Ansible Basic

An Ansible Training Course

DevOps Artisan

4. Basic Playbooks

DevOps Artisan

Topics covered

- YAML overview
- Modules, Tasks, Plays, Playbooks
- General Playbook Structure
- Commonly used Modules
- Task Results (OK vs changed vs failed)
- Validating the Result
- Writing Idempotent Tasks

DevOps Artisan

Why Playbooks?

- Ad-hoc commands can be used to run one or a few tasks
- Ad-hoc commands are convenient to test, or when a complete managed infrastructure hasn't been set up yet.
- Ansible **Playbooks** are used to **run multiple tasks** against **managed hosts** in a scripted way
- In Playbooks, one or multiple plays are started
  - Each play runs one or more tasks
  - In these tasks, different modules are used to perform the actual work

- **Playbooks** are **written** in **YAML**, and have the **.yml** or **.yaml** extension

YAML Overview

- YAML is Yet Another Markup Language according to some.
- According to others it stands for YAML Ain’t Markup Language
- Anyway, it’s an easy-to-read format to structure tasks/items that need to be created

- In YAML files, items are using indentation whit white spaces to indicate the structure of data
- Data elements at the same level should have the same indentation
- Child items are indented more than the parent items

Create a Basic Playbook

- A simple playbook which contains just one play (and particularly this play will contain just one task) and performs the ping command to all hosts in the inventory:

```
---
- name: Ping all hosts
  hosts: all
  tasks:
  - name: Ping task
    ping:
```

Run a Playbook

- Use **`ansible-playbook <playbook.yml>`** to run the playbook

- Notice that a successful run requires the inventory and become parameters to be set correctly, and also requires access to an inventory file
- The output of the **ansible-playbook** command will show what exactly has happened
- Playbooks in general are idempotent, which means that running the same playbook again should lead to the same result
- Notice there is no easy way to undo changes mage by a playbook

Run a Playbook - output

```
student:~$ ansible-playbook playbook.yml

PLAY [Ping all hosts]
*********************************************************************************************

TASK [Gathering Facts]
*********************************************************************************************
ok: [ubuntu]
ok: [hivemaster]
ok: [centos]

TASK [Ping task]
*********************************************************************************************
ok: [ubuntu]
ok: [hivemaster]
ok: [centos]

PLAY RECAP
*********************************************************************************************
centos                     : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
hivemaster                 : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
ubuntu                     : ok=2    changed=0    unreachable=0    failed=0    skipped=0    rescued=0
ignored=0
```

DevOps Artisan

Understanding Modules

- Ansible comes with lots of modules that allow you to perform specific tasks on managed hosts.
- When using Ansible you'll always use modules to tell Ansible what you want to do, in ad-hoc commands as well in playbooks.
- Many modules are provided with Ansible, if required you can develop your own modules.
- Use **ansible-doc -l** for a list of modules currently available.
- All modules work with arguments, **ansible-doc** will show which arguments are available and which are required.

Understanding Modules

```
[student@workstation modules]$ ansible-doc -l
a10_server                     Manage A10 Networks AX/SoftAX/Thunder/vThunder devices
a10_service_group              Manage A10 Networks devices' service groups
a10_virtual_server             Manage A10 Networks devices' virtual servers
acl                            Sets and retrieves file ACL information.
add_host                       add a host (and alternatively a group) to the ansible-
playbook in-memory inventory
airbrake_deployment            Notify airbrake about app deployments
alternatives                   Manages alternative programs for common commands
apache2_module                 enables/disables a module of the Apache2 webserver
apk                            Manages apk packages
apt                            Manages apt-packages
...output omitted...
```

Understanding Modules

```
[student@workstation modules]$ ansible-doc yum
> YUM

        Installs, upgrade, removes, and lists packages and groups with the`yum' package manager.

Options (= is mandatory):

- conf_file
        The remote yum configuration file to use for the transaction.
        [Default: None]

- disable_gpg_check
        Whether to disable the GPG checking of signatures of packages
        being installed. Has an effect only if state is `present' or
        `latest'. (Choices: yes, no) [Default: no]

...output omitted...

EXAMPLES:
- name: install the latest version of Apache
  yum: name=httpd state=latest

- name: remove the Apache package
  yum: name=httpd state=absent
...output omitted...
```

Tasks

- Modules (with various settings) that are executed on remote hosts

- Configuration is declarative
  - We are telling the system what the desired end state needs to be
  - If the remote host configuration is already correct, nothing gets changed

- Example:

```
- name: Task to start the apache service
  service:
    name: httpd
    state: started
```

Handlers

- Used to only execute some actions when they are needed (due to a change)
- These “notify” actions are triggered at the end of each block of tasks in a play, and will only be triggered once even if notified by multiple different tasks.
  - Ex.: Multiple resources may indicate that apache needs to be restarted because they have changed a config file, but apache will only be restarted once.
- Handlers are lists of tasks that are referenced by a globally unique name and are notified as needed.
- If nothing notifies a handler, it will not run.

