neueda

futureproof your workforce

MODULE 3: ANSIBLE PLAYBOOKS

NEUEDA.COM

Poll Question

What sort of scripting are you using in your work?

A. Linux shell scripts

B. Python

C. Perl

D. Something else

E. I don’t use scripts

neueda

futureproof your workforce

Module Overview

- Ansible Playbooks
- Variables
- Using Operators in Playbooks
- Playbook Flow Control

ANSIBLE PLAYBOOKS

neueda
future proof your workforce

4

Ansible Playbooks

- Ad-hoc commands are powerful and flexible
- We want to collect commands into scripts
  - Repeatable
  - Can check into version control
- In Ansible these scripts are call **Playbooks**.
  - Written in YAML
  - Easy to read and edit
- YAML Syntax

Example: Install Apache

```
# Install Apache
yum install --quiet -y httpd httpd-devel
# Copy configuration files
cp httpd.conf /etc/httpd/conf/httpd.conf
cp httpd-vhosts.conf \
 /etc/httpd/conf/httpd-vhosts.conf
# start Apache and set to run at boot
service httpd start
chkconfig httpd on
```

Shell script

Ansible playbook

```
---
- hosts: all
  tasks:
    - name: Install Apache
      command: yum install --quiet -y httpd httpd-devel
    - name: Copy configuration files
      command: >
        cp httpd.conf /etc/httpd/conf/httpd.conf
    - command: >
        cp httpd-vhosts.conf /etc/httpd/conf/httpd-
vhosts.conf
    - name: Start Apache and set to run at boot
      command: service httpd start
    - command: chkconfig httpd on
```

Example: Install Apache Improved!

```yaml
---
- hosts: all
  become: yes

  tasks:
    - name: Install Apache
      yum:
        name:
          - httpd
          - httpd-devel
        state: present

    - name: Copy configuration files
      copy:
        src: "{{ item.src }}"
        dest: "{{ item.dest }}"
        owner: root
        group: root
        mode: 0644
      with_items:
        - src: httpd.conf
          dest: /etc/httpd/conf/httpd.conf
        - src: httpd-vhosts.conf
          dest: /etc/httpd/conf/httpd-vhosts.conf

    - name: Start Apache and set to run at boot
      service:
        name: httpd
        state: started
        enabled: yes
```

Patterns for Targeting Inventory Hosts

- Syntax for determining hosts is quite flexible

Description	Pattern(s)	Targets
All hosts	all (or *)	
One host	host1	
Multiple hosts	host1:host2 (or host1,host2)	
One group	webservers	
Multiple groups	webservers:dbservers	all hosts in webservers plus all hosts in dbservers
Excluding groups	webservers:!atlanta	all hosts in webservers except those in atlanta
Intersection of groups	webservers:&staging	any hosts in webservers that are also in staging

Advanced Targeting Patterns

- Basic patterns can be combined (using a comma or a colon as a separator)

```
webservers:dbservers:&staging:!phoenix
```

- Wildcards are OK

```
192.0.*
*.example.com
*.com
one*.com:dbservers
```

- Also regexes

```
~(web|db).*\.example\.com
```

- Full info in the [documentation]

Multiple Plays

- Multiple plays in a single playbook
- Each play addresses a different inventory group

```
---
- name: Install wget and vim
  hosts: app
  become: yes
  tasks:
  - yum: name=wget state=latest
  - yum: name=vim state=latest

- name: Install Apache
  hosts: web
  become: yes
  tasks:
  - yum: name=httpd state=latest
```

Playbooks can be run as Scripts

- The --- first line can be replaced with the ansible-playbook executable to run a playbook as a script:

```
$ chmod u+x this-playbook.yml
$ ./this-playbook.yml
```

```
#!/usr/bin/ansible-playbook
- name: Install wget and vim
  hosts: app
  become: yes
  tasks:
  - yum: name=wget state=latest
  - yum: name=vim state=latest

- name: Install Apache
  hosts: web
  become: yes
  tasks:
  - yum: name=httpd state=latest
```

Other Playbook Features

- Add debug comments using the debug task

```yaml
---
- hosts: web
  tasks:
  - debug:
      msg:
      - "This is first line"
      - "This is second line"
      - "This is third line"
```

- Check your syntax using the `--syntax-check` command line option

```
ansible-playbook --syntax-check debug.yaml
```

Lab-2

- Follow the instructions for the second lab.

