ansible-workshop Documentation

Release 0.1

Praveen Kumar, Aditya Patawari

May 11, 2017

Contents

Contents:

CHAPTER 1

Introduction

Welcome to Ansible workshop.

Requirements

- A Fedora 24/25 virtual machine.
- Internet connection

Goal

To learn Ansible basics and create a simple Ansible playbook to install a web application and server.

CHAPTER 2

Basics of Ansible

What is Ansible?

Ansible is a modern IT automation tool which makes your life easier by managing your servers for you. You just need to define the configuration in which you are interested and ansible will go ahead and do it for you, be it installing a package or configuring a server application or even restarting a service. Ansible is always ready to manage your servers.

Why do we need it?

Managing a server is easy. Managing 5 is do'able. Managing hundreds or more is a painful task without automation. Ansible is designed to be simple and effective. You can create identical, replicable servers and clusters of servers in a painless and reliable manner.

What are the advantages of using it?

Ansible manages machines in an agent-less manner. You do not need to have anything installed on the client's end. However both push and pull mode are supported. Ansible is a security focused tool. It uses OpenSSH as transport protocol. Ansible scripts (commonly known as playbooks) are written in YAML and are easy to read. If needed, Ansible can easily connect with Kerberos, LDAP, and other centralized authentication management systems.

How to install Ansible?

We will install the Ansible by pip. Package managers like dnf, yum and apt can be used.

- On Fedora machines:

```
# dnf install ansible
```

- On CentOS machines

```
# yum install epel-release
# yum install ansible
```

CHAPTER 3

Inventory File

Inventory defines the groups of hosts which are alike in any way. For example, you would want to group your web servers in one group and application servers in another. A group can have multiple server and one server can be a part of multiple groups.

Name of group is enclosed in square brackets []. Server names can be their DNS names or IP addresses.

```
[webservers]
server1
[application]
server1
server2
```

By default, ansible looks for the inventory file at */etc/ansible/hosts*, but that can be modified by passing a *-i <inventory_path* to the ansible command line.

We can modify the way ansible connects to our hosts by supplying additional information in the inventory file.

```
[webservers]
server1 ansible_port=4242 ansible_user=adimania
[application]
server1
server2
[master]
localhost ansible_connection=local
```

A more exhaustive list of inventory parameters can be seen here - http://docs.ansible.com/ansible/intro_inventory.html#list-of-behavioral-inventory-parameters

There are times when you would want to pull the inventory from a cloud provider, or from LDAP, or you would want the inventory list to be generated using some logic, rather than from a simple text-based inventory list. For such purposes, we can use Dynamic Inventory, but that's a topic for another day.

CHAPTER 4

Modules

Modules are the executable plugins that get the real job done.

Usually modules can take "key=value" arguments and run in customized way depending up on the arguments themselves.

A module can be invoked from commandline or can be included in an Ansible playbook.

We will discuss playbooks in a minute but for now, let us see modules in action.

To use modules from the command line, we write ansible ad-hoc commands, like the following -

```
$ ansible all -m ping
```

Above example will use the ping module to ping all the hosts defined in the inventory. There are several modules available in ansible. Let us try another one.

```
$ ansible webservers -m command -a "ls"
```

In the above example, we use command module to fire ls command on the webservers group.

```
$ ansible -i inventory all -m command -a "iptables -F" --become --ask-become-pass
```

Here, we use the *command* module to flush iptables rules on all the hosts in the inventory, and we tell ansible to execute the command with sudo privileges using *–become* and ask us the sudo password using *–ask-become-pass*.

```
$ ansible all -m setup
```

Ansible gathers facts about the hosts the tasks are being run against, which can be used later in the playbook execution, we can see all facts using the command above.

See how to extract particular facts in the documentation of *setup* module. To see the documentation, run -

```
$ ansible-doc setup
```

Using *ansible-doc <module name>* we can check the documentation of any ansible module.

CHAPTER 5

Playbooks

Playbooks are a description of policies that you want to apply to your systems. They consist of a listing of modules and the arguments that will run on your system so that ansible gets to know the current state. They are written in YAML. They begin with "—", followed by the group name of the hosts where the playbook would be run.

Example:

```
---
hosts: localhost

- name: install nginx
  yum: name=nginx state=installed
```

The example above will install Nginx on our systems. Let us also install pip, flask and our flask app.

```
---
hosts: localhost

- name: install nginx
  yum: name=nginx state=installed

- name: install pip
  yum: name=python-pip state=installed

- name: install flask
  pip: name=flask

- name: fetch application
  git: repo=https://gist.github.com/c454e2e839fcb20605a3.git dest=flask-demo
```

Now we should also copy the config file for Nginx and systemd service file for our flask app. We will also define a couple of handlers. Handlers are executed if there is any change in state of the task which is supposed to notifies them.

