Ansible 入門

目次

1 構成管理ツールと ANSIBLE について

近年 Web アプリが増え、サーバの環境構築を行ったり、アプリのデプロイを行ったりする機会が増えてきました。環境構築やデプロイを何度も行う場合は、シェルスクリプトを使って自動化している事が多いかと思います。

しかし、シェルスクリプトで処理を自動化する際、以下のような問題が起こることがあります。

- 環境構築に使う設定ファイルに誤りがあるなどでスクリプトが途中で止まってしまった場合、ファイルを修正してもう一度スクリプトを実行すると、DB の値や他の設定ファイルの値が不正になってしまう。そのため、再実行の前に手動で変更前の値に戻さなければならない
- 上記のような場合に対応するためのスクリプトにエラー処理が多く記述され、可読性が低くなる

このような問題を解決するには、構成管理ツールを使うと便利です。構成管理ツールは、処理を自動化するという観点ではなく、最終的にマシンを目的の状態にする、という目的で作成されたツールのことです。

例えば、ユーザを追加するという目的を考えてみます。シェルスクリプトではそのままユーザを追加するという操作を書くため、すでにユーザが存在する場合、何も考慮しないと再実行時にすでにユーザが存在するというエラーとなってしまいます。

一方、構成管理ツールでは、ユーザが追加されている状態にするという命令を使います。すでに目的の状態となっているかどうか(=ユーザが存在するかどうか)という判断は構成管理ツールが行ってくれるため、自前で判定処理を書く必要がありません。

上記の構成管理ツールの例のように、同じ操作を何度やっても最終的な状態が同じになる、という性質をべき等性といいます。環境構築やデプロイでは、べき等性を意識して処理を自動化すべきであり、構成管理ツールはその助けとなります。

今回はその構成管理ツールの一つである Ansible(http://www.ansible.com/)を紹介します。Ansible は構成管理ツールの中でもシンプルさを特徴としており、はじめて構成管理を行ってみようという方に適しています。

Ansible ではべき等性を意識した処理(モジュール)が多く用意されており、この処理を組み合わせてプレイブックと呼ばれる手順書を作ることで、プログラマでない人でも簡単に構成管理が行えるようになっています。

また、Ansible で構成管理するマシンには、特別なクライアントソフトのインストールが不要です。そのため、業務の制約上、本番環境で任意のパッケージのインストールができないような場合でも、Ansible の利用が可能です。

この文書では、今までマニュアルやシェルスクリプトを使ってサーバ管理を行っていたけれども、もう少しうまいやり方をしてみたい、という方を対象に Ansible の基本的な使い方を紹介していきます。

2 環境構築

Ansible では、構成管理を行う指示を出すマシン(管理ホスト)から、構成管理されるマシン(被管理ホスト)に SSH で命令を飛ばすことで、構成管理を行います。今回は管理ホストとして CentOS 6.5 を使う場合の環境構築方法を紹介します。また、執筆時点の Ansible の最新バージョンである 1.7 を利用します。その他の OS を利用の方は、以下のインストールガイドを参照してください。

http://docs.ansible.com/intro_installation.html#installing-the-control-machine

2.1 ANSIBLE に必要なパッケージ

Ansible を実行するために、管理ホストには Python 2.6 がインストールされている必要があります。CentOS 6.5 ではデフォルトでインストールされていますので問題ありません。もし別 OS を利用していて Python2.6 以上がインストールされていない場合は、各 OS のパッケージマネージャを使ってインストールしておいてください。

被管理ホストでは、Python2.5 以上がインストールされている必要があります。ただし、Python2.4 でも、python-simplejson というパッケージをインストールすれば動作します。CentOS であれば、以下のコマンドで python-simplejson をインストールできます。

```
yum install python-simplejson
```

なお、Ansible は python3 に対応していません。もし被管理ホストに python3 のパスが通っている場合の対応については、Tips の章で示します。

2.2 ANSIBLE のインストール

必要なパッケージがインストール済みであることを確認したら、さっそく Ansible をインストールしてみます。

Ansible は、yum の拡張リポジトリである EPEL(Extra Packages for Enterprise Linux)からインストールすることが可能です。EPEL リポジトリが設定されていない場合は、以下のコマンドを利用して yum コマンドで EPEL リポジトリを利用できるようにします。

```
yum install epel-release
```

