openstack

Open source software to build public and private clouds.

# OpenStackとAnsibleによる ITインフラ構築・運用の自動化

2014-12-12

Hideki Saito
TwitterID: @saito_hideki
Internet Initiative Japan Inc.

version2.2

# 自己紹介

氏名: 齊藤 秀喜 (さいとう ひでき)

TwitterID: @saito_hideki

所属:

- 株式会社インターネットイニシアティブ
  プラットフォーム本部 システム基盤技術部/技術企画室
- 日本OpenStackユーザ会 ボードメンバー

仕事:

- IIJ GIO関連のちょっとした開発とちょっとした運用
- 次世代データセンタの実証実験

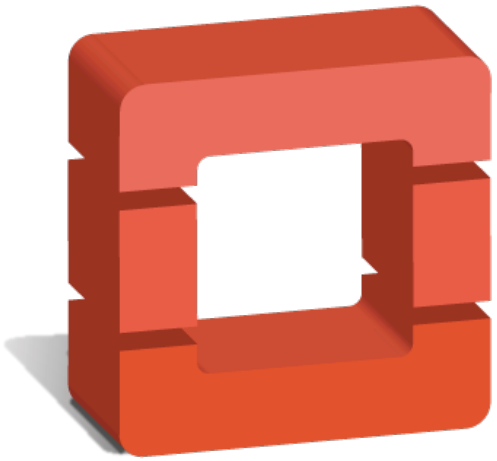
趣味:

- OpenStackとAnsible

# 目次

- はじめに
  - ✓ ITインフラ構築・運用管理の自動化
- ハンズオン前編
  - ✓ 本セッションで利用するシステム構成
  - ✓ Ansibleの仕組みと基本操作
- ハンズオン後編
  - ✓ OpenStackとAnsibleの連携

# はじめに



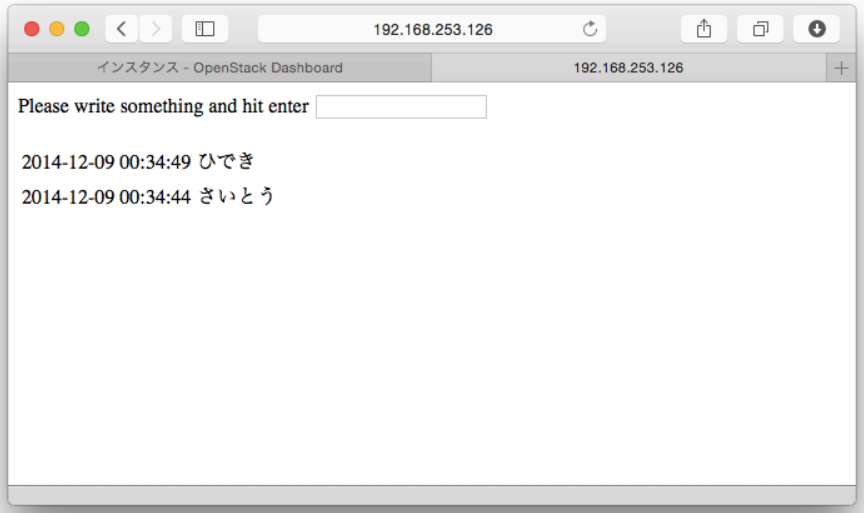
openstack™

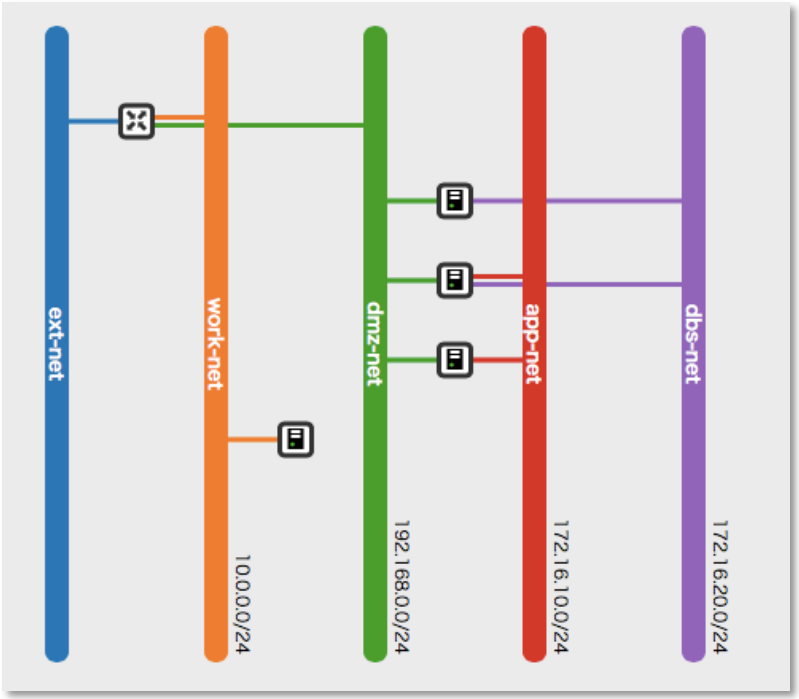
CLOUD SOFTWARE

# ゴール

このハンズオンでは、クラウド基盤(OpenStack)とオーケストレータ(Ansible)を組み合わせた、システムの構築・運用を、自動化する手法を紹介します。

前セッションでuserdataを利用して構築したSNSサイトの構築を、Ansibleを利用することにより自動化してみましょう。



# ITインフラ構築･運用管理の自動化(1)

- 自動化の目的
  - ✓ 日々繰り返される単純作業からエンジニアを解放する
  - ✓ 作業ミスを未然に防ぎ品質を安定させる

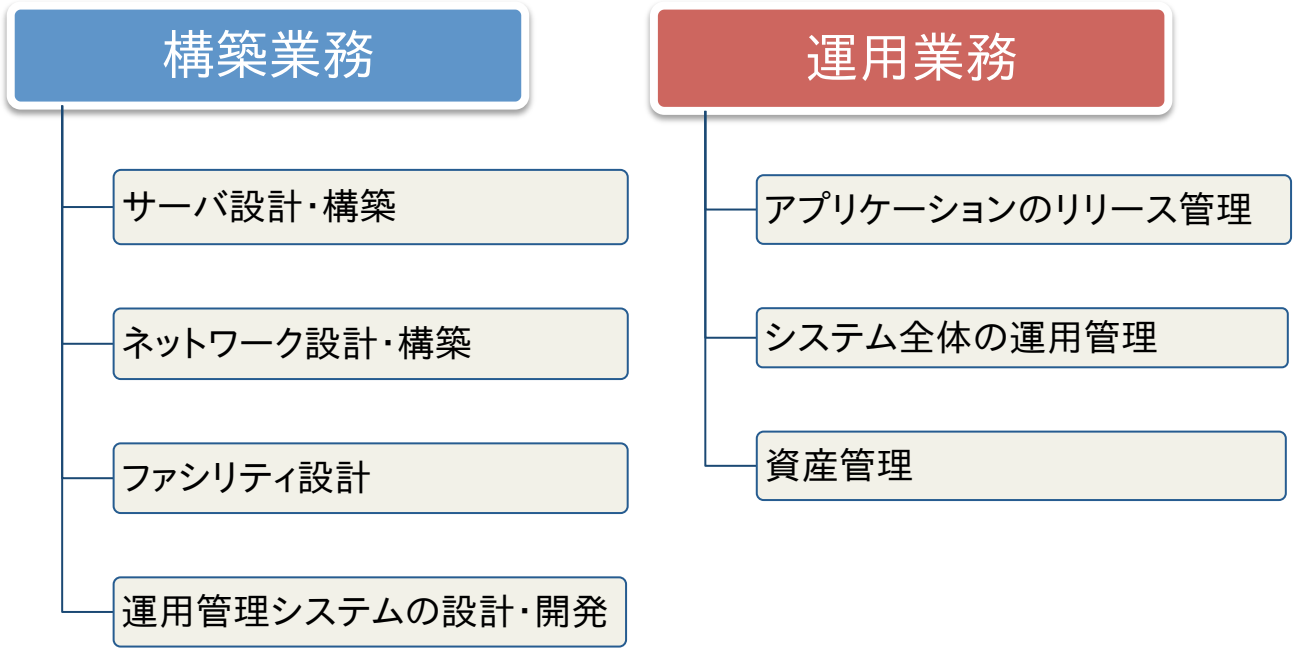
- 自動化の変遷


- クラウド/自動化/構成管理/オーケストレーション
  - ✓ 構成管理ツール: Chef, Puppet など
  - ✓ クラウド基盤: OpenStack/CloudStack/Eucalyptus など
  - ✓ オーケストレータ: Ansible, Kubernetes, OpenStack-Heat など