When we will be done with the workshop, our final playbook will look something like this:

```
---
- hosts: localhost
  remote_user: fedora
  become: yes
  become_method: sudo
  vars:
    - server_port: 8080

  tasks:
    - name: install nginx
      yum: name=nginx state=installed

    - name: serve nginx config
      template: src=../files/flask.conf dest=/etc/nginx/conf.d/
      notify:
      - restart nginx

    - name: install pip
      yum: name=python-pip state=installed

    - name: install flask
      pip: name=flask

    - name: serve flask app systemd unit file
      copy: src=../files/flask-demo.service dest=/etc/systemd/system/

    - name: fetch application
      git: repo=https://gist.github.com/c454e2e839fcb20605a3.git dest=/opt/flask-demo
      notify:
        - restart flask app

    - name: set selinux to permissive for demo
      selinux: policy=targeted state=permissive

    handlers:
    - name: restart nginx
      service: name=nginx state=restarted

    - name: restart flask app
      service: name=flask-demo state=restarted
```

We can also skip a particular task or make a task execute only if a condition is met using the When statement.

```
tasks:
  - shell: yum provides */elinks
    when: ansible_os_family == "RedHat"
```

Suppose we have a list of items we have to iterate on for a particular task, we can use loops like the following

```
- name: add ssh users
  user:
    name: "{{ item }}"
    state: present
    generate_ssh_key: yes
  with_items:
     - sshuser1
     - sshuser2
     - sshuser3
```

We can also run certain tasks from a playbook by tagging them -

```
---
- hosts: localhost
  become: yes

  tasks:
  - name: install nginx
    yum: name=nginx state=present
    tags:
      - system

  - name: install pip
    yum: name=python-pip state=present
    tags:
      - system

  - name: install flask
    pip: name=flask
    tags:
      - dev
```

We can run the system tagged tasks by running *ansible-playbook playbook.yml –ask-become-pass –tags system*

We can skip the system tagges tasks by running *ansible-playbook playbook.yml –ask-become-pass –skip-tags system*

CHAPTER 6

Variables

There are times when we have a bunch of similar servers but they are not exactly the same. For example, consider webservers. They may all run Nginx and might have same set of users accounts and ACLs but they may vary slightly in configuration. For such scenarios, variables are very helpful. A variable name can only consist of letters, numbers, and underscores and should always start with a letter. Below is an example of a variable definition in a playbook.

```
- hosts: webservers
  vars:
    http_port: 80
```

We can save the result of a command to a variable and use that somewhere else in the playbook, e.g. in conditionals.

```
tasks:

    - name: list contents of directory
      command: ls mydir
      register: contents

    - name: check contents for emptiness
      debug: msg="Directory is empty"
      when: contents.stdout == ""
```

CHAPTER 7

Condition handling

Conditionals help us evaluate a variable and take some action on the basis of the outcome.

Example:

```
---

- hosts: localhost
  vars:
    - state: false

  tasks:
    - shell: echo good state
      when: state
```

CHAPTER 8

System Configurations

Important modules

- Software installation: yum, apt, pip
- Services management: service
- Selinux management: selinux
- User manamgement: user

Examples:

```
- name: install git
  yum: name=git state=installed

- name: start nginx
  service: name=nginx state=started

- name: put selinux to enforcing mode
  selinux: policy=targeted state=enforcing

- name: create the user
  user: name=aditya
```

CHAPTER 9

Application Orchestration

Ansible can be used to deploy applications. A very obvious strategy is to package the application into rpm or deb package and use yum or apt module of ansible to install the application. Handlers can be used to reload or restart the application post the deploy. Alternatively, git module can be used to clone or pull the code from a repository and install or update the application.

Example:

```
- name: fetch application
  git: repo=https://gist.github.com/c454e2e839fcb20605a3.git dest=/opt/flask-demo
```

CHAPTER 10

Roles

Ansible playbooks can get very, very long with time, and hence difficult to maintain. Also, if you would like to reuse a subset of tasks from a playbook, that would get difficult as the playbooks get bigger and bigger.

Ansible roles can help you with grouping content, managing playbooks for a large project.

