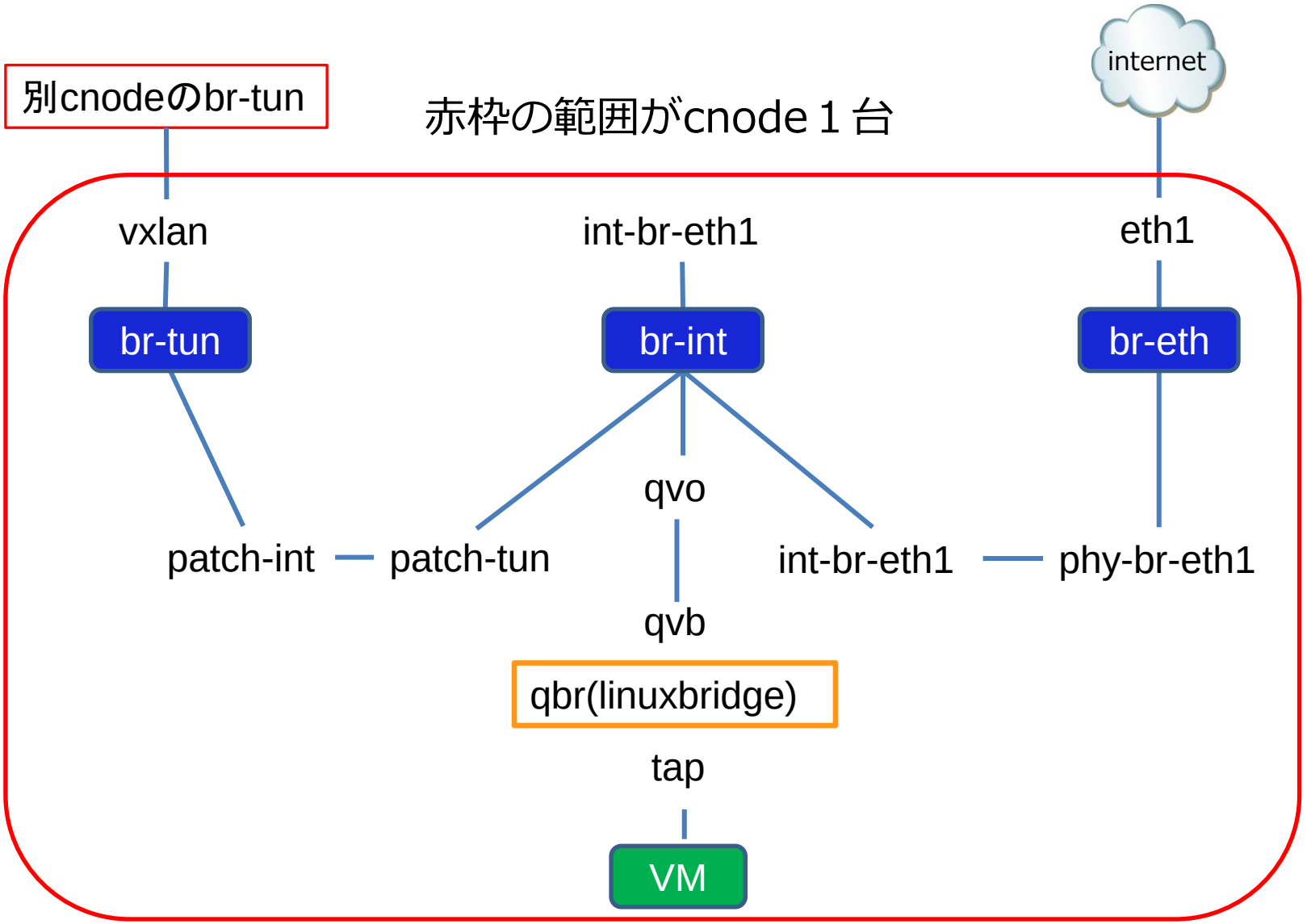
バージョンアップは**難しい**

＝＞データベースのテーブル構造変更や各ソフトの依存関係の為
バージョンアップの間隔が早く機能追加が多いけど…
ex) junoで作成した環境はシステム終焉までjunoで使い続ける
ことになる

