実環境での運用自動化とその管理方法

日本仮想化技術
野津　新

2017.7.20

自己紹介

- 野津 新 (のつ あらた)
- 日本仮想化技術 技術部

- OpenStack: Diablo(2012)〜
  - nova-baraemetal(Ironicの前身)の開発
- とあるプライベートクラウドプロジェクトでのAnsibleを用いた構築・運用自動化の経験を中心にお話しします

アジェンダ

- プロジェクトの概要
- AnsibleによるOpenStackデプロイ
- Ansibleプレイブックの自動テスト

プロジェクト

- プライベートクラウド
- 複数のOpenStack(DC)を運用
- computeノード数: 各DC50～500

- Ansible以前の構築
  - 手動 → openstack-chef → juju
- 100ノードの検証環境構築時にAnsible使用
- プレイブックは0から作成し現在も利用中(2014年～)
- 0から作成したことにより、理解を深めながら使っていくことができた

運用自動化のための構成要素

- 構成管理DB – ホスト情報の管理
- Git – プレイブックの管理
- Gerrit/CI – テスト自動化
- Ansible

Ansible

- プレイブック: 実行したい内容をタスクとして記述
  - タスク:コマンドの実行,パッケージのインストール,設定ファイルの書き替え,...
  - YAML形式
- インベントリ: プレイブックを実行する対象を管理
  - ホスト
  - グループ ← プレイブックとホストを結びつける
  - ホストやグループに変数を定義:同じプレイブックで異なるホストを扱う

- SSHでホストにログイン、タスクを順番に実行

プレイブック

- サービスごとにプレイブック作成
  - keystone
  - glance
  - nova-api,nova-scheduler,nova-conductor ← まとめる場合もある
  - nova-compute
  - ...
- 最小限のタスク、標準的な設定からはじめる
  - 手作業の自動化
  - 典型パターン
    - パッケージインストール
    - 設定ファイル更新
    - サービス再起動

nova-compute.yml(1/2)

```
- hosts: nova-compute
  become: yes
  gather_facts: no
  tasks:
  - name: install service packages
    apt: name={{ item }}
    with_items:
        - nova-compute
        - sysfsutils  # for systool
        - libguestfs-tools  # for file injection

  - template: src=template/etc/nova/nova-compute.conf dest=/e
    notify: restart nova-compute

  - name: override exec to read /etc/nova/nova-compute.conf only
    lineinfile:
      dest: /etc/init/nova-compute.override
      line: 'exec start-stop-daemon --start --chuid nova --exec /usr/bin/nova-compute -- --config-file=/etc/nova/nova-compute.conf'
      regexp: '^exec '
      create: yes
    notify: restart nova-compute
```

nova-compute+依存関係に入っていないが必要なパッケージ

nova-computeはnova-compute.confだけを読むようにコマンドを変更。nova.confが、別にあるnova-api用プレイブックとの2重管理になるのを避けている。

nova-compute.yml(2/2)

```
...

 - name: see if virbr0 (defaut libvirt bridge) exists
   stat: path=/sys/class/net/virbr0
   register: virbr0
 # Checking ansible_interfaces does not work since the bridge is created
 # after facts ware gathered.

 - name: remove virbr0
   shell: virsh net-uuid default && virsh net-destroy default && virsh net-undefine default
   when: virbr0.stat.exists

 handlers:
 - name: restart nova-compute
   service: name=nova-compute state=restarted
```

libvirtのデフォルトブリッジ(virbr0)があれば消す。

confが更新された時はnova-computeを再起動

デプロイ

- サービスに依存関係(デプロイ順序)がある
  - ex. MySQL ← 各サービス
- 全体をまとめるプレイブック: site.yml
  - 正しい順序で各プレイブックをinclude
- 1コマンドで全サービスをデプロイ
  - ansible-playbook -i <inventory> site.yml

