Ansible ハンズオン on AWS

【ハンズオンの流れ】

- AWSへのアクセス情報はスタッフから受け取ってください
- 開始時間になりましたら、ハンズオンの流れなどを説明します（5-10分程度）
- 説明終了後、本テキストを見ながら作業を進めてください
- 作業で解らない点があれば、お気軽に近くのスタッフを呼んでください
- 休憩時間は設定していないため、区切りが良いところで各自休憩してください
- ハンズオン**終了時刻は16:30**です
- ハンズオン終了後、内容をまとめたセッションを行います

【作業時の注意】

- YAMLファイルを編集する時は**インデントに注意**してください
- 設定を**コピーする時はGitHubからダウンロードしたテキストファイルから**実施してください（Googleドキュメントからは行わないでください）
- Githubにあるサンプルコード（wordpress.yaml）は各ステップ毎に作成しています（例: step3_wordpress.yml）。各ステップでの完成形となりますので参考にしてください。
- Step5以降でのタスク定義はまとめて記述して実行するとエラーが発生した時にデバッグが困難です。1〜2のタスクを記述し実行するようにしましょう

【参考】

- Ansible のドキュメント（モジュール）
  http://docs.ansible.com/ansible/modules_by_category.html
- GitHub
  https://github.com/watanabeshuji/devio2017-ansible-handson
- ブログ（AnsibleのためのYAML入門）
  http://dev.classmethod.jp/server-side/ansible/getting-start-yaml-for-ansible/
- ブログ（Ansibleハンズオン on AWS #cmdevio2017）
  http://dev.classmethod.jp/server-side/ansible/devio2017-ansible-handson

Step0. 環境の確認 (0min)

ハンズオン参加者は、着席後、以下の項目を確認してお待ちください。
不明な点などはお近くのスタッフ（アシスタント）に声をかけてください。

- WiFi設定を行い、インターネットに接続ができることを確認してください
- コンソールを開き、**ansible 2.3がインストールされていることを確認してください**
  $ ansible --version

Step1. EC2インスタンスの作成 (10min)

目標

AWSのマネジメントコンソールにログインし、EC2インスタンスを作成します。

背景

Ansibleで構成管理を行うサーバをAWSに作成します。

手順

配布されたIAMユーザのログイン情報を利用します。

1. ウェブブラウザでログイン用のURLにアクセスします
2. IAMユーザとパスワードを入力し、ログインします
3. サービスメニューから「EC2」を選択します
4. 「キーペア」メニューを選択し、「キーペアの作成」をクリックします
5. 指定されたキーペア名（IAMユーザ名-key）を入力し、キーペアを作成します
   この時、秘密鍵がダウンロードされます
6. 「インスタンス」メニューを選択し、「インスタンスの作成」をクリックします
7. Step1：クイックスタートタブのAmazonLinuxを選択します
8. Step2：インスタンスタイプはt2.microを選択し、次の手順をクリックします
9. Step3、Step4：設定変更は不要です。次の手順をクリックします
10. Step5：Nameタグをホスト名（WordPress_IAMユーザの数字２桁）で設定します
11. Step6：セキュリティグループでは、「既存のセキュリティグループ」を選択し、「devio2017」を選択します
12. Step7：内容を確認し、起動をクリックします
13. キーペアでは、作成したキーペアを選択し、インスタンスを起動します
14. インスタンスの詳細を確認し、Public IPアドレスを確認します

AWSマネジメントコンソールの扱いに不慣れな方は、お気軽にスタッフをお呼びください。

Step2. SSH接続の準備 (10min)