Handlers

```yaml
- name: template configuration file
  template:
    src: template.j2
    dest: /etc/foo.conf
  notify:
     - restart memcached
     - restart apache
```

```yaml
handlers:
    - name: restart memcached
      service:
        name: memcached
        state: restarted
    - name: restart apache
      service:
        name: apache
        state: restarted
```

Verifying Playbook Syntax

- **`ansible-playbook –syntax-check <playbook.yml>`** will perform a syntax check

- Use **-v[vvv]** to increase output verbosity
  - **-v** will show task results
  - **-vv** will show task results and task configuration
  - **-vvv** also shows information about connections to managed hosts
  - **-vvvv** adds information about plug-ins, users used to run scripts and names of scripts that are executed
- Use the **-C** option to perform a dry run

Run a Playbook – with -vv output

```
student:~$ ansible-playbook playbook.yml -vv
ansible-playbook 2.9.1
  config file = /etc/ansible/ansible.cfg
  configured module search path = [u'/home/student/.ansible/plugins/modules', u'/usr/share/ansible/plugins/modules']
  ansible python module location = /usr/lib/python2.7/dist-packages/ansible
  executable location = /usr/bin/ansible-playbook
  python version = 2.7.15+ (default, Feb  7 2022, 17:39:04) [GCC 7.4.0]
Using /etc/ansible/ansible.cfg as config file

PLAYBOOK: playbook.yml
*********************************************************************************************
1 plays in playbook.yml

PLAY [Ping all hosts]
*********************************************************************************************

TASK [Gathering Facts]
*********************************************************************************************
task path: /home/student/playbook.yml:2
ok: [ubuntu]
META: ran handlers

TASK [Ping task] ********************************************************************************
task path: /home/student/playbook.yml:5
ok: [ubuntu] => {"changed": false, "ping": "pong"}
META: ran handlers
[---]
```

DevOps Artisan

Understanding Plays

- A play is a series of tasks that are executed against selected hosts from the inventory, using specific credentials.
- Using multiple plays allows running tasks on different hosts, using different credentials from the same playbook.

- Within a play definition, escalation parameters can be defined:
  - **remote_user**: the name of the remote user
  - **become**: to enable or disable privilege escalation
  - **become_method**: to allow using an alternative escalation solution
  - **become_user**: the target user used for privilege escalation

Understanding Plays

```
---
- name: Ping all hosts
  hosts: all
  tasks:
  - name: Ping task
    ping:

- name: Deploy mongodb for dbservers group
  hosts: dbservers
  become: true
  tasks:
  - name: Install mongo
    package:
      name: mongodb
      state: latest
    notify: restart mongodb

  handlers:
  - name: restart mobgodb
    service:
      name: mongodb
      status: restarted
      enabled: yes
```

DevOps Artisan

Commonly used Modules: Package management

- There is a module for most popular package managers, such as YUM and APT, to enable you to install any package on a system.

- Functionality depends entirely on the package manager, but usually these modules can install, upgrade, downgrade, remove, and list packages

```
- name: Install a list of packages
  yum:
     name:
      - nginx
       - postgresql
       - postgresql-server
     state: present
```

Commonly used Modules: Service

- After installing a package, you need a module to start it.

- The service module enables you to start, stop, and reload installed packages.

```
- name: Start service foo, based on running process /usr/bin/foo
  service:
    name: foo
    pattern: /usr/bin/foo
    state: started
```

Commonly used Modules: Copy

- The **copy module** copies a file from the local or remote machine to a location on the remote machine.

```
- name: Copy a new "ntp.conf file into place, backing up the original if it differs
from the copied version
  copy:
    src: /mine/ntp.conf
    dest: /etc/ntp.conf
    owner: root
    group: root
    mode: '0644'
    backup: yes
```

DevOps Artisan

Commonly used Modules: Debug

- The **debug module** prints statements during execution and can be useful for debugging variables or expressions without having to halt the playbook.

```
- name: Display all variables/facts known for a host
  debug:
    var: hostvars[inventory_hostname]
    verbosity: 4
```

