

# Ansibleによる Infrastructure as Code入門

---

2014/12/17 kawasaki.rb #19

---

@kk\_Ataka

# 自己紹介

- Twitter: @kk\_Ataka
- GitHub: gosyujin



# アジェンダ

1. 構成管理ツールの長所/短所
2. **Ansible**の長所/短所
3. **Ansible**入門

# 話さないこと

- ➔ 本格的な**Ansible**の使い方
- ➔ **yaml**とは、**yaml**構文

# 対象者

1. 構成管理ツール何それな人
2. サーバの構成管理を手作業で行っている人
3. **Ansible**を使いたいなーと思っている人

# 構成管理ツールの長所/短所

# サーバの構成管理とは

1. サーバを調達し、必要な**MW, SW**などをインストールすること
  2. 設定ファイルを適切に編集すること
    - ➔ これらの作業を適切に維持、管理してくれるツールの事を「構成管理ツール」という
- ※ 「サーバが正しく稼動していること」の監視、確認は今回対象外

# サーバの構成管理の辛さ

- ➡ サーバが複数台構成になっている場合、
  - ➡ サーバ間で **一部を除き** 同一設定を維持しなければならない
  - ➡ 設定変更が発生した場合、全てのサーバにそれを適用しなければならない



# サーバの構成管理の辛さ

- ➡ 設定ファイルってきちんと管理なされていない印象...
- ➡ 日付管理 `xxx.conf` `xxx.conf.20141001` `xxx.conf.20141101` が多い...
- ➡ 手順(長い)があっても、それ手作業でやるの...?
- ➡ ダブルチェック？トリプルチェック？

そこで構成管理ツール

# 構成管理ツールの嬉しさ

- サーバ構築手順をコード化できる
  - **Infrastructure as Code**
- 何度実行しても同じ結果になる
- 複数のサーバに一発で環境構築できる
- コードなのでコードレビューもできる

代表的なツール(独断)として**Chef, Puppet, Ansible**などが挙げられる、今回は**Ansible**を使ってみようという話

# Ansible

- あんしぶる
  - 由来はハイニツシュ・ユニバースシリーズ<sup>1</sup>に登場する超光速通信技術
- Python製
- 基本理念は シンプル

---

<sup>1</sup>アーシュラ・K・ル・グウィン著

# Ansibleの長所/短所

# Ansibleの長所(構成管理ツールとして)

- ➔ べき等性(**Idempotency**)がある
  - ➔ 前述した、何度実行しても同じ結果になること

## 例

「``hoge.conf`` の最後に `"proxy=http://hoge..."` を追加する」という処理をシェルスクリプトでフツーに作ると、そのシェルスクリプトを実行するたびに `"proxy=..."` が追加されてしまう...  
(回避するための処理を書くのはけっこうめんどくさい)

べき等性があれば、何度やっても同じ結果に。便利！

# Ansibleの長所(構成管理ツールとして)

- 過去の資産を活用できる
  - シェルスクリプトで**Infrastructure as code**っぽいことをしていたなら、それを再利用できる
  - **Ansible**からシェルスクリプトをサーバへ送り、実行できる機能がある
  - 資産をそのまま流用するとべき等性はない<sup>2</sup>

---

<sup>2</sup> うまくやればできる `sh hoge.sh creates=/tmp/exist.txt` で**exist.txt**があればスキップ

# Ansibleの長所(競合ツールと比べて)

- ➔ python コマンドが実行できるサーバに**SSH**接続できればすぐ使える
- ➔ サーバ側に余計なツールをインストールする必要がない
- ➔ **Chef**などでは基本的にサーバにもエージェントをインストールする必要がある
- ➔ 必要がファイルが少ない
  - ➔ とりあえず**2**ファイルあればいい(後述)



# Ansibleの長所(競合ツールと比べて)

- 処理は `yaml` ファイル で書く
  - `Python`製だが`Python`を書く必要はない

競合ツールと比べてきわめてシンプル

# Ansibleの短所(構成管理ツールとして)

- ➔ 学習コスト...Ansible自体, Playbookの書き方...
- ➔ Ansibleの変更に追従していく必要あり
  - ➔ これはちょっと大変かも(ハマった)

何台のサーバに何回(どのくらいの周期で)使うか、それを手作業でやって生きていけるか...天秤にかけてみる

# Ansibleの短所(他の競合ツールと比べて)

競合ツールと比べてきわめてシンプル...とはいうものの

- 大規模システムの構成管理は苦手
- 複雑な処理も苦手

両方ともできないことはないけど、こんなときは素直に**Chef**などを導入したほうが良さ気

逆に小さな環境に**Chef**を導入しようとしたらかなり**Too much**かも...

# Ansible入門

# 登場人物

## 1. ホスト

- **Ansible** を実行するマシン
- **Python 2.6 - (Python 3 未対応)**

## 2. サーバ

- **Ansible** で環境を整えるマシン
- **Python 2.4 -**

# 実行するために必要なファイル

- inventoryファイル
- playbookファイル

# inventoryファイル

➔ ini形式で実行対象のサーバを記述する、変数も使える

```
[web]  
web01.example.com  
web02.example.com
```

```
[web:vars]  
ansible_ssh_port=20022
```

```
[db]  
db01.example.com
```

# playbookファイル

こんなファイル。

- **hosts:** all

  - sudo: yes

  - remote\_user: vagrant

  - vars:

    - username: newuser

  - tasks:

    - **name:** ユーザを追加するよ

      - user: name={{ username }} group=vagrant shell=/bin/bash



# playbookファイル 解説

大きく分けて**3**つのセクションに分けられる

→ **TARGET**セクション

→ **VARs**セクション

→ **TASK**セクション

→ モジュール

# playbookファイル1 TARGETセクション

どこにだれがインストールするか

- `hosts: all` # すべてのホストに
- `sudo: yes` # sudo使う
- `remote_user: vagrant` # vagrantユーザでログイン

# playbookファイル 2 VARSセクション

変数を指定する。**TASK**セクションで使用する

```
vars:
```

```
  username: newuser
```

# playbookファイル 3 TASKセクション

どんなことをするのかモジュールを使って記述する

tasks:

- **name:** ユーザを追加するよ # taskの名前、必須ではない

user: name={{ username }} group=vagrant shell=/bin/bash # モジュール

**VARs**で宣言した変数も使える

# playbookファイル 3 TASKセクション

- **user**モジュールを使って以下のユーザを追加している
  - ユーザ名は **newuser** (**VARs**の変数から)
  - グループは **vagrant** でログインシェルは **bash**

デモ