EPEL リポジトリが設定できたならば、以下のコマンドを実行すれば、Ansible をインストールできます。

```
yum install ansible
```

インストールに成功すると、以下のファイルが作成されます。

- /usr/bin/ansible
- /usr/bin/ansible-playbook

3 ANSIBLE の構成と動作確認

3.1 ANSIBLE の構成

さっそく Ansible を動作させてみたいところですが、その前に Ansible の構成を確認してみます。Ansible を構成する主な要素として、以下の三つがあります。

- インベントリ(inventory)

被管理ホストの情報を記述するファイル

- モジュール(module)

  ユーザを追加する、ファイルをコピーする、などのべき等性を持った操作。200 以上のモジュールがあらかじめが用意されている

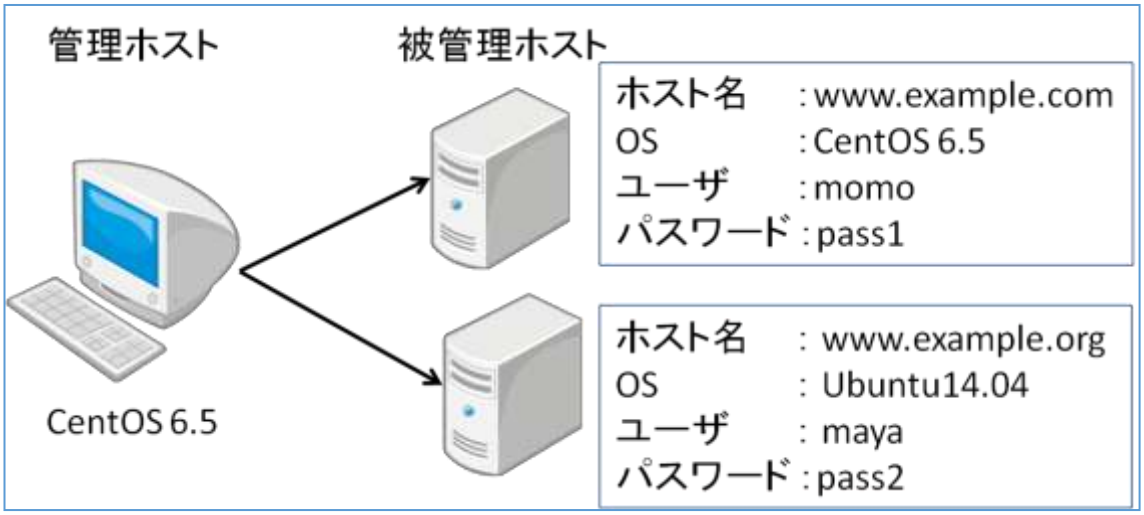
- プレイブック(playbook)

  DB サーバを作る、デプロイを行う、などの目的のためにモジュールを組み合わせた作った手順書

構成管理を行うにあたり、管理者が作成する必要があるファイルはインベントリとプレイブックの 2 つです。モジュールについても自作することが可能ですが、基本的な処理であれば Ansible が用意しているモジュールをそのまま使えば問題ありません。

インベントリファイルだけでも Ansible を実行することができるため、まずはインベントリファイルを作成してみます。

今回は以下の環境を想定してインベントリファイルを記述します。また、被管理ホストのユーザは sudo 権限があるものとします。



3.2 インベントリ

インベントリファイルは INI ファイル形式で被管理ホストの情報、接続方法などのパラメータを記述します。インベントリファイルはデフォルトでは/etc/ansible/hosts が参照されるようになっているので、作成したファイルはこのパスに置いておきましょう。

Ansible のインストール直後は/etc/ansible/hosts にはサンプルが置かれているため、既存のファイルはバックアップを取っておくとよいでしょう。

インベントリファイルに最低限必要な情報はホスト名のみで、その他に、接続ユーザやパスワードなどのオプションを付け加えていきます。

今回の例だと、以下のように記述することになります。

```
www.example.com      ansbile_ssh_user=momo ansible_ssh_pass=pass1
www.example.org      ansible_ssh_user=maya  ansible_ssh_pass=pass2
```

行頭に接続先のホスト名を書き、オプションとして ansible_ssh_user で被管理ホストにログインするユーザ名を、ansible_ssh_pass でログインに使うパスワードを設定しています。

