Package Management Using Ansible Playbooks

Objectives:

After completing this lab you should be able to:
1. Create playbooks.
2. Use the “When” conditional.
3. Target specific hosts with your plays.
4. use Tags.

Required resources:

- 2 PCs with virtualBox [PC1,PC2].
- Internet connection.

Setting up the Work Environment

We will be using 6 VMs as listed in the table bellow:

PC	VM	OS	Hostname	IP	NIC	RAM
PC1	Workstation1	Ubuntu Desktop	Workstation1	DHCP	Bridged	4 GB
	SRVR01	Ubuntu Server	SRVR01	DHCP	Bridged	1 GB
	SRVR02	Ubuntu Server	SRVR02	DHCP	Bridged	1 GB
PC2	Workstation1	Ubuntu Desktop	Workstation2	DHCP	Bridged	4 GB
	SRVR03	Ubuntu Server	SRVR03	DHCP	Bridged	1 GB
	SRVR04	Almalinux	SRVR04	DHCP	Bridged	2 GB

Before you begin:

1. Create the Almalinux VM on PC2 using the template.
2. You need to setup openSSH on the AlmaLinux VM the same way we did in the Automation lab. refer to that lab if you need specific commands.

3. Remove apache server from on all servers.

Playbooks:

A task we want to perform on our hosts using ansible is called a play. A playbook contains one or more tasks we want to execute. playbooks are written using the yaml language which is a human readable data-serialization language. Yaml is usually used for configuration files.

The real strength of ansible comes from playbooks. when we write our playbooks we define the state we want our servers to be in and the commands we want ansible to perform to bring our servers to that state.

Introduction to Playbooks

In the nislab directory on workstation1 create a file called install_apache.yml with the following contents:

```
---

- hosts: all
  become: true
  tasks:

  - name: install apache2 package
    apt:
      name: apache2
```

spaces are very important here. after the --- at the top leave 2 lines then on the 3rd line type one single hyphen (-) then space then hosts: then space then all. after that go down one line then leave two spaces and type become: then space then true.

--- : the start of the file.

- hosts: all : which hosts will be affected by the plays.

become: true: for sudo

tasks: : means after this will be the list of plays to be executed.

- name: install apache2 package : the name (description) of this play

apt: : the module to be executed , here it’s apt

name: apache2 : the name of the package we want to install.

to run the yml file use the following command:

```
ansible-playbook --ask-become-pass install_apache.yml
```

The output will have a few important info when running the play on each host:

ok: this lists the number of plays that ran without problems on the host.

changed: the number of plays that made changes when ran on the host.

unreachable: if the host is offline.

failed: number of failed plays on this host.

skipped: number of plays that were skipped because the host did not meet the conditions for running this play.

rescued: number of plays that ran as a rescue because other plays failed to run.

ignored: number of ignored plays.

The previous playbook could either succeed or fail depending on the repository index status. On Linux systems as you already know we need to update the repository index before trying to install packages because we might get an error that the package was not found. this happens because URLs keep changing all the time and we need the new links for the packages to be downloaded.

we do that by the command apt update. now let’s modify our playbook to include the ansible task equivalent to that:

```
---

- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes

  - name: install apache2 package
    apt:
      name: apache2
```

run the playbook and notice the tasks that are executed successfully. ok=3 which are the gathering fact, update repository, and install apache2 package tasks. the changed will be only 2 since gathering facts makes no changes. The update repository task will always make a change whenever we run the playbook. let’s make another change to the playbook:

```
---

- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes

  - name: install apache2 package
    apt:
      name: apache2

  - name: add php support for apache
    apt:
      name: libapache2-mod-php
```

here we added php support to the Apache server. run the playbook to make changes and notice the output.

now this playbook will install apache2 and libapache2-mod-php packages if they are not installed but it won’t update them if there are updates available. To make the playbook capable of updating packages we need to use the state parameter. let’s edit the playbook as follows:

```
---
- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes

  - name: install apache2 package
    apt:
      name: apache2
      state: latest

  - name: add php support for apache
    apt:
      name: libapache2-mod-php
      state: latest
```

state: latest will make sure the package is always the latest one available.

open a browser and open one of the server IPs to see the default Apache start page.

let’s create another playbook that removes these packages. you can either write it from scratch or make a copy of the install_apache.yml file and make changes.

create remove_apache.yml with the following contents:

```
---

- hosts: all
  become: true
  tasks:

  - name: remove apache2 package
    apt:
      name: apache2
      state: absent

  - name: remove php support for apache
    apt:
      name: libapache2-mod-php
      state: absent
```

the **state: absent** parameter value means removing the package if present.

run the playbook and notice the output.

open a browser and try opening the site on one of your servers.

run the install playbook and refresh your browser page.

