Openstack 導入事例

ヤフー株式会社

システム統括本部基盤システム開発本部

インフラ技術1部リーダー　伊藤 拓矢

2014年2月14日（金）

YAHOO! JAPAN

アジェンダ

- 自己紹介
- OpenStack導入に至る経緯、その理由
- 導入システム構成
- OpenStackを選択した事で困った事/苦労した事
- Openstackを採用する上での注意点
- Openstackでの悩み事、望む事
- 今後について

自己紹介

- ヤフー株式会社（2008年新卒入社）

  入社以来インフラ、主にネットワークを担当

  2010年から IaaS開発に片足を突っ込む

  2013年には IaaS開発に両足を突っ込む

- 現在の業務内容

  検証、設計、開発、運用、障害対応、部外交渉、予算管理、

  教育、物理から仮想までなんでも

OpenStack導入に至る経緯

- 2008、2009年ころ
  - VMの利用はこの頃から開始
  - VMは次Qに利用する数をサービス担当者が申請
  - 申請数に応じて予算化し物理サーバを取得
  - CLI管理ツールからVMを管理

VMの提供まで3か月ほど掛かる

開発環境のみ 数百VM程度の運用

OpenStack導入に至る経緯

- 2010、2011年ころ
  - VMを管理しきれなくなる
  - WebUIからサービス担当者がVMを作成できるシステムの開発着手
  - CLI管理ツールをWebでwrapした感じの実装
  - 物理サーバと同じインストールプロセスを踏む
  - 仮想用イメージの作成をしなくても済む

開発環境のみ 数千VMの運用

リソースオンデマンドのための環境を達成

OpenStack導入に至る経緯

- 2010、2011年ころ WebUIによる下記の機能
  - リソースを事前取得してVMを作成(申請してリソースを得る)
  - ロードバランサの設定(VIP)
  - GSLBの設定
  - Ciscoスイッチのコンフィグ変更(ACL)
  - Volumeストレージの設定(動的attach、detach)
  - DNSの自動設定と任意の設定
  - リソース情報の提供
  - WAF、IDSからセキュリティ情報の提供
  - 社内システムとの連携(構成管理、アカウント管理)

OpenStack導入に至る経緯

- 2012年

  技術的に古くなりパフォーマンスも良くない

  APIは独自のため、OSSとの連携が悪い

  コンポーネントの依存が強く、VMのbootまで時間がかる

  運用工数が上がり、リソース不足の中、新規開発が出来ない

このままでは人がインフラを意識しなければならない状況が続く

開発環境とプロダクションでの運用 数万VMの運用

OpenStack導入に至る経緯

- 2013年

  Openstack、CloudStackの検証を始める

  社内システムとの連携が必須のため、改良し易いOpenstackを選択

  ベーシックな機能は問題なく、すぐに導入を決定

  旧来の機能の割り切りも行う

  APIもサービス担当者は利用可能になり、Jenkinsなどとも連携

  夏頃、開発環境で提供を開始

  プロダクション環境でも提供を開始

１万VMほどが稼働中

900プロジェクト、4000ユーザ

OpenStackで配信されるコンテンツ

既にこの環境使ってサービスの一部の機能を提供している

ショッピング、ヤフオク、知恵袋、トラベル、不動産、
ブックストア、ゲームなど

OpenStackを選択した理由

- コードを読めば動きが分かる
  - 検証において、事前に仮定が立てやすい
  - よって導入目標、工数分析しやすい
  - さっと見て導入する機能か、見送る機能か判断できる

例えば、havanaのhorizonで使えるようになったリサイズ、マイグレーション

sharedストレージを使わないでマイグレーションさせようとすると、

「sshしてディレクトリ作る。rsyncやscpする」

- nova-computeの権限でする
- セキュリティ的な制限を加えたい
- 導入までにはちょっと工数かかる
- sharedストレージの導入を行うモチベーションへ

```
migrate_disk_and_power_off
if not shared_storage:
        utils.execute('ssh', dest, 'mkdir', '-p', inst_base)
--------------------------------------------------
def copy_image(src, dest, host=None):
 try:
execute('rsync', '--sparse', '--compress', '--dry-run', src, dest)
    except processutils.ProcessExecutionError:
        execute('scp', src, dest)
    else:
        execute('rsync', '--sparse', '--compress', src, dest)
```


