Configuration Management

Module Overview

* What is Configuration Management ?
* Configuration Management Tools
* Ansible Basics
* Course Review
* Final Exam

Product Deployment

* Unsuccessful deployment, what went wrong with product delivery ?
* Configuration management helps you put all the pieces together.
* Configuration management is a comprehensive engineering process which keeps track and logs of all the changes which are made on the software until the product goes live
* Why is it important ?
  * Change is known to everyone
  * Fast on recovery
  * Improvisation
  * Reduces System Downtime
  * Improve Resource Utilization
  * Consistency in the infrastructure

Ansible Introduction

* Ansible is IT automation tool
* Main goals
  * Simplicity and ease-of-use
  * Manages machines in an agent less manner
  * De-centralized
* Building Blocks
  * Playbooks
  * Variables
  * Conditionals
  * Loops
  * Roles
  * Modules

Installing Ansible

* Install Ansible Amazon Cloud Instance
  * *sudo amazon-linux-extras install ansible2*
* Verify Install
  * *ansible -version*
* Assign IAM role to EC2
  * Create New Role with AmazonEC2FullAccess policy
  * Attach role to running instance
  * Create playbook file and run the command:
    * *ansible-playbook <<NameOfYAML File>>*

Understanding YAML

* YAML File represents configuration data

```
<Servers>
    <Server>
        <name>Server1</name>
        <owner>John</owner>
        <created>12232012</created>
        <status>active</status>
    </Server>
</Servers>
```

```
{
    Servers: [
        {
        name: Server1,
        owner: John,
        created: 12232012,
        status: active,
        }
    ]
}
```

```
Servers:
    -   name: Server1
        owner: John
        created: 12232012
        status: active
```

* Key value Pairs, Array/List & Dictionary/Map

Key Value Pair

```
Fruit: Apple
Vegetable: Carrot
Liquid: Water
Meat: Chicken
```

Array/Lists

```
Fruits:
-   Orange
-   Apple
-   Banana

Vegetables:
-   Carrot
-   Cauliflower
-   Tomato
```

Dictionary/Map

```
Banana:
    Calories: 105
    Fat: 0.4 g
    Carbs: 27 g

Grapes:
    Calories: 62
    Fat: 0.3 g
    Carbs: 16 g
```

* Dictionary – Unordered, List – ordered
* Lab: https://kodekloud.com/p/ansible-practice-test/?scenario=questions_yaml

Ansible Playbooks

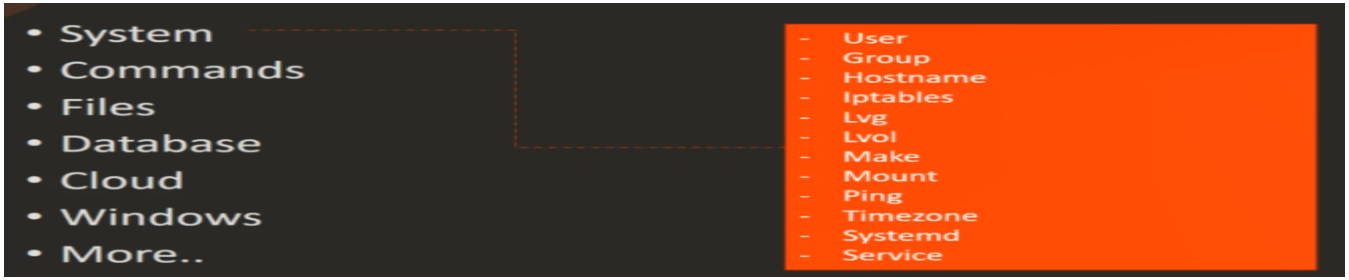
Basis for a really simple configuration management. A minimum of syntax, which intentionally tries to not be a programming language or script, but rather a model of a configuration or a process

* Playbooks can declare configurations, but they can also orchestrate steps of any manual ordered process
* Can launch tasks synchronously or asynchronously
* To be kept in source control and used to push out your configuration or assure the configurations of your remote systems are in spec.
* Playbooks are expressed in YAML format
* Each playbook is composed of one or more ‘plays’ in a list.
* The goal of a play is to define activities, represented by things ansible calls tasks. At a basic level, a task is nothing more than a call to an ansible module
* By composing a playbook of multiple ‘plays’, it is possible to orchestrate multi-machine deployments, running certain steps on all machines in the webservers group, then certain steps on the database server group, then more commands back on the webservers group, etc
* Lab : https://kodekloud.com/p/ansible-practice-test/?scenario=questions_ansible_playbook

Modules

Discrete units of code that can be used from the command line or in a playbook task

* Modules should be idempotent, and should avoid making any changes if they detect that the current state matches the desired final state
* Module Documentation: *ansible-doc yum*



* Available modules: *ansible-doc -l*
* *Lab:* https://kodekloud.com/p/ansible-practice-test/?scenario=questions_ansible_modules

Ansible Variables

Ansible variables deals with differences between systems

* Store values that varies with different items
* Variables can be defined in playbook, inventory or variables file
* Using variables