Commonly used Modules: File

- The **file module** manages the file and its properties.
  - It sets attributes of files, symlinks, or directories.
  - It also removes files, symlinks, or directories.

```
- name: Change file ownership, group and permissions
  file:
    path: /etc/foo.conf
    owner: foo
    group: foo
    mode: '0644'
```

Commonly used Modules: Lineinfile

- The **lineinfile module** manages lines in a text file.
  - It ensures a particular line is in a file or replaces an existing line using a back-referenced regular expression.
  - It's primarily useful when you want to change just a single line in a file.

```
- name: Ensure SELinux is set to enforcing mode
  lineinfile:
    path: /etc/selinux/config
    regexp: '^SELINUX='
    line: SELINUX=enforcing
```

Commonly used Modules: Git

- The **git module** manages git checkouts of repositories to deploy files or software.

```
# Example Create git archive from repo
- git:
    repo: https://github.com/ansible/ansible-examples.git
    dest: /src/ansible-examples
    archive: /tmp/ansible-examples.zip
```

Commonly used Modules: Cli_command

- The **cli_command module**, provides a platform-agnostic way of pushing text-based configurations to network devices over the **network_cli connection** plugin.

```
- name: commit with comment
  cli_config:
    config: set system host-name foo
    commit_comment: this is a test
```

Commonly used Modules: Archive

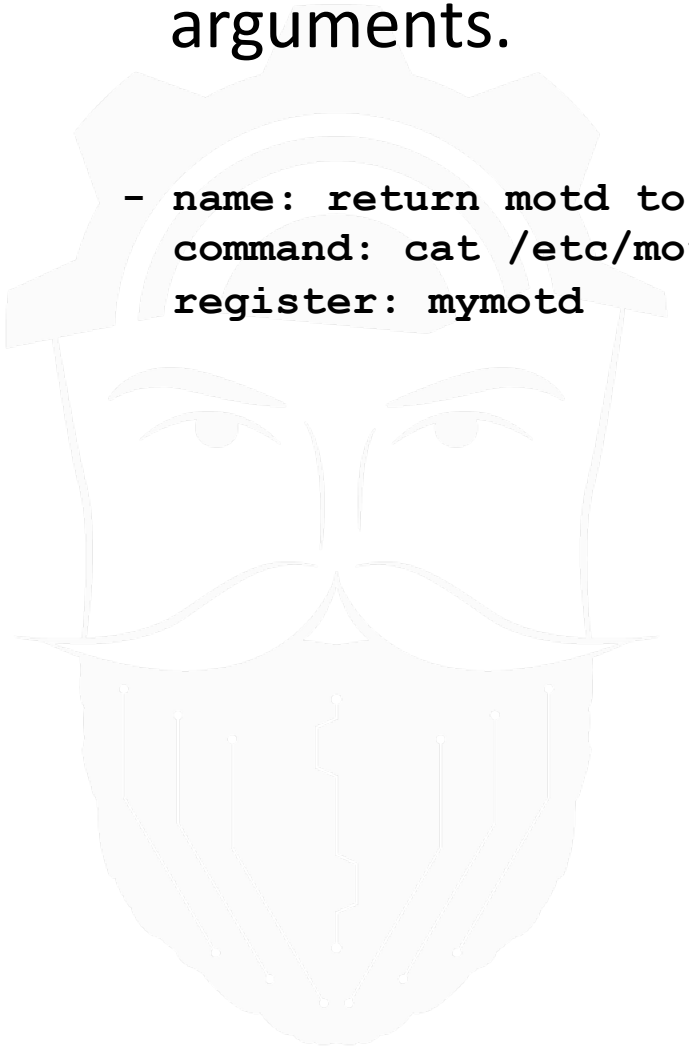
- The **archive module** creates a compressed archive of one or more files.
- By default, it assumes the compression source exists on the target.

```
- name: Compress directory /path/to/foo/ into /path/to/foo.tgz
  archive:
    path: /path/to/foo
    dest: /path/to/foo.tgz
```

Commonly used Modules: Command

- One of the most basic but useful modules, the **command module** takes the command name followed by a list of space-delimited arguments.

```
- name: return motd to registered var
  command: cat /etc/motd
  register: mymotd
```



DevOps Artisan

Commonly used Modules: User

- The module manages users accounts with their attributes.

- This is handy do to the fact that the user properties and attributes can all be configured from this Ansible module.

```
# Add the user 'student' with a specific uid and a primary group of 'admin'
- user:
    name: student
    comment: "Example User"
    uid: 1040
    group: admin

# Remove the user 'student'
- user:
    name: student
    state: absent
    remove: yes
```

OK vs Changed vs Failed

- As you already seen, every task of a play returns a status.