# ITインフラ構築・運用管理の自動化(2)

- 本セッションのテーマ

OpenStackとAnsibleを利用して、インフラエンジニアの日常業務である、クラウド基盤上の仮想リソース管理作業を可能な限り自動化する。



# ITインフラ構築・運用管理の自動化(3)

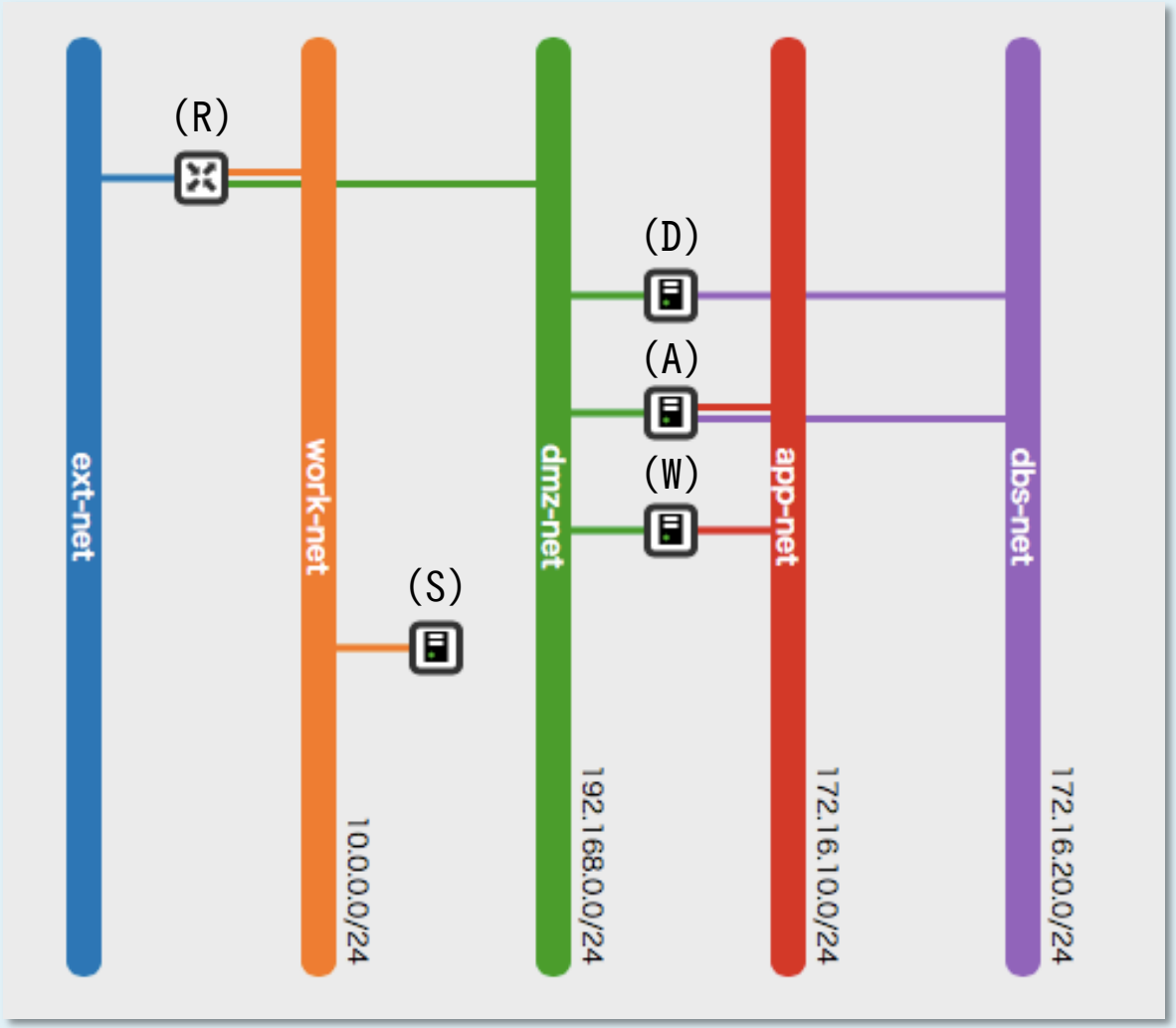
ITインフラエンジニアの典型的な業務は、このような感じでしょうか。

- ネットワーク機器を設定する
  - ✓ ポートを設定する
  - ✓ サーバのNICを物理的に接続する
  - ✓ wikiなどでドキュメントを更新する
- サーバを構築する
  - ✓ OSをインストールする
  - ✓ リモートログイン可能な状態にする
  - ✓ 追加パッケージのインストールや設定等の基本構築を行う
  - ✓ ミドルウェアのインストール＆設定を行う
  - ✓ サービスに投入する
  - ✓ wikiなどのドキュメントを更新する

以降の章では、このような日常業務を省力化する方法を考えてみます。

# ハンズオン前編

# 本セッションで利用するシステム初期構成




(R) Ext-Router
(S) step-server
(W) web01
(A) app01
(D) dbs01

# Ansibleの仕組みと基本操作

従来の構成管理ツールが苦手としていた、管理対象システムの変化に柔軟に対処できるオーケストレーションツール。
特定リソースだけでなく、ITインフラを構成する様々な要素を強調動作させることを目的に開発されている。

**特徴**

| キーワード | 概要 |
|---|---|
| エージェントレス | 管理対象ノードに専用エージェントを導入する必要がない<br>※Python2.4以降のランタイムが事実上必須 |
| 外部インベントリ | 専用の構成管理データベースを持たず、必要に応じて外部システムの構成管理情報を参照する方式を採用している |
| すぐに利用可能 | 多数のモジュールが標準で提供されている。 |
| シナリオ実行 | 多くの小さなタスクを1つにまとめることができる。さらに、タスクの実行結果による条件分岐や繰り返し処理などの制御構造も記述可能 |
| ドキュメントの充実 | 公式サイトのドキュメントが高品質で充実しており、ゼロからのスタートアップがしやすい |