Playbook.yml

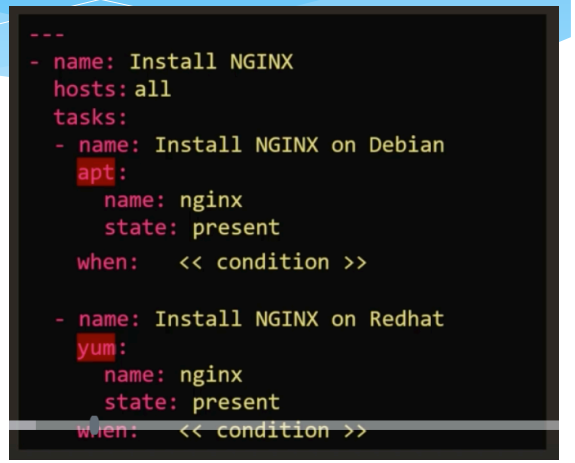
```
-
  name: Add DNS server to resolv.conf
  hosts: localhost
  vars:
    dns_server: 10.1.250.10
  tasks:
      - lineinfile:
          path: /etc/resolv.conf
          line: 'nameserver {{ dns_server }}'
```

* Avoid hard coding values in playbook
* Lab: https://kodekloud.com/p/ansible-practice-test/?scenario=questions_ansible_variables

Conditionals

* Often the result of a play may depend on the value of a variable, fact (something learned about the remote system), or previous task result.
* Sometimes you will want to skip a particular step on a particular host. This is easy to do in ansible, using when clause
* condition is any check which we can perform

```
tasks:
  - name: "shut down CentOS 6 and Debian 7 systems"
    command: /sbin/shutdown -t now
    when: (ansible_facts['distribution'] == "CentOS" and
ansible_facts['distribution_major_version'] == "6") or
         (ansible_facts['distribution'] == "Debian" and
ansible_facts['distribution_major_version'] == "7")
tasks:
  - name: "shut down CentOS 6 systems"
    command: /sbin/shutdown -t now
    when:
      - ansible_facts['distribution'] == "CentOS"
      - ansible_facts['distribution_major_version'] == "6"
```



* Lab: https://kodekloud.com/p/ansible-practice-test/?scenario=questions_ansible_conditionals

Ansible loops

* Sometimes you want to repeat a task multiple times. In computer programming, this is called a loop.
* Iterate over list of objects

```
- name: add several users
  user:
    name: "{{ item.name }}"
    state: present
    groups: "{{ item.groups }}"
  loop:
    - { name: 'testuser1', groups: 'wheel' }
    - { name: 'testuser2', groups: 'root' }
```

* Lab: https://kodekloud.com/p/ansible-practice-test/?scenario=questions_ansible_loops

* Iterate over a simple list

```
- name: add several users
  user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  loop:
    - testuser1
    - testuser2
```

or `loop: “{{ somelist }}”`

* Equivalent to :

```
- name: add user testuser1
  user:
    name: "testuser1"
    state: present
    groups: "wheel"

- name: add user testuser2
  user:
    name: "testuser2"
    state: present
    groups: "wheel"
```

*

Looping with with_*

* with_<lookup> same as loop
* Iterate over list of objects
* with_<lookup> relies on lookup plugins
* loop is equivalent to with_list and is the best choice for simple loops
* common lookup examples, with_file, with_url, with_mongodb, etc.
* lookup plugins allow ansible to access data from outside sources

* Iterate over a simple list

```
- name: add several users
  user:
    name: "{{ item }}"
    state: present
    groups: "wheel"
  with_items:
     - testuser1
     - testuser2
```

* Equivalent to :

```
- name: add user testuser1
  user:
    name: "testuser1"
    state: present
    groups: "wheel"

- name: add user testuser2
  user:
    name: "testuser2"
    state: present
    groups: "wheel"
```

*

Ansible roles

* Roles are ways for automatically loading certain vars files, tasks, handlers based on a known file structure
* Grouping content by roles allows easy sharing of roles with other users
* Roles expect files to be in certain directory names.
* Role must include at least one of these directories
* When in use, each directory must contain a main.yml, which contains relevant content
* Using roles:

```
---
- hosts: webservers
  roles:
    - common
    - webservers
```

* Using your own roles:
  * *ansible-galaxy init myrole*
* *ansible-galaxy list*
* *ansible-galaxy search mysql*
* *ansible-galaxy install mysql*

```
site.yml
webservers.yml
fooservers.yml
roles/
   common/
     tasks/
     handlers/
     files/
     templates/
     vars/
     defaults/
     meta/
   webservers/
     tasks/
     defaults/
     meta/
```

- `tasks` - contains the main list of tasks to be executed by the role.
- `handlers` - contains handlers, which may be used by this role or even anywhere outside this role.
- `defaults` - default variables for the role (see Using Variables for more information).
- `vars` - other variables for the role (see Using Variables for more information).
- `files` - contains files which can be deployed via this role.
- `templates` - contains templates which can be deployed via this role.
- `meta` - defines some meta data for this role. See below for more details.