今回は検証用なので ansible_ssh_pass にパスワードを平文で書いてしまっていますが、セキュリティ上問題がある場合は、Ansible の実行の際にパスワードを入力するようにすることも可能です。この方法は次節で紹介します。

また、上記で設定した以外によく使うパラメータは以下のようなものがあります。

- ansible_ssh_host

ホスト名としてエイリアスを使いたい場合に、実際に接続するホストを指定します。例えば、www.example.com に対し、example というエイリアスを付ける場合は以下のようになります。

```
example ansible_ssh_host=www.example.com
```

- ansible_ssh_port

接続先のポートを指定します。このパラメータを指定しない場合のデフォルト値は 22 になります。

- ansible_ssh_private_key_file

SSH 接続時の認証に使う秘密鍵のファイルパスを指定します。

- ansible_sudo_pass

被管理ホストで sudo でコマンドを実行する際のパスワードを指定します。

3.3 ANSIBLE の動作確認

インベントリファイルを作成し、/etc/ansible/hosts に配置したら、以下のコマンドを実行してください。

```
ansible all -m ping
```

インベントリファイルの記述に問題なければ、以下のように結果が表示されます。

```
www.example.com | success >> {
    "changed": false,
    "ping": "pong"
}

www.example.org | success >> {
    "changed": false,
    "ping": "pong"
}
```

上記で実行した ansible コマンドは、指定したホストに対し、単一のモジュールを実行するコマンドです。基本的な構成は以下の通りとなります。

```
ansible 対象ホスト名 -m モジュール名
```

さきほどの例では、対象ホストとして all を、モジュール名として ping を指定していました。ホスト指定の all は、インベントリファイルに記述したホスト全てを対象とするという意味になります。

Ansible コマンドで指定可能な引数は以下のようなものがあります。

- -i インベントリファイル

  インベントリファイルとして使うファイルを指定します。このオプションが指定されない場合は/etc/ansible/hosts がインベントリファイルとして利用されます。

- --user ユーザ名

被管理ホストでコマンドを実行するユーザ名を指定します。このパラメータを指定しない場合はインベントリファイルで ansible_ssh_user オプションで指定したユーザが利用されます。インベントリファイルにも指定がない場合は、ansible コマンドを実行したユーザが使われます。

- --ask-pass

  SSH 接続に使うパスワードをコンソール入力から受け取るようにします。インベントリファイルでパスワードを記述しなかった場合は、このオプションを使って認証を行います。

4 プレイブック

さきほどの例では、ping という単一のモジュールを実行していましたが、ping を送るだけでは特にべき等性が関係なく、あまり Ansible を使う意味がありません。今度はプレイブックを使って、複数のモジュールを組み合わせて実行してみます。

プレイブックは、YAML(http://www.yaml.org/)という形式で記述するため、まずは簡単に YAML について説明します。

4.1 YAML

YAML とは構造化されたデータを表現するための規格で、シーケンスとマッピングという二つのデータ構造を使います。

シーケンスはいわゆるリスト形式の事で、以下のようにハイフンを使って要素を記述します。ハイフンと要素の間には、半角スペースがないとエラーになるので気を付けてください。

```
-A
-B
-C
```

マッピングはキーと値を持ったデータ構造で、コロンでキーと値を区切って表現します。シーケンスと同様に、コロンと値の間にも、半角スペースが必要です。以下の例では、apple というキーで red という値を持つマッピングを定義しています。

```
 apple: red
```

以下のようにシーケンスの値としてマッピングを持つことも可能です。

```
- apple: red
  banana: yellow
- foo: hoge
  bar: piyo
```

また、シーケンス、マッピングはインデントを使ってネストすることが可能です。インデントには半角スペースを使い、タブは利用できません。インデントには半角スペース 2 つを使うことが一般的です。

```
- apple: red
  hoge:
      foo: bar
      piyo: fuga
```