目標

ssh_config を作成し、sshコマンドでサーバに接続します。

背景

Ansibleでは、リモートサーバにsshで接続し、ミドルウェアなどの構成管理を行います。

手順

1. プロジェクトディレクトリを作成し、ssh_configを作成します
   ホスト名とIPアドレスは、各作業者の環境を設定してください
   キーペアの作成でダウンロードした秘密鍵へのパスを設定します
   また、秘密鍵はパーミッションを600に設定してください
2. ssh コマンドでサーバに接続します
   $ ssh -F ssh_config WordPress
3. EC2インスタンスに接続したことを確認したならば、yum updateを行います
   $ sudo yum -y update
4. EC2インスタンスからログアウトします
   $ exit

ssh_config

```
Host *
  StrictHostKeyChecking no
  UserKnownHostsFile /dev/null

Host WordPress
  HostName xx.xxx.xx.xxx
  User ec2-user
  IdentityFile PATH_TO_KEY_FILE
```

TOPIC: EIPの再アタッチとSSH

SSHでは、接続時ホストキーをチェックする機能があります。あるホストに接続した時、KonwnHostsとしてファイルに記録します。同じホスト名で異なるホストキーを持つホストに接続しようとした場合、警告メッセージが表示されて接続できません。
これは一般的なホストであれば良い機能です。しかし、AWSでは、特に開発・検証中、EC2インスタンスを頻繁に再作成し、同じEIPをアタッチします。すると、ホストキーチェックに失敗してしまいます。KnownHostsFileの削除などが必要です。
このため、ホストキーのチェック機能を無効にしています。

TOPIC: ssh_configの管理

現実のプロジェクトでは、AnsibleのPlaybookはGit等のバージョン管理システムで管理します。この時、ssh_configもあわせて管理しましょう。ssh_confgがあれば、他のエンジニアも秘密鍵があれば簡単にサーバに接続することができます。

Step3. Ansible Playbookの準備 (10min)

目標

ansible-playbook コマンドでサーバに接続し、構成管理を行う準備を整えます。

背景

Ansibleの設定はansible.cfgに、接続するサーバの設定はinventoryファイルに、構成の設定はPlaybookファイルに記述します。

手順

1. ansible.cfgを作成します。
2. inventoryファイル「hosts」を作成します。
3. Playbookファイル「wordpress.yml」を作成します。
4. ansible-playbookコマンドを実行し、動作確認します。
   $ ansible-playbook wordpress.yml

ansible.cfg

```
[defaults]
inventory = hosts
retry_files_enabled = False

[privilege_escalation]
become = True

[ssh_connection]
control_path = %(directory)s/%%h-%%r
ssh_args = -o ControlPersist=15m -F ssh_config -q
scp_if_ssh = True
```

- inventory （＝接続するホスト定義）はhostsファイルを参照（デフォルト）
- ansibleの実行に失敗した場合、retryファイルを作成しますが、邪魔なので無効
- ansible接続後、デフォルトでrootユーザで実行する（become）
- その他SSH接続の設定を追加（ssh_configを参照する）

hosts

```
[wordpress-servers]
WordPress
```

- Ansibleで参照するホストグループ名を「wordpress-servers」と定義
- 名前解決できるホスト名やIPアドレスを定義
  - 同じ構成のサーバを複数定義できる
  - IPアドレスでも可能だがssh_configと連携することを推奨

wordpress.yml

```
- hosts: wordpress-servers
  roles: []
```

- hostsにInventoryファイルで定義したホストグループ「wordpress-servers」を指定

- rolesに構成内容を定義するが、現時点では空

TOPIC: check オプションによるDryRun

ansible-playbookコマンドには、いわゆるDryRunのためのオプションが用意されています。コマンド実行時に「--check」オプションを付与することで、サーバへの変更を行わず、**変更が必要かどうかのチェックのみを実施**します。

$ ansible-playbook wordpress.yml --check

チェックオプションは次のようなケースで有効です。

- Playbookを変更した場合、サーバに変更が行われる部分を確認する
- Playbookを変更していない場合、サーバに変更が行われないことを確認する

ただし、どちらのケースでも、Playbookが「べき等性」を保っていることが前提です。
べき等性に関して、詳細などを知りたい方は以下のブログを参照ください。
Ansibleの冪等性とPlaybook ｜ Developers.IO