Example project structure:

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

In a playbook, it would look like this:

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

This designates the following behaviors, for each role 'x':

- If roles/x/tasks/main.yml exists, tasks listed therein will be added to the play
- If roles/x/handlers/main.yml exists, handlers listed therein will be added to the play
- If roles/x/vars/main.yml exists, variables listed therein will be added to the play
- If roles/x/defaults/main.yml exists, variables listed therein will be added to the play
- If roles/x/meta/main.yml exists, any role dependencies listed therein will be added to the list of roles (1.3 and later)
- Any copy, script, template or include tasks (in the role) can reference files in roles/x/{files,templates,tasks}/ (dir depends on task) without having to path them relatively or absolutely

CHAPTER 11

Cloud Infra Provising

Ansible provide lots of module for different Cloud operators like AWS, Openstack, Rackspace, digitalOcean ...etc. to manage your cloud infra.

http://docs.ansible.com/ansible/list_of_cloud_modules.html

Here we have sample playbook for openstack cloud provider.

vars/main.yml

```
---

OS_USERNAME: user1
OS_PASSWORD: demo_password
OS_TENANT_NAME: user1
OS_AUTH_URL: http://172.29.236.7:35357/v2.0
KEY_NAME: controller-key
SHARED_NETWORK: 11d0eb17-7e18-4a7b-978d-d9475c64d0e0
FLAVOR: m1.tiny
OSIMG: cirros-0.3.3
INSTCNT: 3
INSTNAME: ansible-demo
```

tasks/main.yml

```
---

- name: Launch instances in tenant
  command: nova --os-username={{ OS_USERNAME }} --os-password={{ OS_PASSWORD }} --os-
↪tenant-name={{ OS_TENANT_NAME }}
          --os-auth-url={{ OS_AUTH_URL }} boot --flavor {{ FLAVOR }} --image {{ OSIMG␣
↪}} --nic net-id={{ SHARED_NETWORK }}
          --security-group default --key-name {{ KEY_NAME }} --min-count {{ INSTCNT }}
↪ {{ INSTNAME }}
```

you can use openstack-API instead of CLI to perform same task.

For DigitalOcean Droplet provisioning, refer to the respective directory in the root of this repository for playbooks.

Before running the playbooks, make sure you have done the following -

- *pip install dopy*
- Login to your DO account, and under API, generate your API token, and then export it to the environment as *DO_API_TOKEN=<token>*

CHAPTER 12

Custom Modules

Let we try to build a very basic module which will get and set system time. We will do it in step by step.

- Write a python script to get current time and print json dump.
- Write a python script to get time as argument and set it to system.

Test Module

```
git clone https://github.com/ansible/ansible.git --recursive
source ansible/hacking/env-setup
chmod +x ansible/hacking/test-module

ansible/hacking/test-module -m ./timetest.py

$ hacking/test-module -m workshop-ansible/code/timetest.py
* including generated source, if any, saving to:
  /home/prkumar/.ansible_module_generated
  * this may offset any line numbers in tracebacks/debuggers!
    ***********************************
    RAW OUTPUT
    {"time": "2015-09-03 12:08:40.569710"}


  ***********************************
  PARSED OUTPUT
  {
      "time": "2015-09-03 12:08:40.569710"
  }
```

If you don't get any desired output then you might have to check your test module code again.

Read Input

We will pass a key value pair (time=<string>) to module and check if we are able to set time for a system.

Let's set time to "Oct 7 10:10"

- update timetest.py with latest changes (check in code directory)

```
$ hacking/test-module -m workshop-ansible/code/timetest_update.py -a "time=\"May 7␣
↪10:10\""
* including generated source, if any, saving to:
  /home/prkumar/.ansible_module_generated
  * this may offset any line numbers in tracebacks/debuggers!
    ***********************************
    RAW OUTPUT
    Thu May  7 10:10:00 IST 2015
    {"msg": "failed setting the time", "failed": true}

  date: cannot set date: Operation not permitted

  ***********************************
  INVALID OUTPUT FORMAT
  Thu May  7 10:10:00 IST 2015
  {"msg": "failed setting the time", "failed": true}
```

source

Example

CHAPTER 13

Ansible Vault

This feature of Ansible allows you to keep your sensitive data encrypted like passwords and keys.

- Ansible provide a command line tool *ansible-vault* for edit sensitive files.
- When you run a playbook then command line flag *-ask-vault-pass* or *-vault-password-file* can be used.
- Vault can encrypt any structured data file used by Ansible.

Create Encrypted File

```
ansible-vault create foo.yml
```

Edit Encrypted File

```
ansible-vault edit foo.yml
```

Rekeying Encrypted File

```
ansible-vault rekey foo.yml
```

View Content of Encrypted File

```
ansible-vault view foo.yml
```

Running a playbook with vault

```
ansible-playbook site.yml --ask-vault-pass
ansible-playbook site.yml --vault-password-file ~/.vault_pass.txt
ansible-playbook site.yml --vault-password-file ~/.vault_pass.py
```

CHAPTER 14

Further Reading

- Ansible Documentation: http://docs.ansible.com/ansible/
- Fedora's Ansible repo: https://infrastructure.fedoraproject.org/cgit/ansible.git
- Introduction to Ansible Video: https://www.youtube.com/watch?v=ak4yW6mF7Ns

CHAPTER 15

Indices and tables

- genindex
- modindex
- search