YAHOO! JAPAN

OpenStackを選択した理由

- 必要な物とバージョンの判断が容易に行える
  - HVのパッケージ構成を決める

例えば、ファイルシステム拡張や、ファイル挿入において

「どんなモジュールが選択されるか」

- guestfsがあれば使う
- guestfsがなければ、diskフォーマットが rawならloopbackマウント
- それ以外ならnbdを使う

```
class VFS(object):
    def instance_for_image(imgfile, imgfmt, partition):
        hasGuestfs = False
        try:
            importutils.import_module("guestfs")
            hasGuestfs = True
        if hasGuestfs:
            return importutils.import_object(
                "nova.virt.disk.vfs.guestfs.VFSGuestFS",
                imgfile, imgfmt, partition)
        else:
            return importutils.import_object(
                "nova.virt.disk.vfs.localfs.VFSLocalFS",
                imgfile, imgfmt, partition)
```


YAHOO! JAPAN

OpenStackを選択した理由

- 開発が活発
  - 多ベンダーが開発に参加していて、アプライアンスなどの選択枠が非常に多い
- 致命的なバグは少ない
  - 検証した上で採用すれば問題ない
- オープンで多くの企業が参加できる
  - OSSの活性化は喜ばしい

Openstack全体を共に盛り上げていきたい企業を募集

導入システム構成

- 利用しているOpenstackのコンポーネント

  Keystone, Nova, Neutron, Cinder, Glance, Horizon

- Openstackを動かすために

  mysql, qpid, rabbitmq, qemu, kvm, linuxbridge, centos, nginx

- 大きく手を加えた所

  config driveによる社内システム情報の挿入(アカウント、構成管理)

