#ANSIBLEAUTOMATES

WRITING YOUR FIRST PLAYBOOK

Luca Villa
Senior Solution Architect

redhat | ANSIBLE

INSTALLING ANSIBLE

How-to

```
# ENABLE RPM REPO
subscription-manager repos --enable=rhel-7-server-ansible-2.7-rpms


# INSTALL ANSIBLE
yum install ansible
```

ANSIBLE

KEY COMPONENTS

Understanding Ansible terms

Modules	(Tools)
Inventory	
Tasks	
Roles	
Plays	
Playbook	(Plan)

MODULES

What is this?

Bits of code copied to the target system.
Executed to satisfy the task declaration.
Customizable.

The modules that ship with Ansible are mostly written in Python, but modules can be written in any language.

MODULES

Lot of choices / Ansible secret power…

- Cloud Modules
- Clustering Modules
- Commands Modules
- Crypto Modules
- Database Modules
- Files Modules
- Identity Modules
- Inventory Modules
- Messaging Modules
- Monitoring Modules
- Net Tools Modules
- Network Modules
- Notification Modules
- Packaging Modules
- Remote Management Modules
- Source Control Modules
- Storage Modules
- System Modules
- Utilities Modules
- Web Infrastructure Modules
- Windows Modules

MODULES

Commonly used

- yum
- copy
- file
- get_url
- git
- ping
- service
- command
- template
- uri
- user
- wait_for

IDEMPO-WHAT?

“Idempotence (/ˌaɪdɨmˈpoʊtəns/ EYE-dəm-POH-təns) is the property of certain operations in mathematics and computer science, that can be applied multiple times without changing the result beyond the initial application.”

“When carefully written, an Ansible playbook can be idempotent, in order to prevent unexpected side-effects on the managed systems.”

– Wikipedia

ANSIBLE INVENTORY & COMMANDS

INVENTORY

Use the default one /etc/ansible/hosts or create a host file

```
[joe@rhel ~]$ cat ./hosts
~~~
[all:vars]
ansible_ssh_user=service
[web]
web1 ansible_ssh_host=rhel01.mydomain.int
web2.mydomain.int
[database]
mysql.mydomain.int
~~~
[joe@rhel ~]$ ansible all -i ./hosts -m command -a "uptime"
```

AD-HOC COMMANDS

Run your first Ansible command...

```
              (targets)             (module)   (arguments)
     # ansible all -i ./hosts -m command -a "uptime"

     192.168.250.13 | success | rc=0 >>
      18:57:01 up 11:03,  1 user,  load average: 0.00, 0.01, 0.05

     192.168.250.11 | success | rc=0 >>
      18:57:02 up 11:03,  1 user,  load average: 0.00, 0.01, 0.05
```

ANSIBLE

AD-HOC COMMANDS

Other example of commands

```
# INSTALL HTTPD PACKAGE
ansible web -b -i ./hosts -m yum -a "name=httpd state=present"


# START AND ENABLE HTTPD SERVICE
ansible web -b -i ./hosts -m service -a "name=httpd enabled=yes state=started"
```

ANSIBLE PLAYBOOKS

YAML

1. Designed primarily for the representation of data structures
2. Easy to write, human readable format
3. Design objective : abandoning traditional enclosure syntax

YAML Validator : yamllint.com

PLAYBOOK EXAMPLE

```
- name: This is a Play
  hosts: web-servers
  remote_user: joe
  become: yes
  gather_facts: no
  vars:
    state: present

  tasks:
    - name: Install Apache
      yum: name=httpd state={{ state }}
      with_items:
        - httpd
        - php
```

PLAYS

Naming

```
- name: This is a Play
  hosts: web-servers
  remote_user: joe
  become: yes
  gather_facts: no
  vars:
    state: present

  tasks:
    - name: Install Apache
      yum: name=httpd state={{ state }}
      with_items:
        - httpd
        - php
```

PLAYS

Host selection

```
- name: This is a Play
  hosts: web-servers
  remote_user: joe
  become: yes
  gather_facts: no
  vars:
    state: present

  tasks:
    - name: Install Apache
      yum: name=httpd state={{ state }}
      with_items:
        - httpd
        - php
```

ANSIBLE

PLAYS

Arguments

```
- name: This is a Play
  hosts: web-servers
  remote_user: joe
  become: yes
  gather_facts: no
  vars:
    state: present

  tasks:
    - name: Install Apache
      yum: name=httpd state={{ state }}
      with_items:
        - httpd
        - php
```

ANSIBLE

PLAYS

Variables & tasks

```
- name: This is a Play
  hosts: web-servers
  remote_user: joe
  become: yes
  gather_facts: no
  vars:
    state: present

  tasks:
    - name: Install Apache
      yum: name=httpd state={{ state }}
      with_items:
        - httpd
        - php
```

**** When a variable is used as the first element to start a value, quotes are mandatory.

ANSIBLE

LOOPS

```
- name: This is a Play
  hosts: web-servers
  remote_user: joe
  become: yes
  gather_facts: no
  vars:
    state: present

  tasks:
    - name: Install Apache and PHP
      yum: name={{ item }} state={{ state }}
      with_items:
        - httpd
        - php
```