# Ansibleの仕組みと基本操作

## Ansibleの主な構成要素

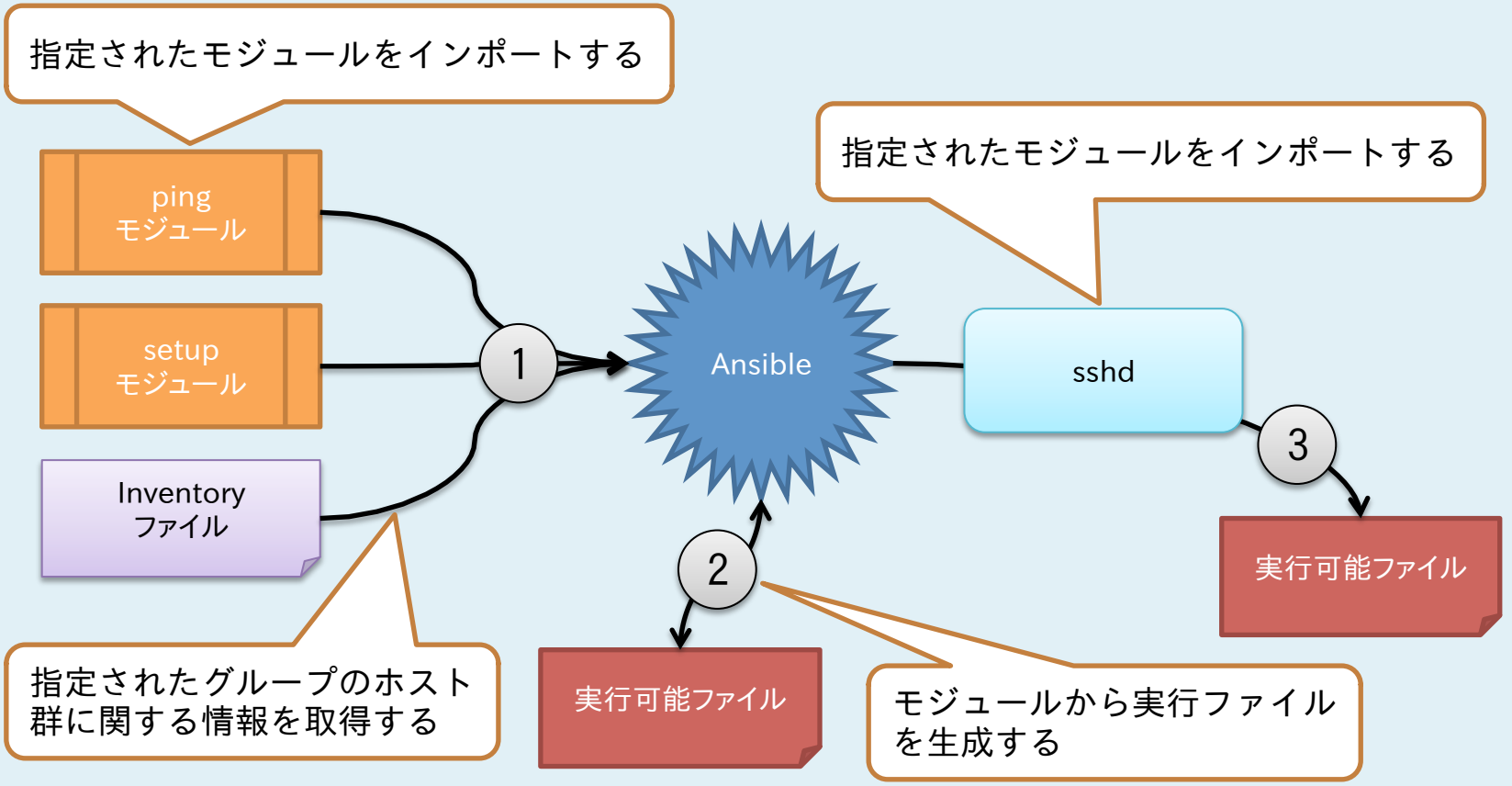
Ansibleは大きく以下の要素から構成されています。次章ではAnsibleを実際に利用しつつ、その仕組を解説します。

| | |
|---|---|
| module | • ファイルの転送、サービスの起動停止などAnsibleに行わせる作業がモジュールとして提供されている |
| Playbook | • Ansibleに行わせる一連の作業の流れをまとめたもの<br>• Chefではrecipe、Puppetではmanifestにあたる |
| Plugin | • 実行結果にり実行されるcallbackモジュール群<br>• 動的にターゲットホストをグルーピングするDynamic Inventoryプログラム |
| Inventory | • 操作対象ノードに関する情報を記録しているファイル<br>• PluginのDynamic Inventoryプログラムを利用して動的生成することも可能 |

# Ansibleの仕組みと基本操作

## Ansibleの挙動

ここまでのテストを例にAnsibleの動きを図示します



# Ansibleの仕組みと基本操作

■ UNIXホストに対する操作


■ Windowsホストに対する操作


■ Netconfを利用したネットワークスイッチ操作

# Ansibleの仕組みと基本操作

## ■ Ansibleのインストール(1)

● 事前準備

=> ansibleを利用するために必要となるパッケージを追加します

```
# export LANG=C LC_ALL=C
# yum install -y python-virtualenv sshpass
```

=> virtualenv環境に切り替えてOpenStackのクライアントライブラリをインストールします

```
# cd $HOME
# virtualenv venv
# source $HOME/venv/bin/activate
(venv)# pip install pbr cffi
(venv)# pip install ¥
python-cinderclient ¥
python-glanceclient ¥
python-keystoneclient ¥
python-novaclient ¥
python-neutronclient
```

# Ansibleの仕組みと基本操作

## ■ Ansibleのインストール(2)

● Ansibleのインストール

=> Ansibleが動作するために必要とするライブラリ群を、あらかじめインストールしておきます

```
(venv)# pip install jinja2 nose passlib pycrypto pyyaml bpython
```

=> Ansibleのインストールと動作確認

```
(venv)# pip install ansible
(venv)# ansible --version
ansible 1.8.2
  configured module search path = None
(venv)# ansible --help
... コマンドラインヘルプが出力されることを確認 ...
```

● pipでインストールされたモジュール群を確認するには

```
(venv)# pip freeze -l
abel==1.3
Jinja2==2.7.3
MarkupSafe==0.23
PyYAML==3.11
ansible==1.8.2
... 以下略 ...
```

# Ansibleの仕組みと基本操作

## ■ Ansibleの設定

●Ansibleの設定(ansible.cfg)を行う
=> Ansibleが利用する設定ファイル($HOME/.ansible.cfg)を作成します
(venv)# vi $HOME/.ansible.cfg

--- ここから(行番号は不要) ---

```
01: [defaults]
02: forks             = 10
03: log_path          = /var/log/ansible.log
04: private_key_file  = /root/key-for-internal.pem
05: host_key_checking = False
06: gathering         = smart
07: transport         = smart
```

--- ここまで ---

# Ansibleの仕組みと基本操作

## ■ インベントリファイル作成とpingモジュールによる動作確認

● テスト用インベントリファイルの作成

=> ローカルホストに対して操作を行うインベントリファイルを作成します

=> connection=localとした場合、ローカルホストでの実行となりSSH接続は行いません

```
(venv)# cd $HOME
(venv)# echo "localhost ansible_connection=local" > ansible_hosts
```

● pingモジュールによる動作確認

```
(venv)# ansible all -i ansible_hosts -m ping
localhost | success >> { "changed": false,  "ping": "pong" }
```

● ログファイルを確認する

```
(venv)# tail /var/log/ansible.log
2014-12-10 21:48:48,466 p=30091 u=root |
2014-12-10 21:48:48,466 p=30091 u=root |  /root/venv/bin/ansible all -i ansible_hosts -m ping
2014-12-10 21:48:48,466 p=30091 u=root |
2014-12-10 21:48:48,661 p=30091 u=root |  localhost | success >> {
    "changed": false,
    "ping": "pong"
}
```

# Ansibleの仕組みと基本操作

## ■ setupモジュールによるホスト情報の取得

● setupモジュールの実行

=> setupモジュールを適用すると、操作対象ホストのインベントリ情報を取得できます

```
(venv)# ansible localhost -i ansible_hosts -m setup
localhost | success >> {
    "ansible_facts": {
        "ansible_all_ipv4_addresses": [
            "10.0.0.1"
        ],
        "ansible_all_ipv6_addresses": [
            "fe80::f816:3eff:feb5:3a68"
        ],
        "ansible_architecture": "x86_64",
        "ansible_bios_date": "01/01/2011",
        "ansible_bios_version": "Bochs",
... 中略 ...
```

# Ansibleの仕組みと基本操作

## ■ インベントリファイルとvarsパラメータ(1)

● SNSサーバ(web01,app01,dbs01)の情報からインベントリファイルを作成する
=> step-serverから到達できるdmz-netのIPアドレスを、インベントリファイルに追記します

```
+----------+-----------------+-------------+-------------+-------------+
| hostname | floating_ip     | dmz-net     | app-net     | dbs-net     |
+==========+=================+=============+=============+=============+
| web01    | 192.168.253.126 | 192.168.0.1 | 172.16.10.1 |             |
| app01    |                 | 192.168.0.3 | 172.16.10.3 | 172.16.20.1 |
| dbs01    |                 | 192.168.0.6 |             | 172.16.20.5 |
+----------+-----------------+-------------+-------------+-------------+

(venv)# vi $HOME/ansible_hosts
--- ファイル末尾に追記 ここから ---
[sns]
192.168.0.1
192.168.0.3
192.168.0.6
[web]
192.168.0.1
[app]
192.168.0.3
[dbs]
192.168.0.6
---ファイル末尾に追記 ここまで ---
```

# Ansibleの仕組みと基本操作

## ■ インベントリファイルとvarsパラメータ(2)

● SNSグループにpingモジュールを適用する

=> SNSグループに対して、pingモジュールをansible.cfgのfork数を上限に同時適用します

```
(venv)# ansible sns -i ansible_hosts -m ping -u root
192.168.0.1 | success >> {
    "changed": false,
    "ping": "pong"
}

192.168.0.6 | success >> {
    "changed": false,
    "ping": "pong"
}

192.168.0.3 | success >> {
    "changed": false,
    "ping": "pong"
}
```