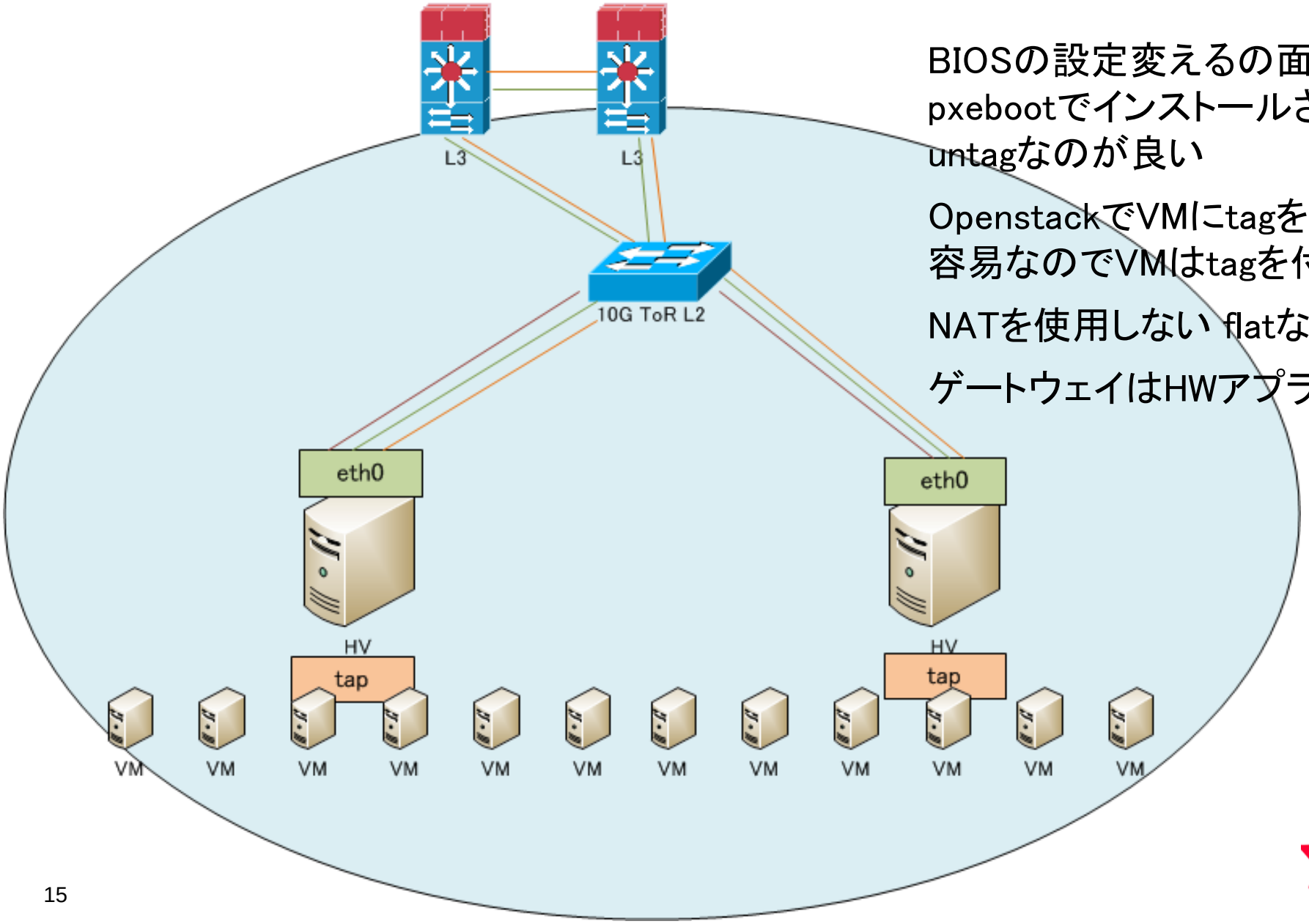
  (ホスト名、アカウント、公開鍵、IPアドレス、GW、ntp.conf、

  resolv.conf、sudoers、group)

導入システム構成

- 構成のポイント
  - 安定して稼働できる
  - パフォーマンスが出る(CPU, IO, Network)
  - 内部統制を実現してしっかり守る仕組み
  - 誰がいつ作成したのか、作成後、即座にポリシーの強制を開始
  - 運用できること(必要なロックは掛ける)
- 使っていない機能
  - Openstackに登録した公開鍵の配布機構
  - FloatingIP