4.2 プレイブックを実行する

さっそくプレイブックを作ってみます。被管理ホストに対し、sora というユーザを追加し、そのホームディレクトリに/etc/ansible/hosts ファイルをコピーする、という 2 つの処理をプレイブックで記述してみます。

main.yml というファイルを作成し、以下の内容を記述してください。マッピングのコロンの後と、リストのハイフンの後には空白スペースが必須なことと、インデントの位置を合わせることに気を付けてください。

```
- hosts: all
  sudo: yes
  vars:
    user_name: sora
    src_file: /etc/ansible/hosts
  tasks:
    - user: name={{ user_name }}
    - copy: src={{ src_file }} dest=/home/{{user_name }}/
```

ファイルが作成できたら、以下のコマンドを実行します。実行すると、sudo に使うパスワードを聞かれるので、インベントリファイルで指定したユーザのパスワードを入力してください。

```
ansible-playbook main.yml --ask-sudo-pass
```

プレイブックが正しく作成されていれば、以下のように出力されます。

```
PLAY [all]*********************************************************

GATHERING FACTS
*************************************************************
ok: [www.example.net]
ok: [www.example.org]


TASK: [user name=sora]
*************************************************************
changed: [www.example.net]
changed: [www.example.org]

TASK: [copy src=/etc/ansible/hosts dest=/home/sora/] *************
changed: [www.example.net]
changed: [www.example.org]

PLAY RECAP
********************************************************************
www.example.net    : ok=3    changed=2    unreachable=0    failed=0
www.example.org    : ok=3    changed=2    unreachable=0    failed=0
```

実行が成功したら、再度同じコマンドを実行してみます。そうすると、さきほどと出力が変わっているところがあります。

```
PLAY [all]*********************************************************
```

```
GATHERING FACTS
*********************************************************************
ok: [www.example.net]
ok: [www.example.org]


TASK: [user name=sora]
*********************************************************************
ok: [www.example.net]
ok: [www.example.org]

TASK: [copy src=/etc/ansible/hosts dest=/home/sora/] *************
ok: [www.example.net]
ok: [www.example.org]

PLAY RECAP
********************************************************************
www.example.net     : ok=3     changed=0     unreachable=0     failed=0
www.example.org     : ok=3     changed=0     unreachable=0     failed=0
```

初回実行時に changed と出ていた箇所が ok に変わっています。これは、実行したモジュールがべき等性を持つため、すでに目的の状態となっている事を判断し、処理を行わなかったことを示しています。

4.3 プレイブックの構成

では、さきほどのプレイブックの内容を上から順に見ていきます。プレイブックは大まかに分けて接続設定を行う箇所と、変数を定義する箇所、実行モジュールを定義する箇所の三つの構成でできています。

4.3.1 接続設定

```
- hosts: all
  sudo: yes
```

1 行目の hosts 要素では、インベントリファイルに記述したホスト名を指定します。複数のホスト名を指定したい場合は、コロンで区切ることで、複数のホストを指定可能です。

hosts の値に all を指定することで、インベントリファイルに記述した全てのホストを対象とすることが可能です。

2 行目では、このプレイブックのモジュールを sudo で実行するように指定しています。(ユーザの追加は管理者が行うことなので、root 権限が必要です。)

もしインベントリファイルで指定したユーザ以外でログインしたい場合は user パラメータを指定して、上書きすることも可能です。

4.3.2 変数定義

```
 vars:
   user_name: sora
   src_file: /etc/ansible/hosts
```

vars 要素では、変数宣言を行っています。今回は user_name という変数に sora という値を、src_file という変数に/etc/ansible/hosts という値を設定しています。

変数名には、アルファベット、数字、アンダースコアのみが利用でき、最初はアルファベットから始まる必要があります。

変数の値を参照するには、{{変数名}}の形式を使います。

また、上記の例では、変数をプレイブックに直接記述していましたが、これでは別のユーザやファイルを追加したい場合に、このプレイブックを毎回書き換えなければならなくなります。そこで、変数を別ファイルから読み込ませることもできます。

変数を読み込ませるには、vars 要素ではなく vars_files 要素を使います。main.yml を以下のように変更してください。

```
- hosts: all
  vars_files:
     var.yml
  tasks:
        …(省略)
```

続いて、上記ファイルで指定している var.yml ファイルを作成し、さきほどプレイブックの var 要素に記述した内容をそのまま記述します。

```
user_name: sora
src_file: /etc/ansible/hosts
```