# Ansibleの仕組みと基本操作

## ■ インベントリファイルとvarsパラメータ(3)

● SNSグループにパラメータを設定する
=> インベントリファイル(ansible_hosts)に[sns:vars]セクションを追加する

```
(venv)# vi $HOME/ansible_hosts
--- ファイル末尾に追記 ここから ---
[sns:vars]
hello = ”Hello, World!”
--- ファイル末尾に追記 ここまで ---
```

=> helloパラメータの値を確認する

```
(venv)# ansible sns -i ansible_hosts -m debug -a ”replace hello  to {{ hello }}” -u root
192.168.0.1 | success >> {
    ”msg”: ”Hello world!”
}

192.168.0.3 | success >> {
    ”msg”: ”Hello world!”
}

192.168.0.6 | success >> {
    ”msg”: ”Hello world!”
}
```

# Ansibleの仕組みと基本操作

## ■ アドホックなコマンド実行

● SNSグループの各ホストで、OSのコマンドを実行する

=> hostnameコマンドを実行

```
(venv)# ansible sns -i ansible_hosts -m command -a "/bin/hostname" -u root
192.168.0.3 | success | rc=0 >>
app01

192.168.0.1 | success | rc=0 >>
web01

192.168.0.6 | success | rc=0 >>
dbs01
```

=> unameコマンドを実行

```
(venv)# ansible sns -i ansible_hosts -m command -a "/bin/uname -a" -u root
192.168.0.1 | success | rc=0 >>
Linux web01 2.6.32-431.el6.x86_64 #1 SMP Fri Nov 22 03:15:09 UTC 2013 x86_64 x86_64 x86_64 GNU/Linux

192.168.0.6 | success | rc=0 >>
Linux dbs01 2.6.32-431.el6.x86_64 #1 SMP Fri Nov 22 03:15:09 UTC 2013 x86_64 x86_64 x86_64 GNU/Linux

192.168.0.3 | success | rc=0 >>
Linux app01 2.6.32-431.el6.x86_64 #1 SMP Fri Nov 22 03:15:09 UTC 2013 x86_64 x86_64 x86_64 GNU/Linux
```

# Ansibleの仕組みと基本操作

## ■ ホストのフィルタ

● SNSグループの中からappサーバを除外してモジュールを適用する

=> SNSグループのappサーバ以外にpingモジュールを適用する

```
(venv)# ansible 'sns:!192.168.0.3' -i ansible_hosts -m ping -u root
192.168.0.6 | success >> {
    "changed": false,
    "ping": "pong"
}

192.168.0.1 | success >> {
    "changed": false,
    "ping": "pong"
}
```

# Ansibleの仕組みと基本操作

## ■ Ansibleが生成する実行プログラム

● Ansibleがモジュールから生成する実行プログラムを確認する

環境変数ANSIBLE_KEEP_REMOTE_FILESにTrueにセットされていると、Ansibleは生成した実行プログラムを削除せずに終了する。

```
(venv)# export ANSIBLE_KEEP_REMOTE_FILES=True
(venv)# ansible localhost -i ansible_hosts -m ping
```

=> 生成されたPythonプログラム(ping)を確認する

```
(venv)# less /root/.ansible/tmp/ansible-tmp-1418223758.11-110307018321210/ping
```

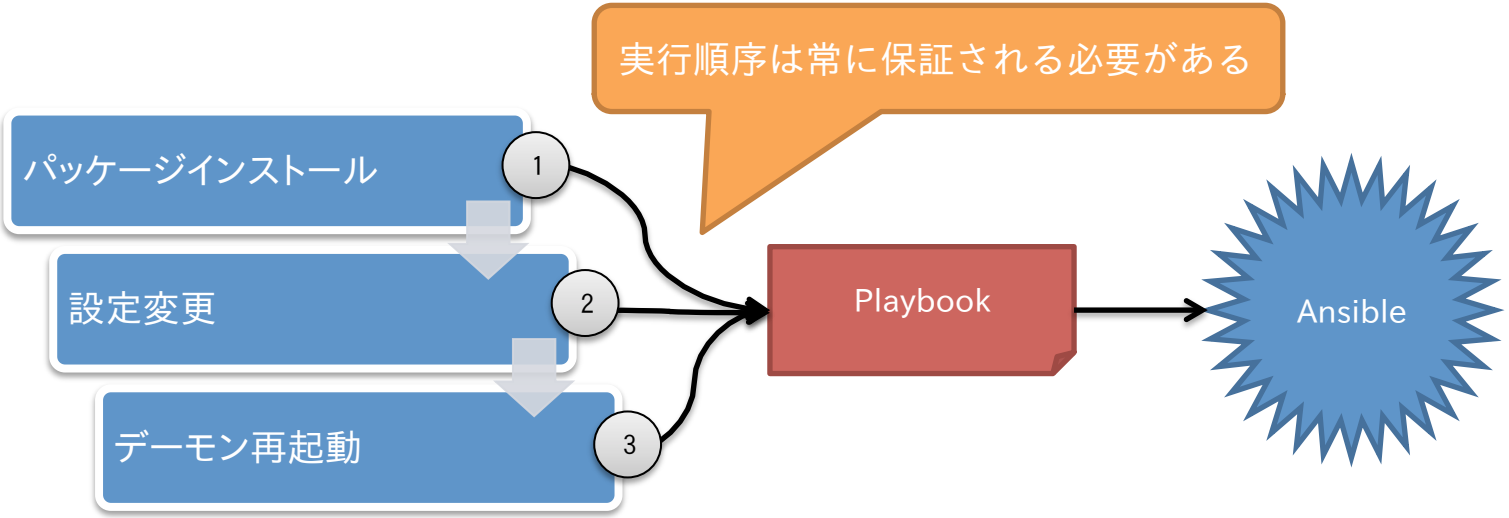
=> Pythonプログラム(ping)を実行する

```
(venv)# python /root/.ansible/tmp/ansible-tmp-1418223758.11-110307018321210/ping
{"changed": false, "ping": "pong"}
```

## 仕事をまとめるPlaybook

ここまでは、ansibleコマンドにより単発の仕事をAnsibleに実行させてきましたが、現実ではこのようが仕事が複数集まって手順として実施されることになります。Ansibleは、下記のような手順をPlaybookとして定義して実行することが可能です。

以降ではAnsibleのPlaybook機能を利用して、手順書にしたがった作業を自動化してみましょう。



# Ansibleの仕組みと基本操作

## ■ Playbookの作成と実行(1)

● githubからサンプルPlaybookを取得する

=> ここでは、あらかじめ用意してあるハンズオン用をサンプルPlaybookを利用します。

```
(venv)# cd $HOME && git clone https://github.com/saito-hideki/handson.git
(venv)# cd handson && git checkout develop
(venv)# cd $HOME

--- 構成 ---
handson ---+--- README.md（※特になにも書いてありません）
           |
           +--- playbooks/ サンプルPlaybook
           |
           +--- scripts/   環境変数設定用スクリプト
```

# Ansibleの仕組みと基本操作

## ■ Playbookの作成と実行(2)

● サンプルPlaybook(hello.yml)を実行する

=> Playbook(handson/playbooks/simple/hello.yml)はYAML形式で記述されています。

```
--- Playbook ここから ---
01: ---
02: - hosts: sns
03:   vars:
04:     hello: "Hello, World!"
05:     hello_file: "sample.txt"
06:   tasks:
07:   - name: create sample file
08:     template:
09:       src: "{{ hello_file }}.j2
10:       dest: /tmp/sample.txt
11:   - name: show {{ hello_file }}
12:     command: "cat /tmp/{{ hello_file }}"
--- Playbook ここまで ---
```

● テンプレートファイルを確認する

=> テンプレートファイル(handson/playbooks/simple/hello.txt.j2)の内容は、varsセクションのパラメータで置換することができます

```
--- テンプレートファイル ここから ---
replace message: {{ hello }}
--- テンプレートファイル ここまで ---
```

# Ansibleの仕組みと基本操作

■ Playbookの作成と実行(2)