Version Control

we added two files to our nislab directory which is connected to a git repository so we need to add both these files to gihub.

nislab$ git status
nislab$git add .
nislab$git commit -m “install/remove apache and php playbooks created by admin 1”
nilab$ git push origin main
now do a git pull on workstation2 to get these files downloaded.

The 'when' Conditional

The playbook we created will work fine if all servers are Debian based systems since we are using the apt module. if some of your servers have a base other than Debian then the playbook will fail when used on them.

on workstation2 modify the inventory file by adding the AlmaLinux server IP address. run the playbook and notice the output especially for AlmaLinux server.

The command failed since AlmaLinux is not a Debian based distribution rather it is based on RHEL (Red Hat Enterprise Linux) which doesn't have apt as a package manager instead it uses dnf.

let's modify the playbook to remove that error:

```
---

- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes
    when: ansible_distribution == "Ubuntu"

  - name: install apache2 package
    apt:
      name: apache2
      state: latest
    when: ansible_distribution == "Ubuntu"

  - name: add php support for apache
    apt:
      name: libapache2-mod-php
      state: latest
    when: ansible_distribution == "Ubuntu"
```

here we added [when: ansible_distribution == "Ubuntu"] to each of our tasks. run the playbook and notice what happens when executing each of the plays on AlmaLinux. you will see that it was skipped since it failed to meet the when condition.

To know the distributions of your servers you can run the gather_facts module on each server:

ansible all -m gather_facts | grep ansible_distribution

now let's modify the playbook to also do the same thing for the AlmaLinux server:

```
---

- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes
    when: ansible_distribution == "Ubuntu"
  - name: install apache2 package
    apt:
      name: apache2
      state: latest
    when: ansible_distribution == "Ubuntu"

  - name: add php support for apache
    apt:
      name: libapache2-mod-php
      state: latest
    when: ansible_distribution == "Ubuntu"

  - name: update repository index
    dnf:
      update_cache: yes
    when: ansible_distribution == "AlmaLinux"

  - name: install httpd package
    dnf:
      name: httpd
      state: latest
    when: ansible_distribution == "AlmaLinux"

  - name: add php support for apache
    dnf:
      name: php
      state: latest
    when: ansible_distribution == "AlmaLinux"
```

run the playbook and check the output. one thing to note is that although Apache and php will be installed on the AlmaLinux server but the site won’t open since AlmaLinux doesn’t automatically start those services after installing like Ubuntu does.

Push all changes to git and pull them on workstation 2 since we need to do some tasks on workstation 2.

Improving The Playbook

our playbook includes many unnecessary lines that we can omit. On workstation 2 edit the playbook as follows:

```
---

- hosts: all
  become: true
  tasks:

  - name: update repository index
    apt:
      update_cache: yes
    when: ansible_distribution == "Ubuntu"

  - name: install apache2 and php packages for Ubuntu
    apt:
      name:
        - apache2
        - libapache2-mod-php
      state: latest
    when: ansible_distribution == "Ubuntu"

  - name: update repository index
    dnf:
      update_cache: yes
    when: ansible_distribution == "AlmaLinux"

  - name: install apache and php packages for AlmaLinux
    dnf:
      name:
        - httpd
        - php
      state: latest
    when: ansible_distribution == "AlmaLinux"
```

here all we did is adding multiple packages to the apt module to be installed on the system. this ways we need only one Task or play to install all needed packages.

run the playbook just to make sure we have no syntax errors in the file.

since the update cache is a parameter of the apt module we can also eliminate that task as follows:

```
---

- hosts: all
  become: true
  tasks:

  - name: install packages Ubuntu
    apt:
      name:
        - apache2
        - libapache2-mod-php
      state: latest
      update_cache: yes
    when: ansible_distribution == "Ubuntu"

  - name: install packages AlmaLinux
    dnf:
      name:
        - httpd
        - php
      state: latest
      update_cache: yes
    when: ansible_distribution == "AlmaLinux"
```

now our playbook is down to 2 plays only. run the playbook to make sure we have no errors. this playbook will run a bit faster.

Create a backup of both the playbook and inventory files before proceeding.

Now we can even get our playbook down to one play using variables. let’s edit the files as follows:

```
---
- hosts: all
  become: true
  tasks:
  - name: install apache and php
    package:
      name:
        - '{{apache_package}}'
        - '{{php_package}}'
      state: latest
      update_cache: yes
```

package module is a generic package manager which means it will use the default package manager of each distribution. the {{apache_package}} and {{php_package}} are variables we created and we can use any name we want. now for this to work we need to edit the inventory file to include the package names. add the following to the right of each IP address that refers to an ubuntu server:

```
A.A.A.A  apache_package=apache2   php_package=libapache2-mod-php

B.B.B.B  apache_package=httpd     php_package=php
```

A.A.A.A: Ubuntu server IP

B.B.B.B: AlmaLinux IP

now run the playbook to make sure it works.