LOOPS

Many types of general and special purpose loops

- with_items
- with_nested
- with_dict
- with_fileglob
- with_together
- with_sequence
- until
- with_random_choice
- with_first_found
- with_indexed_items
- with_lines

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

ANSIBLE

HANDLERS

Only run if task has a “changed” status

```
- name: This is a Play
  hosts: web-servers

  tasks:
    - yum: name={{ item }} state=installed
      with_items:
        - httpd
        - memcached
      notify: Restart Apache

    - template: src=templates/web.conf.j2 dest=/etc/httpd/conf.d/web.conf
      notify: Restart Apache

  handlers:
    - name: Restart Apache
      service: name=httpd state=restarted
```

FACTS

Gathers facts about remote host

- **Ansible provides many facts about the system, automatically**
- **Provided by the setup module**
- **If facter (puppet) or ohai (chef) are installed, variables from these programs will also be snapshotted into the JSON file for usage in templating**
    - **These variables are prefixed with facter_ and ohai_ so it's easy to tell their source.**
- **Using the ansible facts and choosing to not install facter and ohai means you can avoid Ruby-dependencies on your remote systems**

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

FACTS SETTING

Setting facts in a play

```
- name: Example setting the Apache version
  shell: |
    httpd -v|grep version|awk '{print $3}'|cut -f2 -d'/'
  register: result

- set_fact:
    apache_version: "{{ result.stdout }}"
```

ANSIBLE

RUN AN ANSIBLE PLAYBOOK

```
[joe@rhel ansible]$ ansible-playbook play.yml -i hosts
```

Check mode aka “Dry run”:

```
[joe@rhel ansible]$ ansible-playbook play.yml -i hosts --check
```

TAGS

Example of tag usage

```
tasks:

  - yum: name={{ item }} state=installed
    with_items:
      - httpd
      - memcached
    tags:
      - packages

  - template: src=templates/src.j2 dest=/etc/foo.conf
    tags:
      - configuration
```

```
$ ansible-playbook example.yml --tags "configuration"

$ ansible-playbook example.yml --skip-tags "notification"
```

RESULTS

Registering task outputs for debugging or other purposes

```
# Example setting the Apache version
- shell: httpd -v|grep version|awk '{print $3}'|cut -f2 -d'/'
  register: result

- debug: var=result
```

CONDITIONAL TASKS

Only run this on Red Hat OS

```
- name: This is a Play
  hosts: web-servers
  remote_user: joe
  become: sudo

  tasks:
    - name: install Apache
      yum: name=httpd state=installed
      when: ansible_os_family == "RedHat"
```

BLOCKS

Apply a condition to multiple tasks at once

```
tasks:
- name: Install Apache
  block:
    - yum: name={{ item }} state=installed
      with_items:
        - httpd
        - memcached
    - template: src=templates/web.conf.j2 dest=/etc/httpd/conf.d/web.conf
    - service: name=bar state=started enabled=True
  when: ansible_distribution == 'RedHat'
```

ERRORS

Ignoring errors

By default, Ansible stop on errors. Add the ingore_error parameter to skip potential errors.

```
- name: ping host
  command: ping -c1 www.foobar.com
  ignore_errors: yes
```

ERRORS

Defining failure

You can apply a special type of conditional that if true will cause an error to be thrown.

```
    - name: this command prints FAILED when it fails
      command: /usr/bin/example-command -x -y -z
      register: command_result
      failed_when: "'FAILED' in command_result.stderr"
```

ANSIBLE VARIABLES & CONFIGURATION MANAGEMENT

VARIABLE PRECEDENCE

Ansible v2

Higher precedence

1. role defaults
2. inventory file or script group vars
3. inventory group_vars/all
4. playbook group_vars/all
5. inventory group_vars/*
6. playbook group_vars/*
7. inventory file or script host vars
8. inventory host_vars/*
9. playbook host_vars/*
10. host facts / cached set_facts
11. inventory host_vars/*
12. playbook host_vars/*

Higher precedence

13. host facts
14. play vars
15. play vars_prompt
16. play vars_files
17. role vars (defined in role/vars/main.yml)
18. block vars (only for tasks in block)
19. task vars (only for the task)
20. include_vars
21. set_facts / registered vars
22. role (and include_role) params
23. include params
24. extra vars (always win precedence)

ANSIBLE

MAGIC VARIABLES

Ansible creates and maintains information about its current state and other hosts through a series of "magic" variables.

★ hostvars[inventory_hostname]

★ hostvars[<any_hostname>]

```
{{ hostvars['test.example.com']['ansible_distribution'] }}
```

★ group_names

is a list (array) of all the groups the current host is in

★ groups

is a list of all the groups (and hosts) in the inventory.