=> Playbook(sample.yml)を実行する

```
(venv)# ansible-playbook -i ansible_hosts handson/playbooks/simple/hello.yml -v

PLAY [sns] ********************************************************************
GATHERING FACTS ***************************************************************
ok: [192.168.0.1] ok: [192.168.0.3] ok: [192.168.0.6]
TASK: [create sample file] ****************************************************
ok: [192.168.0.1] => {"changed": false, "gid": 0, "group": "root", "mode": "0644", "owner": "root", "path": "/tmp/sample.txt", "size": 31,
"state": "file", "uid": 0}
ok: [192.168.0.6] => {"changed": false, "gid": 0, "group": "root", "mode": "0644", "owner": "root", "path": "/tmp/sample.txt", "size": 31,
"state": "file", "uid": 0}
ok: [192.168.0.3] => {"changed": false, "gid": 0, "group": "root", "mode": "0644", "owner": "root", "path": "/tmp/sample.txt", "size": 31,
"state": "file", "uid": 0}
TASK: [show {{ hello_file }}] *************************************************
changed: [192.168.0.1] => {"changed": true, "cmd": ["cat", "/tmp/sample.txt"], "delta": "0:00:00.002054", "end": "2014-12-11
00:18:24.853074", "rc": 0, "start": "2014-12-11 00:18:24.851020", "stderr": "", "stdout": "replace message: Hello, World!", "warnings":
[]}
changed: [192.168.0.3] => {"changed": true, "cmd": ["cat", "/tmp/sample.txt"], "delta": "0:00:00.002276", "end": "2014-12-11
00:19:17.272249", "rc": 0, "start": "2014-12-11 00:19:17.269973", "stderr": "", "stdout": "replace message: Hello, World!", "warnings":
[]}
changed: [192.168.0.6] => {"changed": true, "cmd": ["cat", "/tmp/sample.txt"], "delta": "0:00:00.002077", "end": "2014-12-11
00:18:30.737127", "rc": 0, "start": "2014-12-11 00:18:30.735050", "stderr": "", "stdout": "replace message: Hello, World!", "warnings":
[]}
PLAY RECAP ********************************************************************
192.168.0.1                : ok=3    changed=1    unreachable=0    failed=0
192.168.0.3                : ok=3    changed=1    unreachable=0    failed=0
192.168.0.6                : ok=3    changed=1    unreachable=0    failed=0
```

# Ansibleの仕組みと基本操作

## ■ Playbookの作成と実行(3)

● 制御構造をもったPlaybook

=> Playbook(handson/playbooks/simple/pkginstall.yml)では制御構造を採用しています

```
--- Playbook ここから ---
01: ---
02: - hosts: ”{{ target }}”
03:   tasks:
04:   - name: install or upgrade packages
05:     yum:
06:       state: latest
07:       name: ”{{ item }}”
08:     with_items:
09:     - bash
10:     - tcsh
11:     - zsh
12:     when: ansible_distribution == ’CentOS’
--- Playbook ここまで ---
```

# Ansibleの仕組みと基本操作

■ Playbookの作成と実行(4)

=> 制御構造をもったPlaybookを実行する(1回目)

```
(venv)# ansible-playbook -i ansible_hosts -e "target=sns" handson/playbooks/simple/pkginstall.yml

PLAY [sns] *********************************************************************

GATHERING FACTS ****************************************************************
ok: [192.168.0.1]
ok: [192.168.0.3]
ok: [192.168.0.6]

TASK: [install or upgrade packages] ********************************************
changed: [192.168.0.6] => (item=bash,tcsh,zsh)
changed: [192.168.0.1] => (item=bash,tcsh,zsh)
changed: [192.168.0.3] => (item=bash,tcsh,zsh)

PLAY RECAP *********************************************************************
192.168.0.1                : ok=2    changed=1    unreachable=0    failed=0
192.168.0.3                : ok=2    changed=1    unreachable=0    failed=0
192.168.0.6                : ok=2    changed=1    unreachable=0    failed=0
```

# Ansibleの仕組みと基本操作

## ■ Playbookの作成と実行(5)

=> 制御構造をもったPlaybookを実行する(2回目)
chengedが0となっており、重複した操作が行われない(冪等性)が担保されているのが確認できます

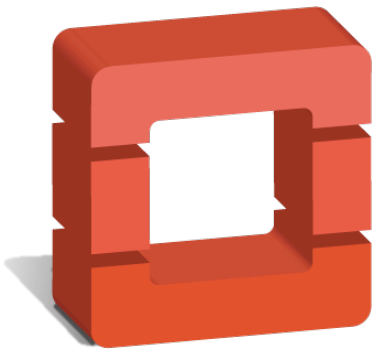
```
(venv)# ansible-playbook -i ansible_hosts -e "target=sns" handson/playbooks/simple/pkginstall.yml

PLAY [sns] *********************************************************************

GATHERING FACTS ****************************************************************
ok: [192.168.0.3]
ok: [192.168.0.6]
ok: [192.168.0.1]

TASK: [install or upgrade packages] ********************************************
ok: [192.168.0.6] => (item=bash,tcsh,zsh)
ok: [192.168.0.1] => (item=bash,tcsh,zsh)
ok: [192.168.0.3] => (item=bash,tcsh,zsh)

PLAY RECAP *********************************************************************
192.168.0.1                : ok=2    changed=0    unreachable=0    failed=0
192.168.0.3                : ok=2    changed=0    unreachable=0    failed=0
192.168.0.6                : ok=2    changed=0    unreachable=0    failed=0
```

# まとめ - ハンズオン前編

ここまでの作業をまとめます

- ✓ Ansibleの仕組みを説明しました
- ✓ step-serverに対してAnsibleのインストールを実施しました
- ✓ ansibleコマンドの基本的な操作を体験しました
- ✓ Playbookの基本構造を説明しました
- ✓ 実際に簡単なPlaybookを作成し、ansible-playbookコマンドから利用しました

# ハンズオン後編



openstack™
CLOUD SOFTWARE

&

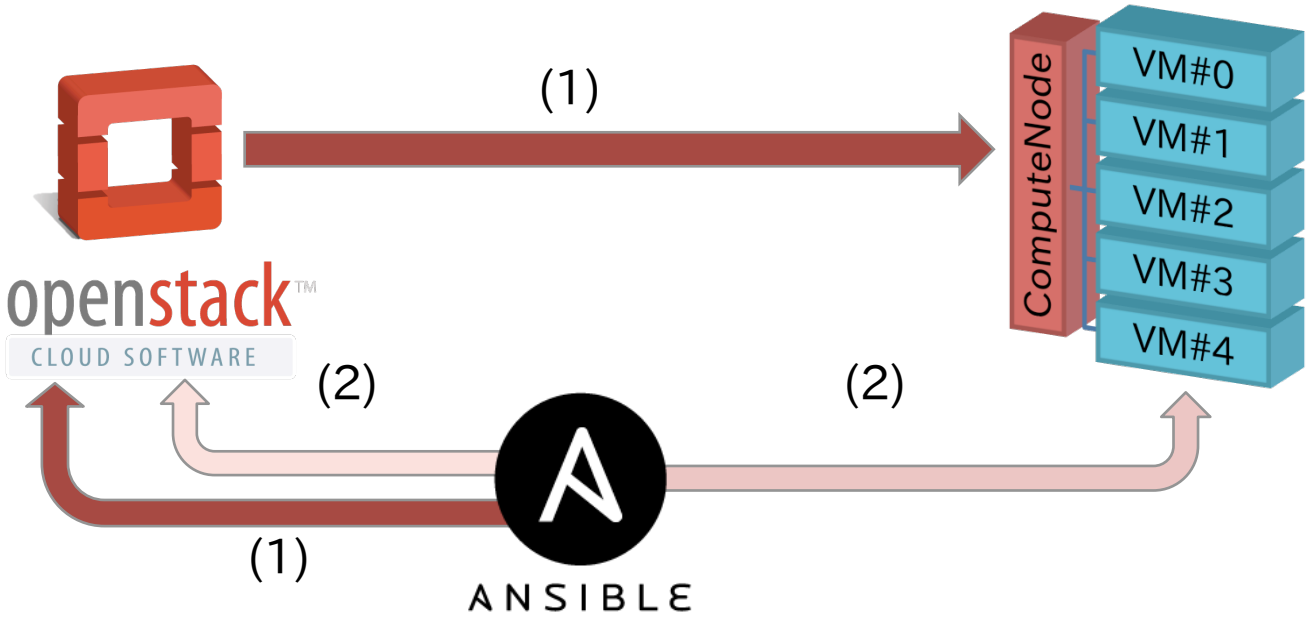
ANSIBLE

# OpenStackとAnsibleの連携

OpenStackは管理下にあるクラウド基盤を外部から制御するためのAPIを提供しています。

AnsibleはこのAPIを利用して、オーケストレーションを実現します。

1) OpenStack API経由でリソースの作成・削除といった管理作業を実施。
2) 仮想マシンのリストやIPアドレス情報をOpenStackのAPIを利用して取得し、仮想マシンに対してリモート・コントロールを実施