- This status is aiming to give the user a feedback about whether the task succeeded or not for a certain host

```
PLAY [playbook] ****************************************************************
... Output omitted ...
TASK: [Install a service] ******************************************************
ok: [demoservera]
ok: [demoserverb]

PLAY RECAP *********************************************************************
demoservera :          ok=2    changed=0       unreachable=0   failed=0
demoserverb :          ok=2    changed=0       unreachable=0   failed=0
```

DevOps Artisan

Writing Idempotent Tasks

- When possible, try to avoid the **command**, **shell**, and **raw** modules in playbooks, as simple as they may seem to use.

- Because these take arbitrary commands, it is very easy to write non-idempotent playbooks with these modules.

DevOps Artisan

Writing Idempotent Tasks

- When possible, try to avoid the **command**, **shell**, and **raw** modules in playbooks, as simple as they may seem to use.

- Because these take arbitrary commands, it is very easy to write non-idempotent playbooks with these modules.

- In order to achieve idempotency an useful feature is “creates” (used in **command** and **shell** modules).
- **Creates** (there is also **removes**) won’t create the file if it already exists

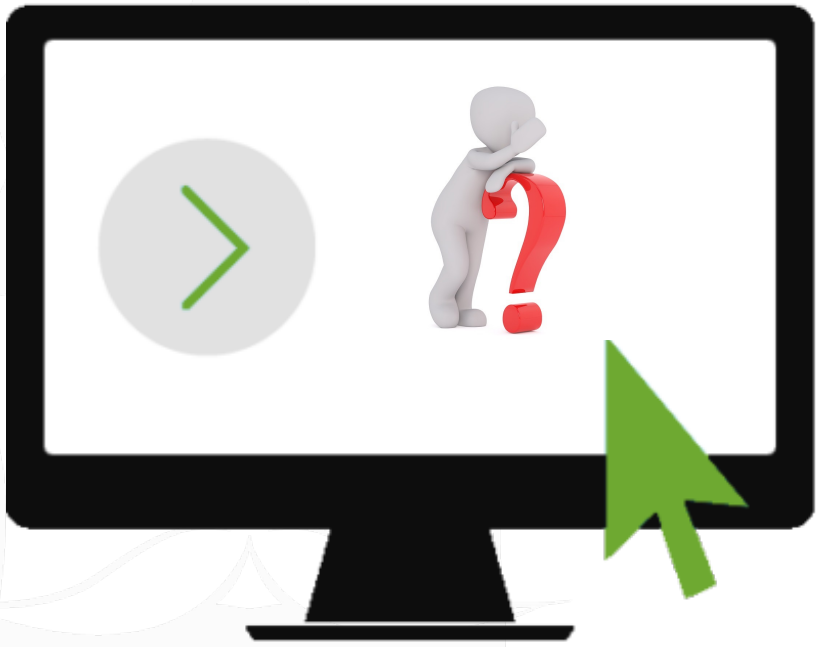
Writing Idempotent Tasks

```
---
- name: Command modules Playbook
  hosts: all
  become: yes
  tasks:
  - name: Raw module
    raw: cat /etc/hosts
    register: raw_ouput

  - name: Shell module
    shell: ls -l /var/log | grep log > /tmp/tmp.log
    args:
      creates: /tmp/tmp.log

  - name: Command module
    command: cat /etc/shadow
    register: cmd_output
```

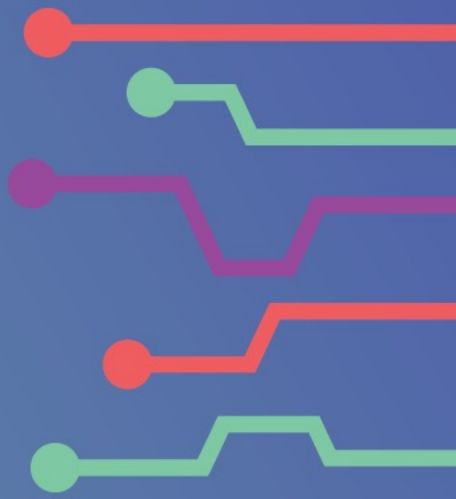
DevOps Artisan



DevOps Artisan

Lab 4: ***Basic Playbooks***

DevOps Artisan



More practice, less theory

askformore@devopsartisan.com

DevOps Artisan