このように変数定義部分を外出しすることで、プレイブックを使いまわしやすくなります。利用する変数に変更がありうる場合は、外出ししておいた方が便利です。

4.3.3 実行モジュール

```
tasks:
    - user: name={{ user_name }}
    - copy: src={{ src_file }} dest=/home/{{user_name }}/
```

最後の tasks 要素では、実行するモジュールを記述します。

今回はユーザ追加を行う user モジュールと、管理ホストから被管理ホストへのファイルコピーを行う copy というモジュールを利用しました。

モジュールはキーとしてモジュール名を、値としてモジュール毎のオプションを設定します。user モジュールであれば、追加するユーザ名が必須項目としてあるので、これを name というオプションで渡しています。copy モジュールでは、src オプションに管理ホストのコピー元ファイルのパスを、dest オプションに被管理ホストのコピー先パスを記述します。

次の章ではよく使われるモジュールを紹介していきます。

5 モジュール

5.1.1 ユーザを追加する

まっさらなサーバに操作ユーザを追加することはよくあることかと思います。さきほどの例でもみましたが、ユーザの追加は user モジュールを使います。

user モジュールの必須項目はユーザ名だけですが、グループ名やパスワードも同時に設定することができます。

ユーザ sora を wheel グループに追加し、パスワードも設定する例は以下の通りです。

```
user: name=sora groups=wheel password=1$sugar$Q9IYVtrmI0Xu12.Wu1
```

ここで、パスワードの値が平文ではないことが見て取れると思います。user モジュールではユーザのパスワードを平文では指定ができなく、暗号化されている必要があります。

パスワードの暗号化は以下のコマンドで可能です。ソルトの部分には適当な文字列を入れておいてください。

```
openssl passwd -salt ソルト -1 パスワード
```

なお、あまりないケースではあるかと思いますが、ユーザの削除を行うこともできます。

その場合は state=absent というオプションを追加します。

absent という単語は～が存在しないという意味の形容詞です。構成管理ツールでは最終的な状態を示すため、delete のような動詞ではなく、absent という形容詞が使われています。

5.1.2 ファイルの行を追加/削除/置換する

環境構築で設定ファイルを編集する際、任意の行を追加する、もしくは値を書き換えるということがよくあります。その場合は lineinfile モジュールを使います。

/tmp/testfile に hoge という行を追加する例は以下の通りです。dest オプションで編集対象のファイルを、line オプションで追加する内容を記述しています。

```
lineinfile: dest=/tmp/testfile line=hoge
```

行を削除する場合、user モジュールと同様、state オプションに absent を指定します。

```
lineinfile: dest=/tmp/testfile line=hoge state=absent
```

また、正規表現を使って行を置換することも可能です。

マッチする行の条件を regexp オプションで指定します。もしマッチする行が複数あった場合、最後にマッチした行が置換されます。

以下は#Port で始まる行を置換する例です。regexp オプションでマッチする条件を指定しています。

```
var:
```

```
  ssh_port: 10022
tasks:
  lineinfile: dest=/etc/ssh/sshd_config regexp="^#Port " line="Port {{ ssh_port }}"
backrefs=yes
```

backrefs オプションについては必須ではありませんが、このオプションに yes を指定しない場合、もし正規表現にマッチしない場合に行が追加されてしまいます。

べき等性の確保のために、regexp オプションを指定して置換を行う場合は、backrefs オプションを yes にしておいたほうがよいでしょう。

5.1.3　テンプレートファイルに値を設定して転送する

1 行ずつ設定ファイルを置換するのではなく、ファイルを丸ごと置き換えたい、ただし一部の値は変数で変えたい、というようなケースもあります。

その場合は、template モジュールを使うことで、プレイブックで定義した変数をファイルに埋め込んでコピーする事ができます。

管理ホストにある sshd_config.j2 というファイルをテンプレートとして使い、そのファイル内の{{ssh_port}}の部分を変数に置換し、被管理ホストの/etc/ssh/sshd_config に転送する例は以下の通りとなります。

・sshd_config.j2

```
Port {{ ssh_port }}
```

・プレイブックの記述

```
  vars:
    ssh_port=10022
  tasks:
    template: src=sshd_config.j2 dest=/etc/ssh/sshd_config
```