# OpenStackとAnsibleの連携

■ 直接制御 – AnsibleのモジュールからOpenStack APIを利用する

| モジュール名 | 概要 |
|---|---|
| glance_image | 仮想マシンイメージの登録・削除 |
| keystone_user | プロジェクト／ユーザ／ロールの作成・削除 |
| nova_compute | 仮想マシンインスタンスの作成・削除 |
| nova_keypair | キーペアの登録・削除 |
| quantum_floating_ip | フローティングIPの新規払い出しと仮想マシンへの割り当て |
| quantum_floating_ip_associate | フローティングIPの割り当てと割り当て解除 |
| quantum_network | 仮想ネットワークの作成・削除 |
| quantum_router | 仮想ルータの作成・削除 |
| quantum_router_gateway | 仮想ルータと外部ネットワークセグメントの接続・切断 |
| quantum_router_interface | 仮想ルータと仮想ネットワークセグメントの接続・切断 |
| quantum_subnet | 仮想ネットワーク内のサブネットの作成・削除 |

■ 間接利用 – 構成管理データベースとして間接利用する
仮想マシンの情報を管理する外部インベントリとして利用

# OpenStackとAnsibleの連携

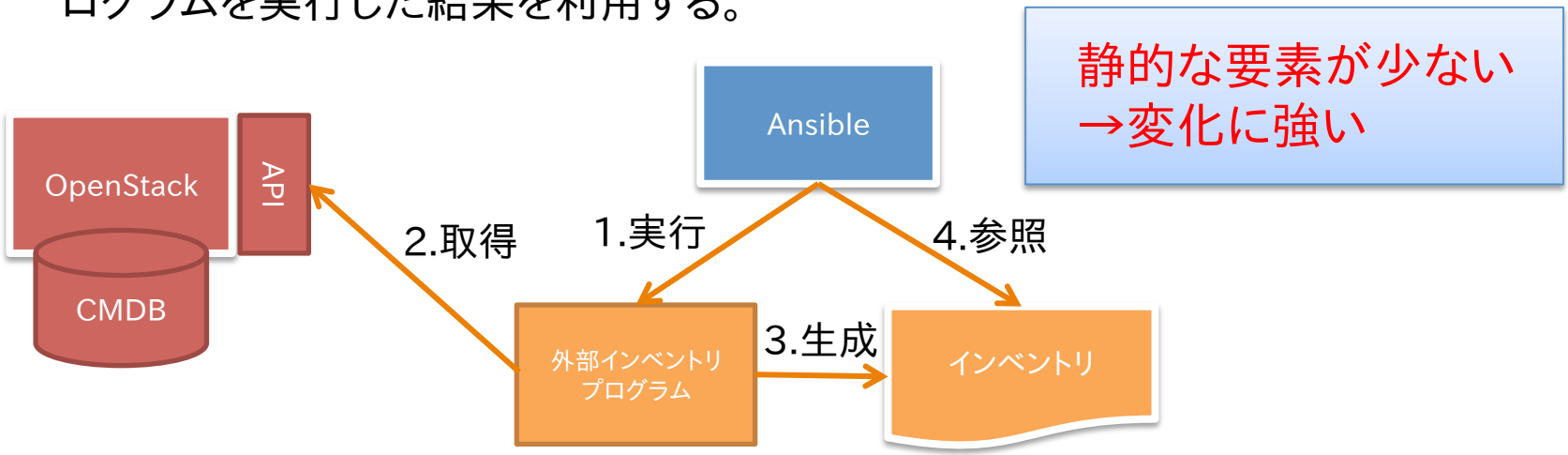
Ansibleは管理対象ノードのインベントリ情報を、a)またはb)の方法で取得します。このインベントリ情報から操作対象となるホストを特定して操作します。

a) 静的なインベントリ

UNIX系OSの/etc/hostsのようにホストのリストやパラメータが登録されたプレーンテキストファイルを利用する。

b) 動的なインベントリ(外部インベントリ)

JSON形式で構造化されたホストのリストとパラメータが標準出力される外部プログラムを実行した結果を利用する。



# OpenStackとAnsibleの連携

■ OpenStackクライアントライブラリの利用(1)

● openrcを利用する
=> OpenStack APIの接続情報を設定します
(venv)# cd $HOME
(venv)# source openrc
=> bpythonを起動します
(venv)# bpython
>>>

# OpenStackとAnsibleの連携

■ OpenStackクライアントライブラリの利用(2)

=> OpenStackが提供するクライアントライブラリを利用して、仮想マシン名を取得します

```
--- サンプルコード ここから ---
import os
user = os.environ.get('OS_USERNAME')
passwd = os.environ.get('OS_PASSWORD')
url = os.environ.get('OS_AUTH_URL')
tenant = os.environ.get('OS_TENANT_NAME')
region = os.environ.get('OS_REGION_NAME')
from novaclient.client import Client
nv = Client(1.1, user, passwd, tenant, url,region_name=region)
for sv in nv.servers.list():
    print sv.name

exit()
--- サンプルコード ここまで ---

--- 出力 ---
u'dbs01'
u'app01'
u'web01'
u'step-server'
```

# OpenStackとAnsibleの連携

■ OpenStackクライアントライブラリの利用(3)

=> OpenStackが提供するクライアントライブラリを利用して、仮想ネットワーク名を取得します

--- サンプルコード ここから ---

```
import os
user = os.environ.get('OS_USERNAME')
passwd = os.environ.get('OS_PASSWORD')
url = os.environ.get('OS_AUTH_URL')
tenant = os.environ.get('OS_TENANT_NAME')
region = os.environ.get('OS_REGION_NAME')
from keystoneclient.v2_0 import client as ksclient
from neutronclient.v2_0 import client
kc = ksclient.Client(
username=user,password=passwd,tenant_name=tenant,region_name=region,auth_url=url)
tk = kc.auth_token
ep = kc.service_catalog.url_for(service_type='network', endpoint_type='publicURL')
nc = client.Client(token=tk, endpoint_url=ep)
print [network['name'] for network in nc.list_networks()['networks']]
exit()
```

--- サンプルコード ここから ---

--- 出力 ---

```
[u'work-net', u'dmz-net', u'app-net', u'dbs-net', u'ext-net']
```

# OpenStackとAnsibleの連携

■ Playbookを利用して仮想マシンを起動する

● 仮想マシンを起動するためのサンプルPlaybookを利用してweb,app,dbsサーバを起動させます
=> 起動前に、Playbookが内部で利用している環境変数にネットワークIDをセットします

```
(venv)# cd $HOME
(venv)# source handson/scripts/get_network.sh
OS_DBS_NET=afba2415-f58b-455c-9c99-dfb5a80864d0
OS_APP_NET=ad993d9a-c369-4a33-8b0e-6972b88ea3f3
OS_DMZ_NET=443debfc-1e22-4e8b-8764-6fbf8ef1d4c9
```

# OpenStackとAnsibleの連携

■ 仮想マシンを起動する(web)

```
=> webサーバを起動します
(venv)# ansible-playbook -i ansible_hosts -e target=web handson/playbooks/openstack/create_vm.yml

PLAY [localhost] ***************************************************************

GATHERING FACTS ****************************************************************
ok: [localhost]

TASK: [ansible_python_interpreter setup] ***************************************
ok: [localhost]

TASK: [get uuid for generate hostname] *****************************************
changed: [localhost]

TASK: [create {{ target }}-server on nova-compute with floating_ip] ************
changed: [localhost]

TASK: [create {{ target }}-server on nova-compute without floating_ip] ********
skipping: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=4    changed=2    unreachable=0    failed=0
```