導入システム構成図 VM net



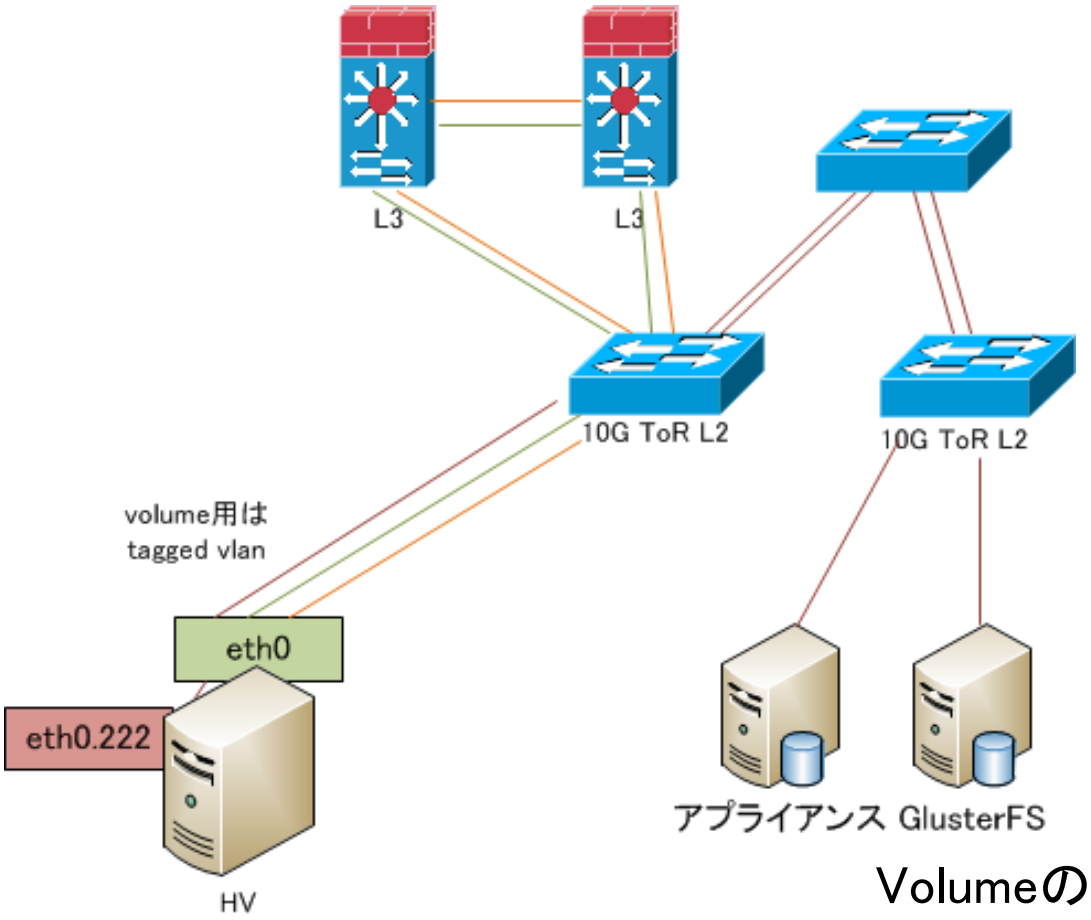
BIOSの設定変えるの面倒なので、pxebootでインストールされるHVはuntagなのが良い

OpenstackでVMにtagを付けるのは容易なのでVMはtagを付ける

NATを使用しない flatなnetwork構成

ゲートウェイはHWアプライアンスに

導入システム構成図 Volume



Volumeの提供はToRSWから
枝分かれしたネットワークで行う

1G時代はToRを2つ置き、物理配線を分けていた
ので昔の名残という側面も

導入システム構成 config

コンポーネント全てに言える事として、sqlalchemyまわりのチューニングは必須

[database]