Step4. 共通設定用Roleの追加 (15min)

目標

サーバのタイムゾーンやロケールの設定など、共通となる構成を提供します。

背景

Ansibleでは構成設定をRoleという単位に分割できます。Roleは一度作れば再利用可能なのでよく利用する構成はRoleとして管理します。

手順

1. ディレクトリ rolesを作成する
2. ディレクトリ roles/commonを作成する
3. ディレクトリ roles/common/tasksを作成する
4. roles/common/tasks/main.yml を作成する
   Gitから内容を参照してください
5. Playbookファイル「wordpress.yml」を更新する
6. サーバにSSHログインし、現在の設定を確認する
   $ date
   $ exit
7. ansible-playbookコマンドを実行し、サーバに反映する
   $ ansible-playbook wordpress.yml
8. 再度サーバにSSHログインし、設定が変更されたことを確認する
   $ date

wordpress.yml

```
- hosts: wordpress-servers
  roles:
    - common
```

TOPIC: 手順2完了時のディレクトリ構成は以下となります。

```
$ tree
.
├── ansible.cfg
├── hosts
├── roles
│   └── common
```

```
│       └── tasks
│           └── main.yml
├── ssh_config
└── wordpress.yml
```

Step5. MySQL Serverのインストール（15min）

目標

サーバにMySQLサーバをインストールします。

背景

WordPressで利用するデータベースをサーバにインストールします。

手順

1. ディレクトリ roles/mysql56-server を作成する
2. ディレクトリ roles/mysql56-server/tasks を作成する
3. ディレクトリ roles/mysql56-server/handlers を作成する
4. ディレクトリ roles/mysql56-server/templates を作成する
5. roles/mysql56-server/tasks/main.yml を作成する
6. roles/mysql56-server/handlers/main.yml を作成する
7. roles/mysql56-server/templates/my.cnf を作成する
8. ディレクトリ groups_vars を作成する
9. group_vars/wordpress-servers.yml を作成する
10. Playbookファイル「wordpress.yml」を更新する
11. ansible-playbookコマンドを実行し、サーバに反映する
    $ ansible-playbook wordpress.yml
12. サーバにSSHログインし、設定が変更されたことを確認する
    $ sudo service mysqld status
    $ cat /etc/my.cnf
    $ mysql -h localhost -u master --password=Pass1234 wp

roles/mysql56-server/tasks/main.yml

```
---
- name: mysql56-server installed
  yum:
    name: mysql56-server
- name: "/etc/my.cnf"
  template:
    src: my.cnf
    dest: /etc/my.cnf
    backup: yes
    owner: root
    group: root
    mode: "0644"
  notify:
    - restart mysqld service
- name: mysqld service started
  service:
    name: mysqld
    state: started
```

```
    enabled: yes

- name: "install MySQL-python27 for DB module"
  yum:
    name: MySQL-python27
- name: "DB master user"
  mysql_user:
    name: "{{ db_user }}"
    password: "{{ db_password }}"
    priv: '*.*:ALL'
- name: "Database"
  mysql_db:
    name: "{{ db_name }}"
```

- yumモジュールでは、nameで指定したパッケージを管理する
- 設定ファイル(/ete/my.cnf）はtemplateモジュールで設定する
    - templateモジュールでは、templatesディレクトリのファイルをサーバにコピーする
    - templateモジュールでは、Jinja2テンプレートが利用できる
    - backup オプションを有効にすることで、変更前のファイルを残す
    - コピーしたファイルのownerやgroup、パーミッションなどを指定できる
- serviceモジュールでは、サービスの状態を管理する
    - stateオプションは、started, stopped, restarted などが指定できる
    - enabledオプションで、chkconfigの設定を追加できる
- mysql_user, mysql_db モジュールでは、データベースに接続し指定されたユーザやデータベースを作成する
    - {{ と }} で囲まれたフレーズは変数を展開を行う
    - 変数はグループ変数などで定義する
    - “（ダブルクォート）で囲まなければならない

roles/mysql56-server/handlers/main.yml

```
---
- name: restart mysqld service
  service:
    name: mysqld
    state: restarted
```