VARIABLES

neueda
future proof your workforce

14

Using Variables

- We have already seen a variable used in lab 2
- **Simple variables**
  - defined by `variable_name: variable_value` expressions
  - in the `vars:` section at the top level of a play
  - Referenced using Jinja2 syntax {{ variable_name }}
  - YAML requirement: the whole Jinja2 expression has to be quoted to be valid

```
- hosts: app_servers
  vars:
      app_path: "{{ base_path }}/22"
```

Variable Types

- Boolean
  - You can use almost any standard Boolean values: `True/False`, `true/false`, `yes/no`, `1/0` etc.
- List Variables:

```yaml
region:
  - northeast
  - southeast
  - midwest
```

```yaml
region: "{{ region[0] }}"
```

- Key:value dictionaries:

```yaml
foo:
  field1: one
  field2: two
```

```
foo['field1']
foo.field1
```

Registering Variables

- Remember that a playbook is run top-down in order.
- This means the output of one play could be used for the input of another.
- You can do this using the ``register`` task keyword:

```
- hosts: web_servers

  tasks:

    - name: Run a shell command and register its output as a variable
      ansible.builtin.shell: /usr/bin/foo
      register: foo_result
      ignore_errors: true

    - name: Run a shell command using output of the previous task
      ansible.builtin.shell: /usr/bin/bar
      when: foo_result.rc == 5
```

More about Registered Variables

- They can be simple variables, list variables, dictionary variables or complex nested data structures.
- If you check the documentation for the module you are using in a task, it will include a RETURN section that tells you the return values for that module.
- They are stored in memory and only valid for the current playbook run.

Ansible Facts

- A whole lot of info about your remote systems are discovered in a playbook run.
  - Operating systems
  - IP addresses
  - etc
- These are stored in a variable called `ansible_facts`
- You can see all of them by adding this task to a play

```
- name: Print all available facts
  ansible.builtin.debug:
    var: ansible_facts
```

Referencing nested variables

- Some registered variables (and `ansible_facts`) are nested data structures.
- They are referenced using either bracket or dot notation

```
- name: Print IPv4 address two ways
  ansible.builtin.debug:
    msg:
    - '{{ ansible_facts["eth0"]["ipv4"]["address"] }}'
    - '{{ ansible_facts.eth0.ipv4.address }}'
```

Where to Set Variables

- In your inventory
- In a playbook play:
- In included files:

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

```
---
somevar: somevalue
password: magic
```

/vars/external_vars.yml

```
---
- hosts: all
  vars:
    favcolor: blue
  vars_files:
    - /vars/external_vars.yml
```

- At runtime using `--extra-vars`:

```
ansible-playbook release.yml --extra-vars "version=1.23.45 other_variable=foo"
ansible-playbook release.yml --extra-vars '{"version":"1.23.45","other_variable":"foo"}'
ansible-playbook arcade.yml --extra-vars '{"pacman":"mrs","ghosts":["inky","pinky","clyde","sue"]}'
ansible-playbook release.yml --extra-vars "@some_file.json"
```

USING OPERATORS IN PLAYBOOKS

neueda
future proof your workforce

Using Operators in Playbooks

- Ansible provides extensive support for operators in playbooks
- Operators and flow control (next section) can supercharge your playbooks
- Operators make most sense when used with variables
- **Arithmetic** operators act on variables (or constants) within the {{}} syntax
- **Comparison** operators (`==`, `!=`, `<`, `>`, `>=`, `<=`) have exactly the same syntax and resolve to `True` or `False`

```yaml
---
 - name: Perform arithmetic operations
   hosts: localhost
   gather_facts: false
   vars:
     num1: 30
     num2: 15
   tasks:
     - name: "Calculations"
       debug:
         msg:
          - "Multiply num1 and num2: {{ num1*num2 }}"
          - "Add num1 and num2: {{ num1+num2 }}"
          - "Subtract num2 from num1: {{ num1-num2 }}"
          - "Divide num1 by num2: {{ num1/num2 }}"
          - "Check remainder: {{ num1%num2 }}"
```

Test Operators

- Test operators are more sophisticated Boolean operators using Jinja tests.
- There are a lot of these, just a sample:

defined	Returns true if the variable is defined
undefined	The opposite of defined
none	Returns true if the variable is defined, but the value is none
even	Returns true if the number is divisible by 2
odd	Returns true if the number is not divisible by 2
is	Returns true is a variable is something
is not	Returns true is a variable is not something
in	Returns true is a variable is in a list