idle_timeout=480

min_pool_size=10

max_pool_size=40

max_retries=300

retry_interval=2

max_overflow=60

grizzlyのquantum-serverにはこの設定項目が無いので、pythonのライブラリの初期値を直接書き換える必要があった

導入システム構成 config Nova

timeout値の緩和、定期タスクの実行間隔を延ばす

report_interval=60

service_down_time=120

rpc_response_timeout=120

neutron_url_timeout=90

sync_power_state_interval=600

heal_instance_info_cache_interval=120

host_state_interval=300

NeutronやCinderなども同じような項目があるので、延ばします

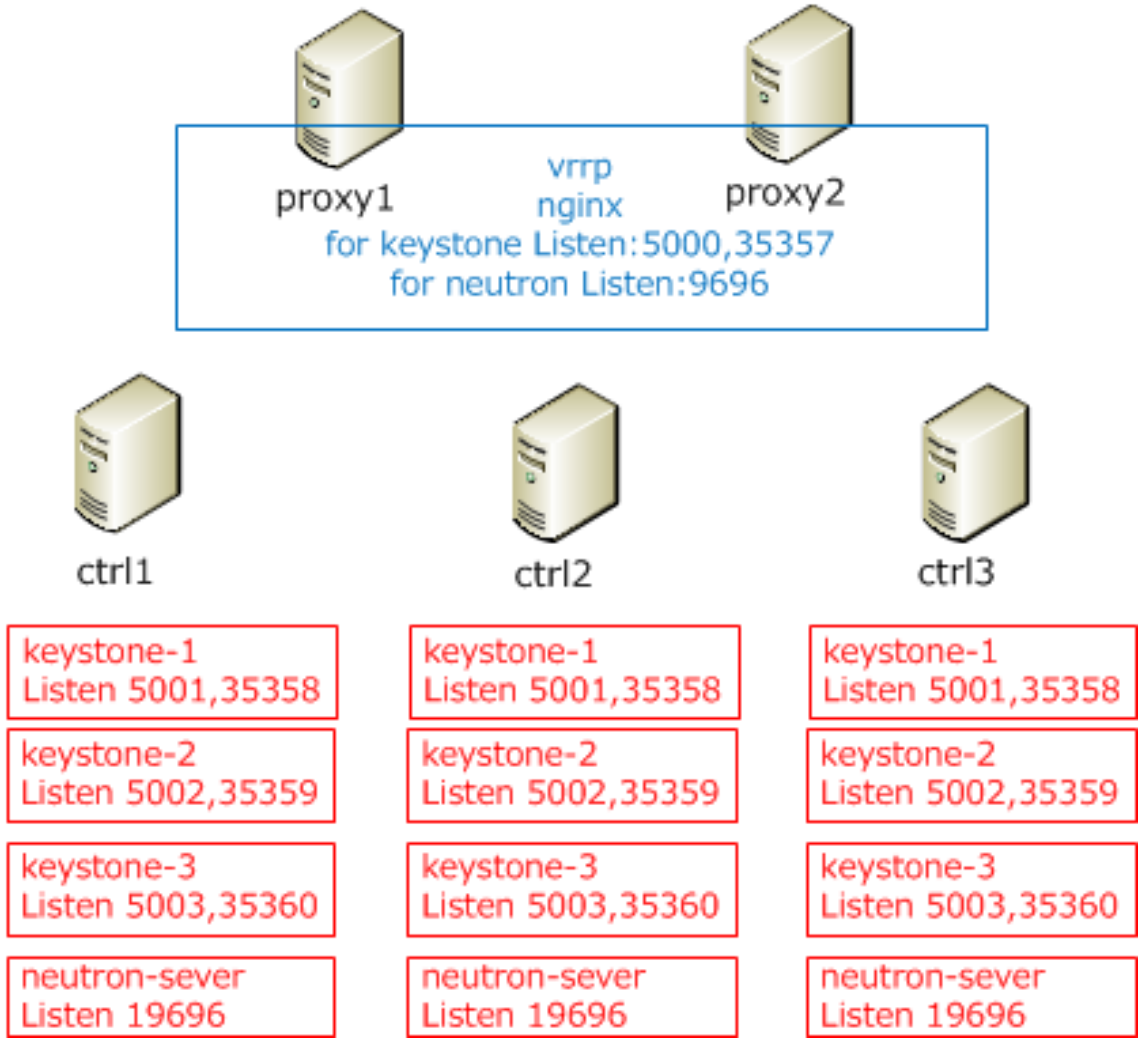
コンフィグドライブを常に利用する(後述)