独自パッチの管理方法？

- 独自にソースに手をいれたい場合がある
  - バックポートされていない変更
  - プロジェクト固有の問題への対処
- debパッケージ作成が正攻法だが維持が大変だった
  - パッケージ作成が難しい
  - aptリポジトリ管理が面倒
- 通常のパッケージをインストールしてからファイルを差し替える(patch,copy)ことに落ち着いた

インベントリ

- DCごとのインベントリ
  - 構成管理と連携(タグ=グループ)
- DC固有部分を変数化

```
external_network:
  cidr: xxx.yyy.22.0/23
  gateway_ip: xxx.yyy.23.254
  allocation_pool_start: xxx.yyy.22.1
  allocation_pool_end: xxx.yyy.23.249
```

```
external_network:
  cidr: xxx.yyy.48.0/23
  gateway_ip: xxx.yyy.49.254
  allocation_pool_start: xxx.yyy.48.1
  allocation_pool_end: xxx.yyy.48.249
```

インベントリ管理:変数の検証

- プレイブックの更新で必須の変数を追加する場合がある
- インベントリ側での定義漏れ/誤りチェックが必要

- Ansibleのfilterとして作成してみた
- site.ymlの冒頭でチェック実行
  - assert: that={{ hostvars[inventory_hostname] | validate_vars }}
  - 変数が定義済みか
  - 型が正しいか
  - 古い形式を使っていないか: 新形式への移行を促す
  - 知らない変数が定義されていないか: 廃止された or TYPO

Bootstrap

- BIOS(UEFI),BMC,RAID設定
- OSデプロイ
  - MAAS:ベアメタルとVMに両対応
  - cloud-init
  - DNS登録、SSH鍵配布 => Ansible Ready

運用

- 日々の更新が必要
  - 脆弱性対応
  - 設定変更
  - バージョンアップ
  - イメージ更新
  - 新サービス/新機能導入
  - 監視強化

- プレイブックのアップデートと実行
- ユーザへの影響を考慮

ユーザ影響

- データの喪失
- インスタンスorネットワークのダウンタイム
  - 基本NG
  - 依頼:落として/再起動して/消して
  - ライブマイグレーション
- インスタンスのパフォーマンス低下
- APIのダウンタイム,パフォーマンス低下
  - ローリングアップデートで0を目指す
- 仕様変更

テストのための環境

- Staging
  - 利用HW,NW機器がProductionと同一
  - Productionよりも小規模
- Development
  - Sandbox
  - セルフサービス
  - VM & ベアメタル
- CI
  - 自動テスト用の使い捨て環境を提供

Production適用までの流れ

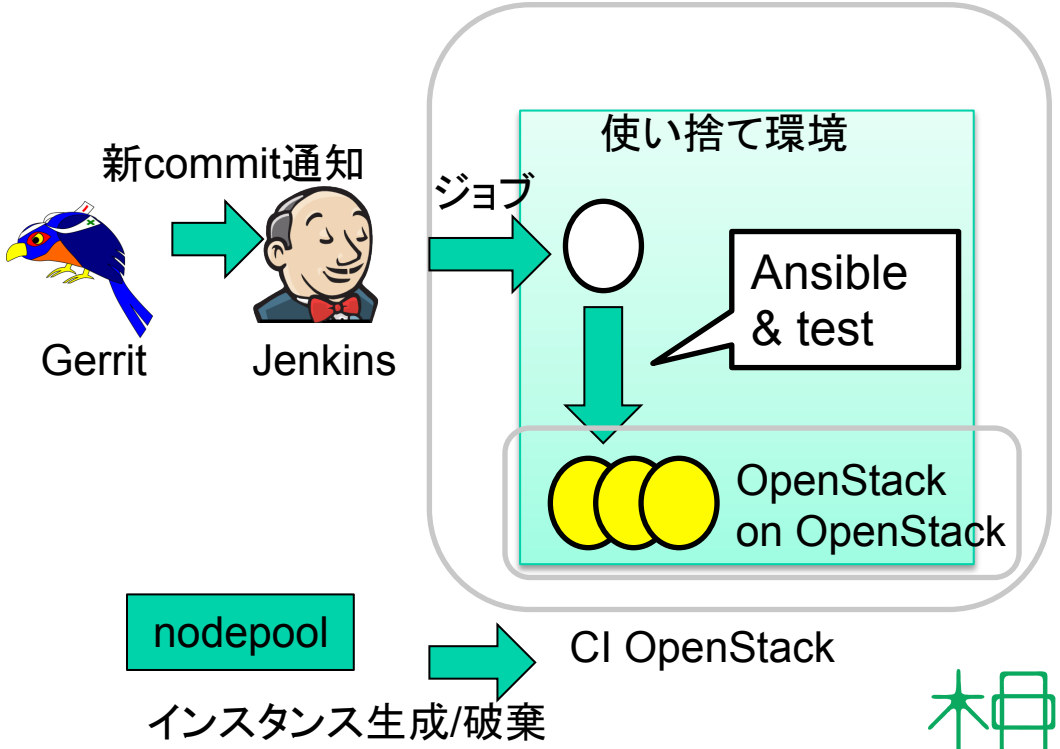
- チケット作成
  - 要求内容、変更動機、実装方法検討の記録
- Development
- CI + コードレビュー
- Staging
- Production
  - check mode
  - 範囲限定して適用&テスト