# OpenStackとAnsibleの連携

■ 仮想マシンを起動する(app)

```
=> appサーバを起動します
(venv)# ansible-playbook -i ansible_hosts -e target=app handson/playbooks/openstack/create_vm.yml

PLAY [localhost] ***************************************************************

GATHERING FACTS ****************************************************************
ok: [localhost]

TASK: [ansible_python_interpreter setup] ***************************************
ok: [localhost]

TASK: [get uuid for generate hostname] *****************************************
changed: [localhost]

TASK: [create {{ target }}-server on nova-compute with floating_ip] ************
skipping: [localhost]

TASK: [create {{ target }}-server on nova-compute without floating_ip] *********
changed: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=4    changed=2    unreachable=0    failed=0
```

# OpenStackとAnsibleの連携

■ 仮想マシンを起動する(dbs)

```
=> dbsサーバを起動します
(venv)# ansible-playbook -i ansible_hosts -e target=dbs handson/playbooks/openstack/create_vm.yml

PLAY [localhost] ***************************************************************

GATHERING FACTS ****************************************************************
ok: [localhost]

TASK: [ansible_python_interpreter setup] ***************************************
ok: [localhost]

TASK: [get uuid for generate hostname] *****************************************
changed: [localhost]

TASK: [create {{ target }}-server on nova-compute with floating_ip] ************
skipping: [localhost]

TASK: [create {{ target }}-server on nova-compute without floating_ip] *********
changed: [localhost]

PLAY RECAP *********************************************************************
localhost                  : ok=4    changed=2    unreachable=0    failed=0
```

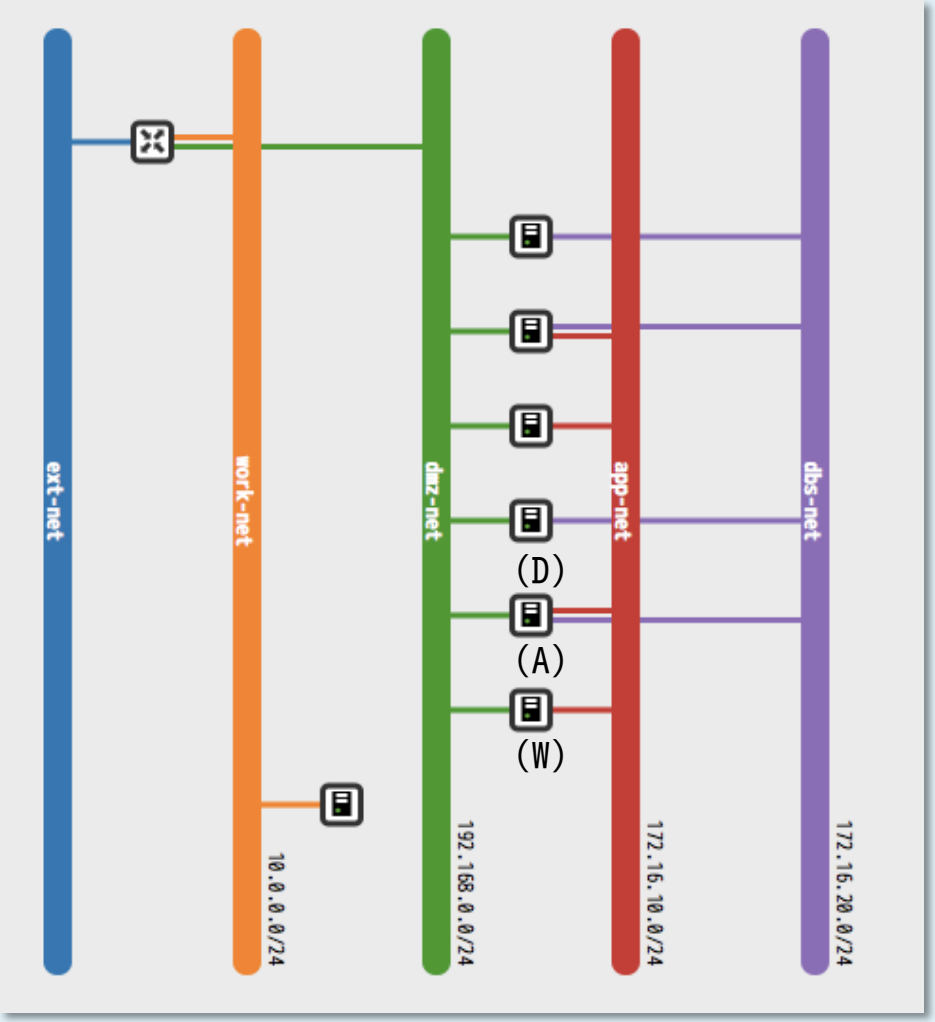
# OpenStackとAnsibleの連携

■ 仮想マシンを起動確認

=> 仮想マシンが起動したことを確認します

```
# nova list --fields name,networks
+--------------------------------------+--------------------------------------+-----------------------------------------------------------------------+
| ID                                   | Name                                 | Networks                                                              |
+--------------------------------------+--------------------------------------+-----------------------------------------------------------------------+
| 66f50ab1-89ad-4310-8559-3a67e0808806 | app-d477ec8b-7f11-4669-b6f0-f085b15849a5 | dmz-net=192.168.0.11; app-net=172.16.10.7; dbs-net=172.16.20.8 |
| c01db87c-19a7-41d1-8791-3fcbb7c768ac | app01                                | dmz-net=192.168.0.3; app-net=172.16.10.3; dbs-net=172.16.20.1  |
| a6d4a329-087a-4027-93d9-69ed1f37806c | dbs-94eb3c30-a545-470e-aaf2-ea09012546e9 | dmz-net=192.168.0.12; dbs-net=172.16.20.9                      |
| b28e161e-e9b5-469a-9909-471e055eed9a | dbs01                                | dmz-net=192.168.0.6; dbs-net=172.16.20.5                       |
| 5c47687f-b461-4556-be5f-e2ff01b627e0 | step-server                          | work-net=10.0.0.1, 192.168.253.120                             |
| a96825ab-73c4-4d99-85ae-d111f2680c4e | web-723650a2-e9f3-46de-9e73-295790276e11 | dmz-net=192.168.0.10, 192.168.253.130; app-net=172.16.10.6     |
| 110ad3c0-a355-470e-b2fa-e18e9a26849a | web01                                | dmz-net=192.168.0.1, 192.168.253.126; app-net=172.16.10.1      |
+--------------------------------------+--------------------------------------+-----------------------------------------------------------------------+
```

# 新たに追加された3台のSNSサーバ




AnsibleのPlaybookを利用して、
3台の仮想マシンを起動しました

(W) web-<UUID>
(A) app-<UUID>
(D) dbs-<UUID>

# OpenStackとAnsibleの連携

## ■ 外部インベントリの利用

Ansibleは、操作対象ホストをインベントリーファイルで特定します。この仕組みの問題点は、管理対象ホストが増減して変化があった場合に、それに合わせて、インベントリファイルの修正が必要な点です。これは、利用者の増減に応じて仮想マシンインスタンスを増減するような、OpenStackに代表されるクラウド基盤上のシステムを管理する場合は、特に厄介な問題になります。

この問題への対策として、Ansibleでは、静的なインベントリーファイルの代わりに、外部プログラム(外部インベントリ)を実行して取得したインベントリー情報を利用する、「ダイナミックインベントリー」の機能が提供されています。ダイナミックインベントリーとして動作する、外部プログラムが満たすべき要件は次の通りです。

● ダイナミックインベントリ

=> 外部プログラムとして実行するプログラムは、単体実行可能で、--listと--hostオプションを持ちます

=> sns_inventory.ini にOpenStackの接続情報を設定してください

```
(venv)# cd $HOME/handson/playbooks/openstack
(venv)# chmod +x sns_inventory.py
(venv)# ./sns_inventory.py --list
... 省略 ...
(venv)# ./sns_inventory.py --host 192.168.0.7
{ "app": "172.16.10.5" }
```

# OpenStackとAnsibleの連携

■ SNSアプリケーションのデプロイ(dbs)

● dbs->app->webの順にアプリケーションをデプロイします

```
(venv)# ansible-playbook -i sns_inventory.py -e target=dbs install_sns.yml
PLAY [dbs] ********************************************************************

GATHERING FACTS ***************************************************************
ok: [192.168.0.12]

TASK: [change the timezone] ***************************************************
changed: [192.168.0.12]

TASK: [checkout app from github] **********************************************
changed: [192.168.0.12]

TASK: [execute install script] ************************************************
ok: [192.168.0.12]

TASK: [copy endpoint.conf to web and app server] ******************************
skipping: [192.168.0.12]

PLAY RECAP ********************************************************************
192.168.0.12               : ok=4    changed=2    unreachable=0    failed=0
```