force_config_drive=always

RPCに極力負荷を掛けないようにチューニングしていくと、本来の動きが正しく、早く動く

導入システム構成 プロセスの並列化

- APIはnginxを挟んで複数プロセスを立ち上げる



endpointはnginxに向ける

nginxのupstreamとして下の
コントローラの各ポートに
ロードバランスさせる

keystone9つ、neutron3つ

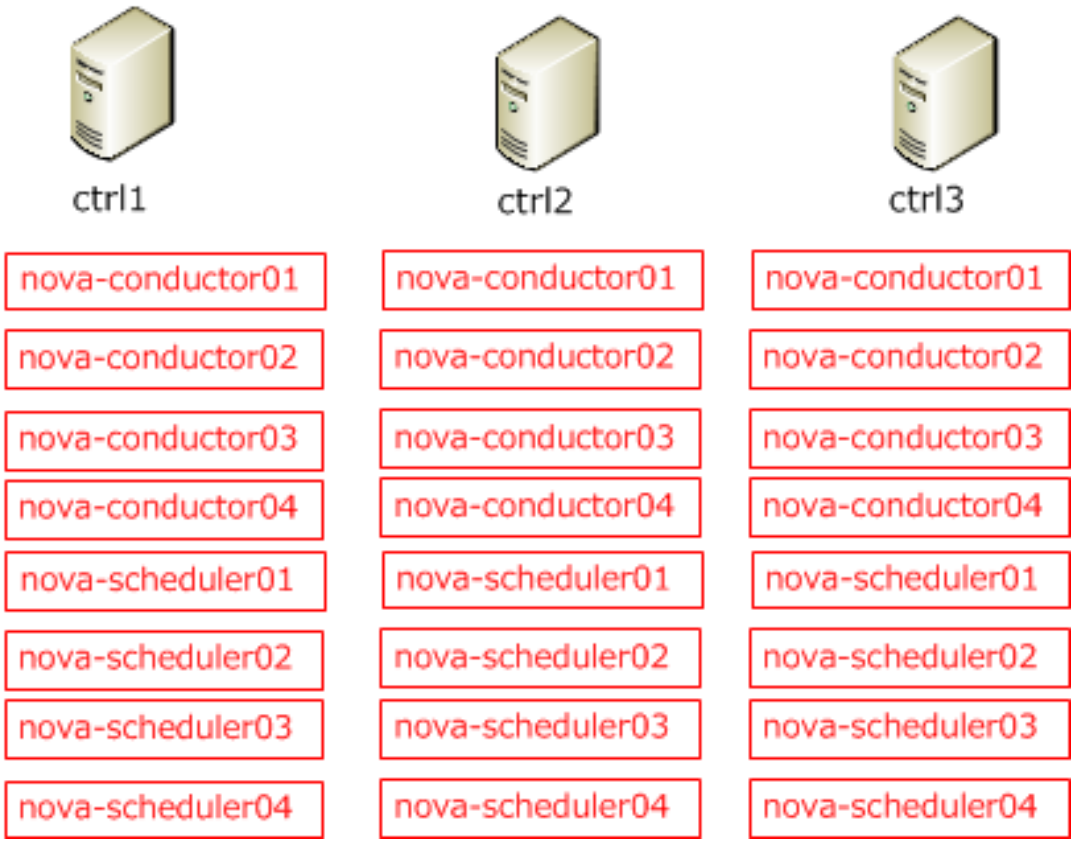
本番の運用は大体こんな感じ

haproxyがリファレンスか

YAHOO! JAPAN

導入システム構成 プロセスの並列化

- nova-scheduler, nova-conductorは増やす



この2つに余裕がないと
VMのbootは遅くなる

ただし、conductorのamqp
に対するセッション数は多
いため、amqpのセッション
数は増設後しばらく見張る

導入システム構成 プロセスの並列化

- 理由として、パフォーマンスの問題が大きい
- シングルプロセス、シングルスレッドで動くためプロセスのCPU使用率が100%に張り付く
- havanaではマルチスレッドで動くものが増えた
- computeノードの増加に合わせて、プロセスを増やすのが良い
- 同じコンフィグで問題ないので増やすのは楽
- mysqldのパフォーマンス測定も忘れずに