- handler は設定ファイルの更新時にサービスを再起動するような用途で使います
- タスクのnotify句に記述することで、そのタスクで変更があった場合、Playbook実行の最後のタイミングでhandlerが、1回だけ実行されます

roles/mysql56-server/templates/my.cnf

```
[mysqld]
datadir=/var/lib/mysql
socket=/var/lib/mysql/mysql.sock
symbolic-links=0
character-set-server=utf8
skip-character-set-client-handshake

[mysqld_safe]
log-error=/var/log/mysqld.log
```

```
pid-file=/var/run/mysqld/mysqld.pid

[mysql]
default-character-set=utf8
```

group_vars/wordpress-servers.yml

```
---
db_user: master
db_password: Pass1234
db_name: wp
```

wordpress.yml

```
- hosts: wordpress-servers
  roles:
    - common
    - mysql56-server
```

Step6. Apache24のインストール (10min)

目標

サーバにApache2.4をインストールします。

背景

WordPressで利用するApache2.4をサーバにインストールします。

手順

1. ディレクトリ roles/apache24 を作成する
2. ディレクトリ roles/apache24/handlers を作成する
3. ディレクトリ roles/apache24/tasks を作成する
4. roles/apache24/handlers/main.yml を作成する
5. roles/apache24/tasks/main.yml を作成する
6. Playbookファイル「wordpress.yml」を更新する
7. ansible-playbookコマンドを実行し、サーバに反映する
   $ ansible-playbook wordpress.yml
8. サーバにSSHログインし、設定が変更されたことを確認する
   $ sudo service httpd status
9. ウェブブラウザからPublic IPアドレスにアクセスし、Apacheのテストページが表示されることを確認する

roles/apache24/handlers/main.yml

```
---
- name: restart apache24 service
  service:
    name: httpd
    state: restarted
```

- handler はStep7：roles/wordpress/tasks/main.yml内で使います

roles/apache24/tasks/main.yml

```
---
- name: apache24 installed
  yum:
    name: httpd24
- name: httpd service started
  service:
    name: httpd
    state: started
    enabled: yes
```

wordpress.yml

```
---
- hosts: wordpress-servers
  roles:
    - common
    - mysql56-server
    - apache24
```

TOPIC: Apacheの設定ファイル

Roleを作る時、設定ファイルの配置をそのRoleで行うか、別のRoleで行うかは悩ましい問題です。自分の考えでは、再利用を意識する場合はそのRoleで行わず、再利用を意識しない場合はそのRoleで行うのが良いと思います。
これは一見、逆ではないかと感じます。しかし、Apacheの設定を細かく汎用化していくと、そのApache Roleの変数が際限なく増えていきます。条件分岐なども複雑になり、Roleのメンテナンスにも支障がでます。それよりは設定ファイルをアプリケーション固有のRoleで設定した方がスッキリします。一方、本ハンズオンのMySQLサーバのように、固有設定も含めたRoleを作っても良いでしょう。ただし、このようなRoleは再利用しにくくなります。

Step7. WordPressのインストール (20min)

目標

サーバにWordPressをインストールします。

背景

WordPressをサーバにインストールします。

手順

1. ディレクトリ roles/wordpress を作成する
2. ディレクトリ roles/wordpress/tasks を作成する
3. ディレクトリ roles/wordpress/templates を作成する
4. group_vars/wordpress-servers.yml を更新する
5. roles/wordpress/tasks/main.yml を作成する
6. roles/wordpress/templates/wordpress.cnf を作成する
7. Playbookファイル「wordpress.yml」を更新する
8. ansible-playbookコマンドを実行し、サーバに反映する
   $ ansible-playbook wordpress.yml
9. ウェブブラウザからPublic IPアドレスにアクセスしWordPressの動作を確認する