Logical Operators

- Jinja offers the logical operators `{% if %}`, `{% elif %}` `{% else %}` and `{% endif %}`
- These can allow us some more flexibility when implementing conditional logic
- We can also use `and` & `or` in the Jinja brackets to combine logical statements

```
---
 - name: Logical Operator
   hosts: localhost
   gather_facts: false
   vars:
     hello: true
     say_something: "{% if hello == true %} Hello Jinja {% else %} Goodbye Ansible {% endif %}"
   tasks:
     - debug:
         msg:
          - "{{ say_something }}"
```

PLAYBOOK FLOW CONTROL

neueda
futureproof your workforce

Ansible Conditional Statement

- Sometimes we need to execute code conditionally based on a variable value or ansible fact.
- The when conditional statement enables this.
- In practice you can execute a task based on the OS of a host (for example).

```yaml
---
 - name: when conditional operator
   hosts: localhost
   gather_facts: false
   vars:
     x: 10
     y: 15
     list: [10,20,30]
   tasks:
     - debug:
         msg:
          - "The value of x is {{ x }} and the value of y is {{ y }}"
          - "Our list contains: {{ list }}"
       when: x == y
     - debug:
         msg: "x is present in the list"
       when: x in list
```

Ansible Loop

- Ansible offers the `loop` keyword to execute a task multiple times over a list.
- Useful for things like
  - Creating multiple users
  - Repeating a polling step
  - Changing ownership on a list of files.
- You can loop over a list, list of hashes, or a dictionary
- Loops are very flexible, much more info is in the docs

Loop over a List

```yaml
---
- name: Looping over a list
  hosts: localhost
  gather_facts: false
  tasks:
    - name: Echo the value
      command: echo "{{ item }}"
      loop:
        - Huey
        - Dewey
        - Louie
        - Uncle Scrooge
```

- You can also register the output:

```yaml
      register: loopresult

    - name: print the results from loop
      debug:
        var: loopresult
```

Loop over a list of hashes

```yaml
---
- name: Looping over a list of hashes
  hosts: localhost
  gather_facts: false
  tasks:
    - name: Iterate over hashes
      debug:
        msg:
        - "Hello '{{ item.fname }}', nice to meet you"
        - "Your last name as per our record is '{{ item.lname }}'"
        - "Your planet of residence is '{{ item.location }}'"

      loop:
        - { fname: 'Luke', lname: 'Skywalker', location: 'Tatooine' }
        - { fname: 'James', lname: 'Kirk', location: 'Earth' }
        - { fname: 'Yoda', lname: '', location: 'Dagoba System' }
        - { fname: 'Mr', lname: 'Spock', location: 'Vulcan' }
```

Looping over a Dictionary

```yaml
---
- name: Looping over a dictionary
  hosts: localhost
  gather_facts: false
  tasks:
    - name: Iterate over dictionary
      debug:
        msg:
        - "The {{ item.key }} of your car is {{ item.value }}"

      loop: "{{ my_car | dict2items }}"
      vars:
        my_car:
          Color: Blue
          Model: Corvette
          Transmission: Manual
          Price: $20,000
```

REVIEW

neueda
futureproof your workforce

32

Module Review

In this module you learned about:

- ✓ Ansible Playbooks
- ✓ Variables
- ✓ Using Operators in Playbooks
- ✓ Playbook Flow Control

Next we will do a short quiz

- Knowledge Check

More Info

- Ansible playbooks — Ansible Documentation
- Using Variables — Ansible Documentation
- Jinja Test Conditions — Jinja Documentation

KNOWLEDGE CHECK

neueda
futureproof your workforce

Knowledge check question 1

What is the declaration the yum module uses to uninstall a package?

Choice	Response
A	uninstall: true
B	state: uinstalled
C	state: absent

Knowledge check question 1

What is the declaration the yum module uses to uninstall a package?

Choice	Response
A	uninstall: true
B	state: uinstalled
C	state: absent

Knowledge check question 2

True or false: can a playbook be run as a script?

Choice	Response
A	True
B	False

Knowledge check question 2

True or false: can a playbook be run as a script?

Choice	Response
A	True
B	False

Knowledge check question 3

Which variable contains information on the host OS?

Choice	Response
A	ansible_hosts
B	ansible_facts
C	ansible_inventory
D	ansible_os

Knowledge check question 3

Which variable contains information on the host OS?

Choice	Response
A	ansible_hosts
B	ansible_facts
C	ansible_inventory
D	ansible_os

Knowledge check question 4

Which command line argument includes external variables into a playbook?

Choice	Response
A	--extra-vars
B	--vars
C	--external-vars
D	--var-file

Knowledge check question 4

Which command line argument includes external variables into a playbook?

Choice	Response
A	--extra-vars
B	--vars
C	--external-vars
D	--var-file