システムに組み込んだものは使い続けることになる

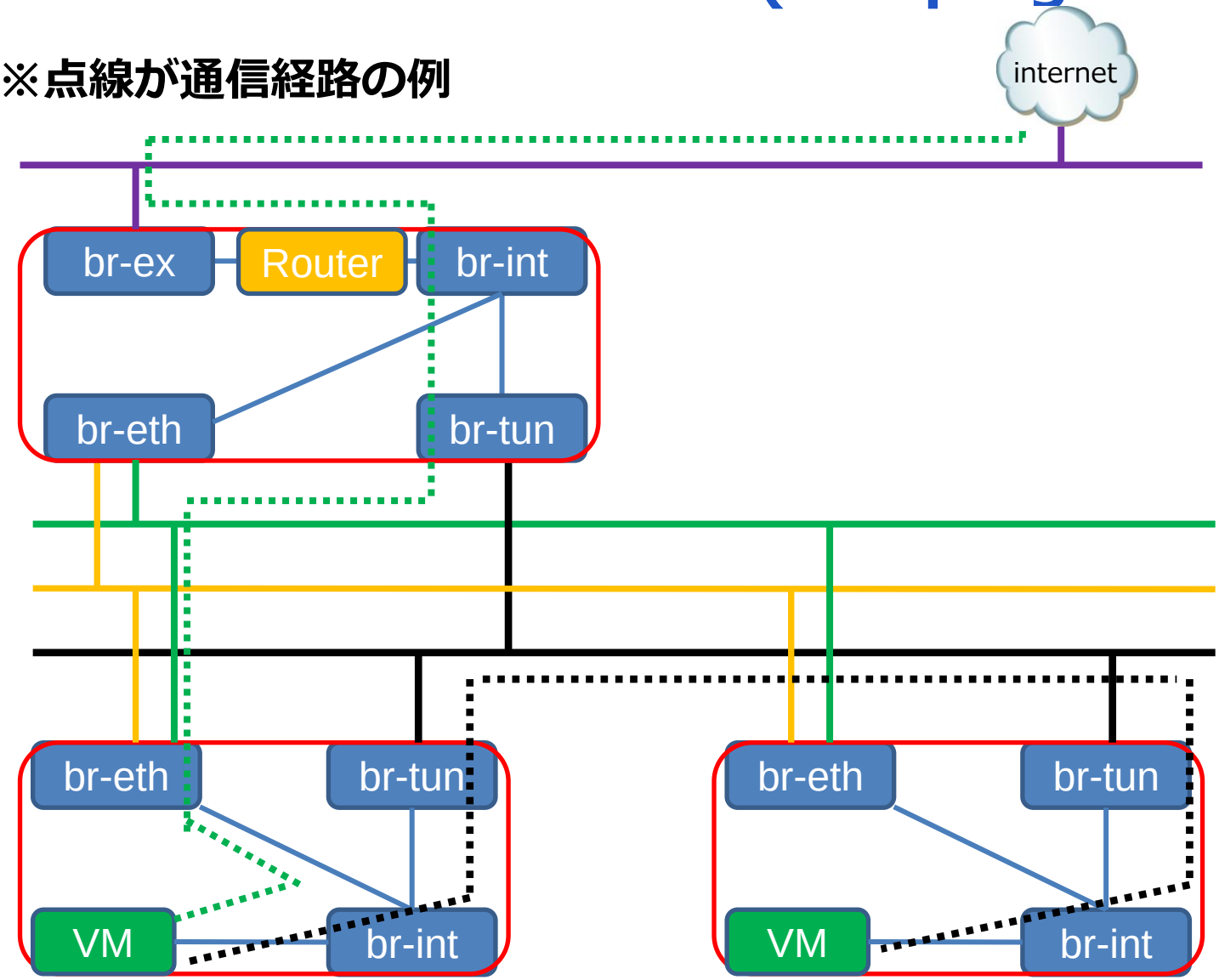
＝＞ベンダpluginやライセンスは**ロックイン**になってしまう可能性

cnode内の内部ネットワーク(ovs plugin使用時)



cnode間のネットワーク例(ovs plugin使用時)

※点線が通信経路の例



vlan1003:
203.0.113.0/24
public用

vlan1002:
198.51.100.0/24
public用

vlan1001:
172.16.0.0/24
vlan　private用

vlan101:
10.0.0.0/24
vxlan　overlay用

ovs plugin使用時のなかみ

vmへのネットワーク提供はvlanと、vxlan,greを使用した
オーバーレイネットワークが提供可能

今の仕様ではルータを使用すると、ルータの存在するnodeを通るため
ボトルネックとなってしまう。グローバルIPセグメントを割り当てた
vlanをvmに提供して**ルータを介さず**アクセスさせる等の方法も可能

iptablesのフィルタを適用する為にlinuxbridgeも使われている

用途別に**br-ex**(外部接続),**br-int**(vm接続),**br-tun**(tunnel),**br-eth**
(vlan接続)の４種類のovs-bridgeが作成される

flowtableを　ovs-**ctl コマンドで追ってくと構成が掴めます