# OpenStackとAnsibleの連携

■ SNSアプリケーションのデプロイ(2)

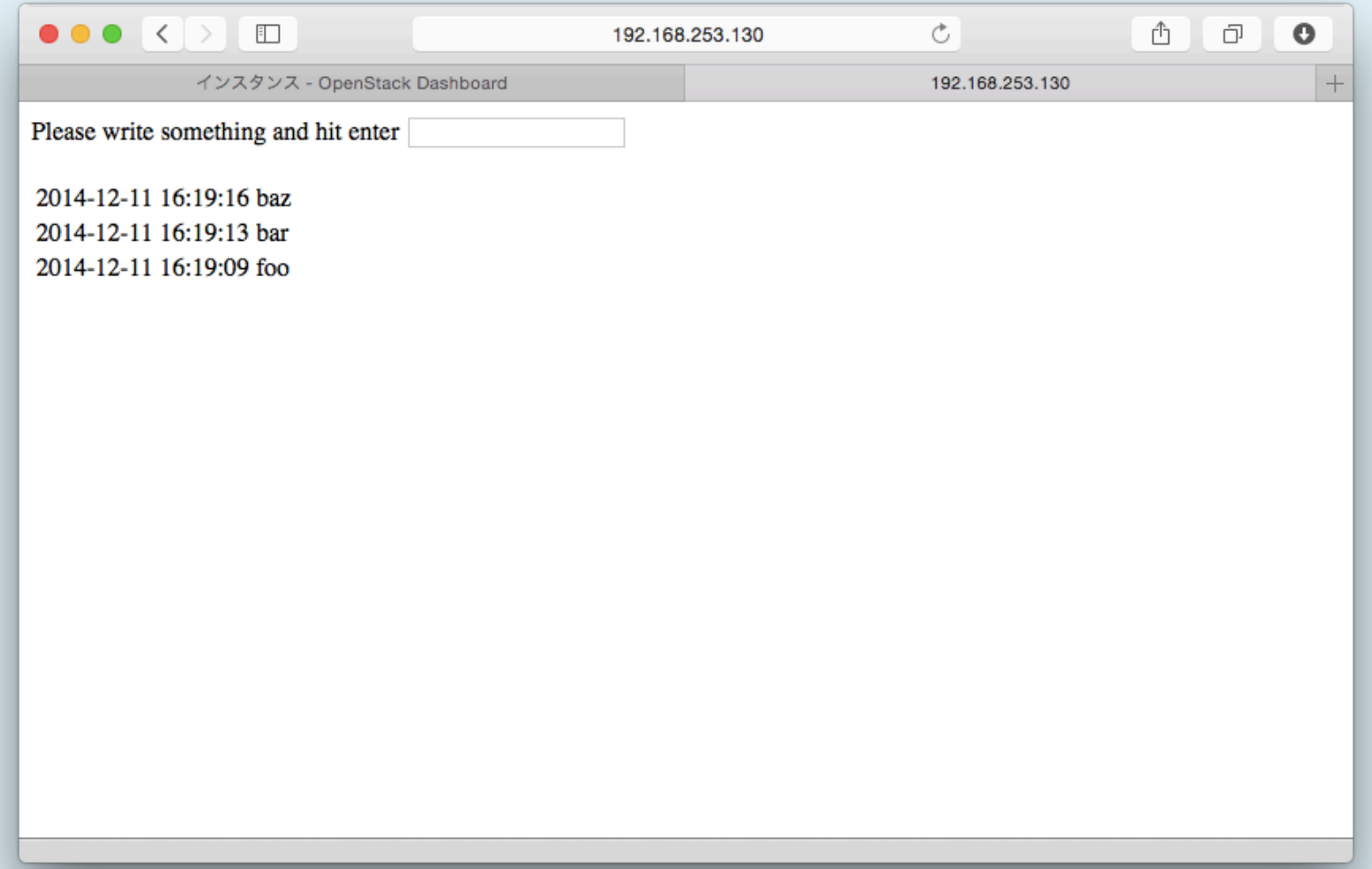
```
(venv)# ansible-playbook -i sns_inventory.py -e target=app install_sns.yml
PLAY [app] *********************************************************************

GATHERING FACTS ****************************************************************
ok: [192.168.0.11]

TASK: [change the timezone] ****************************************************
changed: [192.168.0.11]

TASK: [checkout app from github] ***********************************************
changed: [192.168.0.11]

TASK: [execute install script] *************************************************
ok: [192.168.0.11]

TASK: [copy endpoint.conf to web and app server] *******************************
changed: [192.168.0.11]

NOTIFIED: [startup service] ****************************************************
changed: [192.168.0.11]

PLAY RECAP *********************************************************************
192.168.0.11               : ok=6    changed=4    unreachable=0    failed=0
```

# OpenStackとAnsibleの連携

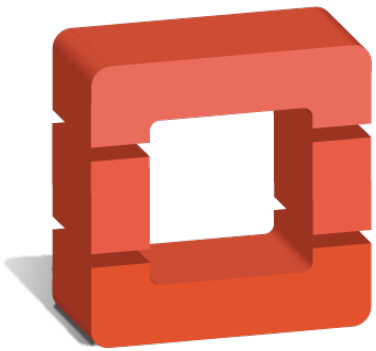
■ SNSアプリケーションのデプロイ(3)

```
(venv)# ansible-playbook -i sns_inventory.py -e target=web install_sns.yml
PLAY [web] *********************************************************************

GATHERING FACTS ****************************************************************
ok: [192.168.0.10]

TASK: [change the timezone] ****************************************************
changed: [192.168.0.10]

TASK: [checkout app from github] ***********************************************
changed: [192.168.0.10]

TASK: [execute install script] *************************************************
ok: [192.168.0.10]

TASK: [copy endpoint.conf to web and app server] *******************************
changed: [192.168.0.10]

NOTIFIED: [startup service] ****************************************************
changed: [192.168.0.10]

PLAY RECAP *********************************************************************
192.168.0.10               : ok=6    changed=4    unreachable=0    failed=0
```

# 起動したSNSアプリケーション



# まとめ



openstack™
CLOUD SOFTWARE

&

ANSIBLE

# ハンズオン作業のまとめ

ここまでの作業をまとめます

- ✓ OpenStackとAnsibleの組み合わせる方法を紹介しました
- ✓ Ansibleを利用して、OpenStackのクラウド基盤上に仮想マシンを作成しました
- ✓ 外部インベントリを利用してOpenStackから動的にインベントリ情報を取得しました
- ✓ 外部インベントリを利用してSNSアプリケーションをデプロイしました

# 組み合わせによる相乗効果

AnsibleとOpenStackをリファレンスモデルとした、ツールの組み合わせによる相乗効果をあげるための5つの重要な特徴

■ クラウド基盤

**1) 最新情報の管理と提供**

構成管理データベースによるシステム情報の管理

**2) コントローラブル**

オーケストレータから制御可能なAPIを持つ

■ オーケストレータ

**3) 情報は源泉から**

管理対象となる情報はクラウド基盤から引く

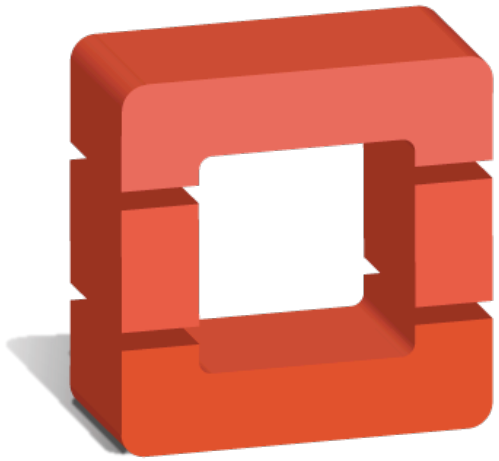
**4) 一貫性を保つ**

自身では管理するのは必要最低限の情報のみ

**5) 相手を選ばない**

サーバだけでなく、ネットワーク機器やストレージも管理対象となる場合は、エージェントレス型が理想。

# 自動化編 ～おつかれさまでした～



openstack™

CLOUD SOFTWARE