Targeting Specific Nodes

For this part we need to **revert to the backups** of the inventory and playbook files we created. now suppose that we have different roles that our servers have. suppose we have web, database, and file servers. if we want to target the web servers with certain tasks while doing other tasks for the file and web servers then we need to categories our servers in the inventory file. let’s edit the inventory file as follows:

```
[web_servers]
A.A.A.A
B.B.B.B

[db_servers]
C.C.C.C
B.B.B.B

[file_servers]
D.D.D.D
```

A.A.A.A and B.B.B.B are one ubuntu and one AlmaLinux server IP addresses, while C.C.C.C and D.D.D.D are the remaining two ubuntu servers.

Make a new copy of the install_apache.yml playbook and name it site.yml. now open site.yml and do the following changes:

```
---

- hosts: all
  become: true
  tasks:

  - name: install updates (AlmaLinux)
    dnf:
      update_only: yes
      update_cache: yes
    when: ansible_distribution == "AlmaLinux"

  - name: install updates (Ubuntu)
    apt:
      upgrade: dist
      update_cache: yes
    when: ansible_distribution == "Ubuntu"


- hosts: web_servers
  become: true
  tasks:

  - name: install apache and php for Ubuntu servers
    apt:
      name:
        - apache2
        - libapache2-mod-php
      state: latest
    when: ansible_distribution == "Ubuntu"

  - name: install apache and php for AlmaLinux servers
    dnf:
      name:
        - httpd
        - php
      state: latest
    when: ansible_distribution == "AlmaLinux"
```

On Ubuntu to do an apt upgrade we use the upgrade: dist parameter and on `Alma Linux` we use the upgrade_only: yes parameter.
run the playbook and notice how the execution now is targeting certain roles.
Note: if you need certain tasks to be executed first the use **pre_tasks:** instead of **tasks:**.

let’s add tasks for other roles as well. edit the playbook again by adding the following:

```
 - hosts: db_servers
   become: true
   tasks:

   - name: install Database package (AlmaLinux)
     dnf:
       name: mariadb
       state: latest
     when: ansible_distribution == "AlmaLinux"

   - name: install mariadb server
     apt:
       name: mariadb-server
       state: latest
     when: ansible_distribution == "Ubuntu"
```

run the playbook again.
add a section for file servers too:

```
 - hosts: file_servers
   become: true
   tasks:

   - name: install samba package
     package:
       name: samba
       state: latest
```

run the playbook again.

Using Tags

We can add tags to our tasks which is another way to execute plays on certain hosts. suppose we want to run all plays on Ubuntu servers that have Apache installed. Tags are words you add to each task in the tag line. let’s edit our playbook as follows:

```
---
 - hosts: all
   become: true
   pre_tasks:
   - name: install updates (AlmaLinux)
     tags: always
     dnf:
       update_only: yes
       update_cache: yes
```

```
    when: ansible_distribution == "AlmaLinux"
  - name: install updates (Ubuntu)
    tags: always
    apt:
      upgrade: dist
      update_cache: yes
    when: ansible_distribution == "Ubuntu"
- hosts: web_servers
  become: true
  tasks:
  - name: install httpd package (AlmaLinux)
    tags: apache,Alma,httpd
    dnf:
      name:
        - httpd
        - php
      state: latest
    when: ansible_distribution == "AlmaLinux"
  - name: install apache2 package (Ubuntu)
    tags: apache,apache2,ubuntu
    apt:
      name:
        - apache2
        - libapache2-mod-php
      state: latest
    when: ansible_distribution == "Ubuntu"
- hosts: db_servers
  become: true
  tasks:
  - name: install mariadb server package (AlmaLinux)
    tags: Alma,db,mariadb
    dnf:
      name: mariadb
      state: latest
    when: ansible_distribution == "AlmaLinux"
  - name: install mariadb server
    tags: db,mariadb,ubuntu
    apt:
      name: mariadb-server
      state: latest
    when: ansible_distribution == "Ubuntu"
- hosts: file_servers
  tags: samba
  become: true
  tasks:
  - name: install samba package
    tags: samba
    package:
      name: samba
      state: latest
```

here all we did was adding the lines in bold to the file. now let’s play with tags. to list available tags in a file use the following command:

```
ansible-playbook --list-tags site.yml
```

now let’s run the playbook using different tags and notice the output:

```
ansible-playbook --tags db --ask-become-pass site.yml
ansible-playbook --tags Alma --ask-become-pass site.yml
ansible-playbook --tags “apache,db” --ask-become-pass site.yml
```

now push all changes to GitHub and pull them on the second workstation.

Reflection

1. What is a playbook?
2. What is Yaml?
3. What is a package manager?
4. What is an ansible module?

END