{{ ssh_port }} の部分が 10022 に置換されて被管理ホストに転送されます。

なお、テンプレートファイルとして j2 という見慣れない拡張子を使いましたが、このテンプレートファイルは Jinja2(http://jinja.pocoo.org/docs/)というテンプレート言語で解析されるため、この言語に合わせて拡張子は.j2 としています。

5.1.4　パッケージをインストールする

HTTP サーバや DB サーバなど、必要なパッケージをインストールする場合、パッケージマネージャ毎に用意されたモジュールを使います。

被管理ホストが CentOS で yum を使ってパッケージをインストールする場合、そのまま yum というモジュールが使えます。httpd をインストールする場合は以下のように書きます。

```
yum: name=httpd state=latest
```

Ubuntu のように apt-get を使う場合、apt というモジュールを使います。

```
apt: name=apache2
```

上記のように、被管理ホストの OS が異なる場合、同じモジュールを使う事ができません。1 つのプレイブックで複数の OS に対応したい場合は、次章で説明する条件分岐で対応できます。

5.1.5　サービスを起動/停止する

デプロイを行った後 HTTP サーバを再起動する、というような操作はよくあります。そのようなときは service モジュールを使います。

httpd サービスを起動した状態にする例は以下のとおりです。

```
service: name=httpd state=started
```

停止する場合は state に stopped を、再起動する場合は restarted を指定します。

5.1.6　任意のコマンドを実行する

すでに既存のシェルスクリプトの資産があり、Ansible に全て移行するのが現実的でないという場合もあるでしょう。その場合、shell モジュールを使うことで被管理ホストで任意のコマンドを実行することができます。

そのため、一部の処理は既存のスクリプトを使い、今後作成していく部分は Ansible のモジュールを使ってべき等性を確保する、ということができます。

被管理ホストの操作ユーザのホームディレクトリにある hoge.sh を実行するモジュール例は以下のとおりです。

```
shell: ~/hoge.sh
```

6 プレイブックの条件制御

6.1 条件分岐

パッケージマネージャのように、被管理ホストの環境によって異なるモジュールを実行しなければならない場合、when 節を使うことで、そのモジュールを実行する条件を設定できます。

被管理ホストのパッケージマネージャに応じて、タスクを実行するかどうかを判断するプレイブックは以下の通りです。

```
- hosts: example_cent:example_ubuntu
  sudo: yes
tasks:
   - yum: name=httpd state=latest
     when: ansible_pkg_mgr == "yum"
   - apt: name=apache2
     when: ansible_pkg_mgr == "apt"
```

このプレイブックを実行すると、以下のように、when 節の条件に当てはまらない場合は skipped という表記で実行されなかったことが分かります。

```
TASK: [yum name=httpd state=latest]
*********************************************
skipping: [example_ubuntu]
ok: [example_cent]

TASK: [apt name=apache2]
*****************************************************
skipping: [example_cent]
ok: [example_ubuntu]
```

ここでは、ansible_pkg_mgr という変数の値を見て、使うべきパッケージマネージャの判定をしています。

この変数は明示的に宣言していませんが、実は Ansible は実行時に自動で被管理ホストの情報を収集し、対応する変数に入れてくれています。収集されている変数については、以下のコマンドを実行することで確認ができます。

```
ansible ホスト名 -m setup
```

6.2 繰り返し実行

一度に複数のユーザを追加したり、複数のパッケージをインストールしたりするという場合には、繰り返し実行が利用できます。

以下は sora と maya というユーザを追加する例です。

```
tasks:
  user: name={{ item }}
  with_items:
    - sora
    - maya
```

with_item 要素にシーケンスを渡してあげると、その値が item 変数に設定されます。あらかじめ変数でシーケンスを作っておき、with_items に渡すことも可能です。

```
vars:
  user_names:
    - sora
    - maya
tasks:
  user: name={{item}}
  with_items: user_names
```

シーケンスではなく、マッピングを繰り返しの値に使うことも可能です。以下は、ユーザ毎に違うグループ、パスワードを指定してユーザを作成する例です。マッピングのキーは item.key 変数に、マッピングの値は item.value に設定されます。

```
vars:
  users:
    sora:
      group: wheel
      pass: $1$sugar$Q9IYVtr/TqOmI0Xu12
    maya:
      group: cow
      pass: $1$sugar2$Q9IYVtr/TqOmI0X
tasks:
  - user: name= {{ item.key }} password={{ item.value.pass }}
group={{item.value.group }}
    with_dict: users
```