★注意★

main.yml に定義するタスクをまとめて記述すると、エラー発生時にデバッグが困難です。
「1〜2タスクを記述して流す」といった形で試してください。

group_vars/wordpress-servers.yml

```
---
db_user: master
db_password: Pass1234
db_name: wp
wp_path: /opt/wordpress
wp_download_url: https://ja.wordpress.org/wordpress-4.7.5-ja.zip
```

roles/wordpress/tasks/main.yml

```
---
- name: php70 installed
  yum:
    name: php70,php70-mysqlnd

- name: "check {{ wp_path }}' if exist"
  stat:
    path: "{{ wp_path }}"
  register: stat_wp_home
- name: download wordpress
  unarchive:
    src: "{{ wp_download_url }}"
```

```
    dest: /opt
    remote_src: yes
    owner: apache
    group: apache
  when: stat_wp_home.stat.exists == false

- name: "check {{ wp_path }}/wp-config.php' if exist"
  stat:
    path: "{{ wp_path }}/wp-config.php"
  register: stat_wp_config
- name: "rename {{ wp_path }}/wp-config-sample.php to {{ wp_path }}/wp-config.php"
  shell: "mv {{ wp_path }}/wp-config-sample.php {{ wp_path }}/wp-config.php"
  when: stat_wp_config.stat.exists == false

- name: "DB_NAME: {{ wp_path }}/wp-config.php"
  replace:
    path: "{{ wp_path }}/wp-config.php"
    regexp: "{{ item.regexp }}"
    replace: "{{ item.replace }}"
    backup: yes
  with_items:
    - { regexp: "^define\\('DB_NAME'.+$",     replace: "define('DB_NAME', '{{ db_name }}');"}
    - { regexp: "^define\\('DB_USER'.+$",     replace: "define('DB_USER', '{{ db_user }}');"}
    - { regexp: "^define\\('DB_PASSWORD'.+$", replace: "define('DB_PASSWORD', '{{
db_password }}');"}
    - { regexp: "^define\\('DB_HOST'.+$",     replace: "define('DB_HOST', 'localhost');"}

- name: "/etc/httpd/conf.d/wordpress.conf"
  template:
    src: wordpress.conf
    dest: /etc/httpd/conf.d/wordpress.conf
    owner: root
    group: root
    mode: "0644"
  notify:
    - restart apache24 service
```

- yumモジュールではカンマ区切りで複数のモジュールを指定できます。
- statモジュールはファイルやパスの存在チェックを行うモジュールです。
  register句で指定した変数に結果を格納します。
  when句と併用し、パスが存在しない場合の処理で利用します。
- unarchiveモジュールは圧縮ファイルをサーバに転送し解凍します。
  remote_src オプションを利用すると指定URLからファイルをダウンロードします。
- when句を指定すると、条件付きでタスクを実行します。
  ここではstatモジュールの判定結果からソースのダウンロードを実施します。
- shellモジュールはシェルスクリプトを実行します。
  このモジュールは常にサーバに変更を行うとみなされます。
  冪等性を保つPlaybookを記述する際は、注意して利用しなければなりません。
- replaceモジュールは設定ファイル等の置換を行います。
  対象ファイルはpathで指定します。
  置換対象の文字列は正規表現を使いregexpで指定します。
  置換後の文字列はreplaceで指定します。

- with_itemsはAnsibleでのループ処理を行います。
  リストの各項目についてタスクが実行されます。
  タスクでは変数「item」でitemsの各項目を参照できます。

roles/wordpress/templates/wordpress.cnf

```
<VirtualHost *:80>
 DocumentRoot {{ wp_path }}
 <Directory "{{ wp_path }}">
   AllowOverride All
   Require all granted
 </Directory>
</VirtualHost>
```

wordpress.yml

```
---
- hosts: wordpress-servers
  roles:
    - common
    - mysql56-server
    - apache24
    - wordpress
```