レビューシステム

- Gerrit(レビューシステム)にパッチを投稿
  - 担当者によるレビュー
  - 改善のセルフサービス化？
- 投稿されたパッチを自動デプロイ&テスト(CI)
  - 投稿者・レビュアーにフィードバック
  - 自力でテストする必要性を軽減

CI

- 使い捨て環境を作成(プール)
- インベントリ生成
- プレイブック実行
- 動作確認
- 運用系プレイブックの動作確認
- 使い捨て環境の削除



CIの結果確認

- プレイブックが全て実行できていること
- 成果物の確認
  - Ansibleログ
  - Tempest(rally verify)ログ
  - 各ホストの/etc,/var/log
  - 各ホストのリソース使用量
- アップデートCIの確認
  - HEAD^(=一個前のコミット)で実行、HEAD(=このコミット)でもう一度実行
  - 「このコミットで何が変わるか?」の確認
    - 変わるはずの箇所が変わっているか
    - 関係ない箇所が変わっていないか
- 静的チェック
  - ansible-lint: プレイブックの潜在的な誤りをみつける
  - flake8: 同python

Production適用までの流れ

- チケット作成
  - 要求内容、変更動機、実装方法検討の記録
- Development
- CI + コードレビュー
- Staging
- Production
  - check mode
  - 範囲限定して適用&テスト

標準テスト

- 変更中/変更後に必ず実施する
- 重大な影響が出ていないことの保証
- 全compute & networkノードの健全性確認
  - 指定ホストにインスタンス/ルータ作成&基本動作チェック
- 定期実行: Tempest,API動作監視

- 希望ユーザにはアプリケーションレベルの監視サービスを提供(内部監視)
- 開発中:各ノードにテストインスタンスを常駐させて死活監視

```
$ ./perhost_after.sh
Checking nova show
Checking nova console-log
Checking FIP
Checking volume
name        boot  log  dhcp  meta  login  ssh  volume  buildtime
X-host10    1     1    1     1     1      1    1       13
X-host9     1     1    1     1     1      0    -       11
...
```

変更内容に応じたテスト

- 作業前にコマンドベースでレビュー
- 基本的に一回きりなので自動化しない
- 有用なものは別途プレイブック/スクリプトにして標準化

現在抱えている問題

- 大規模化による不具合/障害(と思われるもの)が増加
- Production環境に限定される問題で再現試験が困難
- CIで大規模環境をシミュレート？

まとめ:自動化で得たもの

- 規模への耐性
  - 構築作業で数が問題になることはなくなった
- テスト可能性
  - 全自動化でテスト環境構築のコストが飛躍的に低下

- 構築自体は問題なし
- テストの充実が課題