7 TIPS

最後にいくつかの Tips を紹介します。

7.1 はまりどころ

Ansible の実行でエラーが発生したり、処理が止まってしまったりする場合は、以下のような原因が考えられます。

- パラメータ名をタイプミスしている

  インベントリファイル内のパラメータでタイプミスをしても、Ansible は警告やエラーを出力しません。正しい設定のはずなのに接続に失敗する場合は、パラメータ名のタイプミスを調べてみるとよいでしょう。ansible を ansilbe とタイプミスするのはありがちです。

- 被管理ホストに python3 のパスが通っている

  被管理ホストが python2 ではなく python3 へのパスが通っている場合、以下のようなエラーが表示され接続できません。

```
www.example.com | FAILED >> {
    "failed": true,
    "msg": "    File
¥"/tmp/ansible-tmp-1413772898.25-181094223988833/ping¥", line 442¥r¥n
except OSError, e:¥r¥n                          ^¥r¥nSyntaxError: invalid
syntax¥r¥n",
    "parsed": false
}
```

  この場合、インベントリファイルに以下のように ansible_python_interpreter パラメータを指定し、python2 へのパスを指定します。

```
www.example.com ansible_python_interpreter=/usr/bin/python2
```

- プレイブックで sudo を行うように記述しているが、sudo のパスワードを指定していない

このような状態の時、被管理ホストは sudo のパスワードの入力待ちで固まってしまいます。インベントリファイルで ansible_sudo_pass オプションを使って sudo 時のパスワードを指定するか、ansible-playbook の実行時に --ask-sudo-pass 引数を指定してください。

上記にあてはまらないエラーの場合は、ansible コマンドの実行時に-vvv 引数をつけて実行してみましょう。この引数を指定すると、詳細な実行情報が出力されるようになるため、どこでエラーになっているかのヒントとなります。

7.2 既存の SSH 接続の設定を使う

被管理ホストに対し、普段 SSH で接続している場合.~/ssh/config ファイルなどに接続に使うユーザや認証設定が存在するかと思います。この場合、インベントリファイルに同じことを書かずに、既存の config の設定を使うことも可能です。

インベントリファイルに ansible_connection パラメータを追加し、ssh を指定するようにしてみてください。

```
www.example.com ansible_connection=ssh
```

これで接続ユーザや利用する認証鍵を指定しなくても、SSH の config ファイルから読み取ってくれます。

ただし、CentOS 6.5 では、上記パラメータを指定すると以下のエラーが発生します。

```
www.example.com | FAILED => using -c ssh on certain older ssh versions may not support ControlPersist, set ANSIBLE_SSH_ARGS="" (or ssh_args in [ssh_connection] section of the config file) before running again
```

この場合は、上記エラーメッセージの通り、環境変数 ANSIBLE_SSH_ARGS をセットすることで、動作するようになります。

```
export ANSIBLE_SSH_ARGS=""
```

8 まとめ

本資料では、シェルスクリプトを使ってサーバ管理をしている人向けに、よりサーバ管理を行いやすくするための手段として Ansible という構成管理ツールを紹介しました。Ansible はシンプルさを特徴とする構成管理ツールで、何度実行しても最終的な状態が同じ状態となる、べき等性という性質を持った処理が多く用意されており、プログラマでなくともその処理を組み合わせて処理の自動化が行いやすくなっています。また、管理されるサーバに特別なクライアントソフトのインストールが不要なため、制約がある環境下でも利用が可能です。

さらに、Ansible のインストール方法と、Ansible の利用に必要な 2 つのファイル、被管理ホストの情報を記述するインベントリファイルと、被管理ホストで行う処理を記述するプレイブックの作成例を示しました。

そして、環境構築やデプロイで使うであろう基本的なモジュールについても紹介しました。今回紹介したもの以外にも、DB 操作や git リポジトリからのチェックアウトなど様々なモジュールが用意されており、これらのモジュールを組み合わせて使うことで、既存のシェルスクリプトの処理を、べき等性を持った処理に置き換えることができます。

本資料をきっかけに現状のサーバ管理の問題を解消し、よりよいサーバ管理を行っていただければ幸いです。

2014/10/28

開発部　今泉 俊幸