導入システム構成 config_drive

- 肝となるシステム

cloud-init と独自のinitスクリプトで拡張まわりの実装

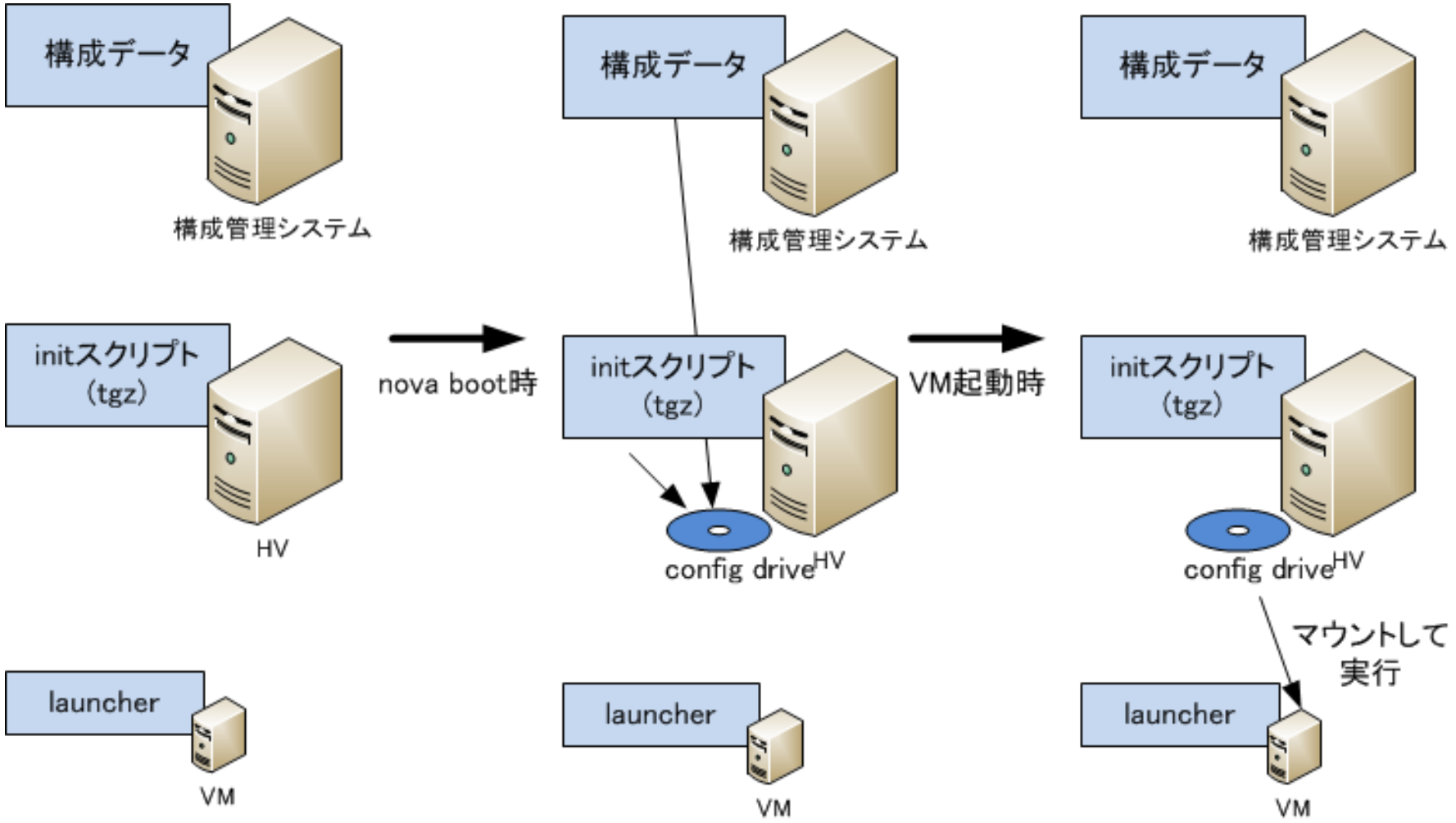
VMのイメージには、ランチャーのみ配置
ランチャーはconfig_driveに付属するスクリプトを実行する