MAGIC VARIABLES

Using debug mode to view content

```
- name: debug
  hosts: all

  tasks:
    - name: Show hostvars[inventory_hostname]
      debug: var=hostvars[inventory_hostname]

    - name: Show ansible_ssh_host variable in hostvars
      debug: var=hostvars[inventory_hostname].ansible_ssh_host

    - name: Show group_names
      debug: var=group_names

    - name: Show groups
      debug: var=groups
```

```
ansible-playbook -i ../hosts --limit <hostname> debug.yml
```

ANSIBLE

LINEINFILE

Add, remove or update a particular line

```
- lineinfile:
    dest: /etc/selinux/config
    regexp: '^SELINUX='
    line: 'SELINUX=enforcing'

- lineinfile:
    dest: /etc/httpd/conf/httpd.conf
    regexp: '^Listen '
    insertafter: '^#Listen '
    line: 'Listen 8080'
```

Great example here :
https://relativkreativ.at/articles/how-to-use-ansibles-lineinfile-module-in-a-bulletproof-way

Note : Using template or a dedicated module is more powerful

TEMPLATE MODULE

Using Jinja2

Templates allow you to create dynamic configuration files (and not only) using variables.

```
- name: Put configuration in place
  template:
    src=/mytemplates/foo.j2
    dest=/etc/file.conf
    owner=bin
    group=wheel
    mode=0644
```

Documentation:
http://docs.ansible.com/ansible/template_module.html

A **Shinto shrine** (神社 *jinja*, archaic: *shinsha*, meaning: "place of the god(s)") is a structure whose main purpose is to house ("enshrine") one or more *kami* (spirits or phenomena).

Jinja2 is a full featured template engine for Python. It has full unicode support, an optional integrated sandboxed execution environment, widely used and BSD licensed.

http://jinja.pocoo.org

ANSIBLE

JINJA2

Delimiters

Ansible uses Jinja2.
Highly recommend reading about Jinja2 to understand how templates are built.

```
    {{variable }}
    {# comment #}
    {% for server in groups.webservers %}
```

JINJA2

Loops

```
{% for server in groups.web %}
{{ server }} {{ hostvars[server].ansible_default_ipv4.address }}
{% endfor %}
```

```
web1 10.0.1.1
web2 10.0.1.2
web3 10.0.1.3
```

JINJA2

Conditional and testing

```
{% if ansible_processor_cores >= 2 %}
-smp enable
{% else %}
-smp disable
{% endif %}
```

```
{% if variable is defined %}
{% if variable is none %}
{% if variable is even %}
{% if variable is string %}
{% if variable is sequence %}
```

JINJA2

Variable filters

```
{% set my_var='this-is-a-test' %}
{{ my_var | replace('-', '_') }}
```

```
this_is_a_test
```

JINJA2

Variable filters

```
{% set servers = "server1,server2,server3" %}
{% for server in servers.split(",") %}
{{ server }}
{% endfor %}
```

```
server1
server2
server3
```

ANSIBLE

ANSIBLE ROLES

ROLES

A redistributable and reusable collection of:

- tasks / handlers
- files
- scripts
- templates
- variables

ROLES

A redistributable and reusable collection of:

```
roles
└── myapp
    ├── defaults
    ├── files
    ├── handlers
    ├── meta
    ├── tasks
    ├── templates
    └── vars
```

DIRECTORY STRUCTURE

EASY SHARING !!!

ANSIBLE

ROLES

Create folder structure automatically

```
ansible-galaxy init <role_name>
```

ROLES

Using Roles (the classic way):

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

ROLES

With parameters:

```
---
- hosts: webservers
  roles:
    - common
    - { role: myapp, dir: '/opt/a',  port: 5000 }
    - { role: myapp, dir: '/opt/b',  port: 5001 }
```

ROLES

With conditional:

```
---
- hosts: webservers
  roles:
    - { role: foo, when: "ansible_os_family == 'RedHat'" }
```

ROLES

Pre and Post - rolling upgrade example

```
- hosts: webservers
  serial: 1

  pre_tasks:
    - command:lb_rm.sh {{ inventory_hostname }}
      delegate_to: lb
    - command: mon_rm.sh {{ inventory_hostname }}
      delegate_to: nagios

  roles:
    - myapp

  post_tasks:
    - command: mon_add.sh {{ inventory_hostname }}
      delegate_to: nagios
    - command: lb_add.sh {{ inventory_hostname }}
      delegate_to: lb
```

ROLES

Playbook examples (using the new syntax - v. 2.4+):

```
---
- hosts: webservers

  tasks:
    - import_role:   << static inclusion
        name: common
    - include_role:  << dynamic inclusion
        name: my_app
      vars:
        dir: '/opt/a'
        port: 5000
```

GALAXY

15,000 ROLES AT YOUR DISPOSAL

Reusable Roles and Container Apps that allow you to do more, faster

Built into the Ansible CLI and Tower

galaxy.ansible.com

ANSIBLE

GETTING STARTED

Have you used Ansible already?
Try Tower for free: **ansible.com/tower-trial**

Would you like to learn Ansible?
It's easy to get started: **ansible.com/get-started**

Want to learn more?
Videos, webinars, case studies, whitepapers: **ansible.com/resources**

ANSIBLE

AUTOMATION EVERYWHERE

redhat. | ANSIBLE