スクリプトは構成管理データを読み込んで、ファイルの配置変更を行う

頻繁に更新が入るために、スクリプトの実体はHVに配置

VM起動時点で既に、構成管理システムと接続された状態になる

導入システム構成 config_drive



Openstackの開発と管理

- 社内のgithubとJenkinsを利用してUpstreamと独自コードのマージ
- Jenkinsによる自動テスト、自動ビルド
- QA環境に持っていって、展開、テスト
- Openstack+LBの動作テストといったアプライアンスとのテストはまだまだ手でやる
- 今のところは構築、運用に工数を取られる状況が続き、仕様変更をここで知る・・・

OpenStackを採用した上で困った事/苦労した事

- ロードバランサなど、IaaS用途で使いたい機能面の弱さ、設計の甘さ
- amqpの高い稼働率、起こる不具合原因はほぼここ
- Upstreamを追いかけるのが大変・・・
- 仕様がよく変わりますね

Openstackを採用する上での注意点

- 自由すぎる設計
  - 選択枠が多すぎて、何を使うかで悩んでしまう
  - 選択枠の中で、組合せを間違えると動かない
  - コンポーネント間でコンフィグを合わせなければいけないことも多い

ある程度の設計思想を持った上で、検証を開始すると良い

無邪気にチョイスしてみるのも面白い

Openstackを採用する上での注意点

- リファレンスモデルと実際の採用モデルが結構変わるのは当然
  - なんちゃらstackで、ひとまず試してみる
  - 小規模であれば問題なく動く
  - 大規模はひたすらチューニングとの戦い

設計を決めたら、腹をくくる

DBとログに情報は集まるので、ブレなければ大丈夫

Openstackでの悩み事、望む事

- Openstackクラスタに依存しないHV上のVMの移動
  - openstack-vmexport のようなコマンドで、HV上のVM全てを別の環境に持って行ける(nova, neutron, cinderなどまるまる)
  - openstack-vmimport でそれらのデータをまるまる取り込み

Openstackでの悩み事、望む事

- HVのデータ消失時のVMの復旧
  - 7000台くらいHVが有ると、2か月に1台くらいデータ消失が発生
  - HV復旧後にVMをイメージを入れた直後の状態でユーザに戻してあげたいが、現状は難しい
  - novaのデータベースにはVMが存在しているが、実際のHVには無い状態
  - nova deleteすると、fixed ipが解放されてしまう
  - nova replay-bootのようなもので、nova boot直後の状態を再現したい

Openstackでの悩み事、望む事

- Openstackクラスタ自体のマイグレーション
  - このOpenstackのクラスタ、いつまで運用すれば良いのだろうか
  - リリースに併せて、HWとOSをバージョンアップしたいが、Openstackの自律した動き以外の事をさせるのが難しい
- やはりHVに依存する望みが多いです

Openstackでの悩み事、望む事

- 今回話した内容がベストだと思えない
  - 時間的な制約の中、大規模実装を行ったため、今後ベストな実装を検討していく
- スケールさせた事例が出てこないため、自分たちで試行錯誤した結果が今回の内容
- 幸い複数クラスタが有るので、毎回コンフィグを変えて試す事が多い

ナレッジ共有したい

今後

- Neutronまわりもっと検証
  ベンダーAPIとの連携
  オーバレイNW(パフォーマンスと管理性の問題解決)
  様々なNWレイヤの要求の対応
- Ironic導入
  VMの柔軟性や近年のオーバヘッドの極小化を加味しても、物理サーバの利点は大きい。TripleOにも使える
- OSS連携の強化
  開発の効率化、テストの自動化、本番環境展開の高速化
  良いツールを試していく

今後

- Openstackクラスタを越えてのイメージの共有
  - 開発環境クラスタでsnapshot取って、そのままプロダクション環境クラスタへ イメージを持っていく仕組み
  - 数%のトラフィックだけ試してみたいといったサイエンス用途には良い
- インフラ抽象化の推進
  - 使いたい時に使えるようになった今、大規模環境でエンジニアが、インフラを意識せずともサービスを行える環境作り
  - 開発に集中できる環境を作っていきたい