OPTIMIZING ANSIBLE

Overview	
Goal	Tune how Ansible executes plays and tasks using host patterns, delegation, and parallelism
Objectives	• Specify managed hosts for plays and ad hoc commands using host patterns • Configure delegation in a playbook • Configure parallelism in Ansible
Sections	• Selecting Hosts with Host Patterns (and Guided Exercise) • Configuring Delegation (and Guided Exercise) • Configuring Parallelism (and Guided Exercise)
Lab	• Optimizing Ansible

Selecting Hosts with Host Patterns

Referencing Inventory Hosts

Host patterns are used to specify the hosts which should be targeted by a play or ad hoc command. In its simplest form, the name of a managed host or a host group in the inventory is a host pattern that specifies that host or host group.

You've already been using host patterns in this course. In a play, the **hosts** directive specifies the managed hosts to run the play against. For an ad hoc command, the host pattern is provided as a command line argument to the **ansible** command.

Host patterns are important to understand. It is usually easier to control what hosts a play targets by carefully using host patterns and having appropriate inventory groups, instead of setting complex conditionals on the play's tasks.
Managed Host

The most basic host pattern is the name for a single managed host listed in the inventory. This specifies that the host will be the only one in the inventory that will be acted upon by the **ansible** command.

The following example inventory will be used throughout this section to illustrate host patterns. The ansible command's **--list-hosts** option will be used to illustrate how some of these host patterns resolve.

References

Patterns — Ansible Documentation
http://docs.ansible.com/ansible/intro_patterns.html
Inventory — Ansible Documentation
http://docs.ansible.com/ansible/intro_inventory.html

Guided Exercise: Selecting Hosts with Host Patterns

Inspect the example inventory file. Notice how the inventory is organized. Explore which hosts are in the inventory, which domains are used, and which groups are in that inventory.

#cat inventory

```
[ansible@robo playbook]$ cat inventory
[local]
robo

[dev]
robo2

[test]
robo2
robo

[ip]
192.168.56.4
192.168.56.5
```

#ansible robo2 -i inventory --list-hosts

```
  hosts (1):
    robo2
```

#ansible 192.168.56.5 -i inventory --list-hosts

```
  hosts (1):
    192.168.56.5
```

#ansible all -i inventory --list-hosts

```
  hosts (4):
    robo2
    robo
    192.168.56.4
    192.168.56.5
```

#ansible test -i inventory --list-hosts

```
  hosts (2):
    robo2
    robo
```

#ansible '192.168.*' -i inventory --list-hosts

```
  hosts (2):
    192.168.56.4
    192.168.56.5
```

Using a list, access all hosts that belong to both the **test** and **dev** groups.

#ansible 'test,&dev' -i inventory --list-hosts

```
  hosts (1):
    robo2
```

Configuring Delegation

Configuring delegation

In order to complete some configuration tasks, it may be necessary for actions to be taken on a different server than the one being configured. Some examples of this might include an action that requires waiting for the server to be restarted, adding a server to a load balancer or a monitoring server, or making changes to the DHCP or DNS database needed for the server being configured.

Delegation can help by performing necessary actions for tasks on hosts other than the managed host being targeted by the play in the inventory. Some scenarios that delegation can handle include:

- Delegating a task to the local machine
- Delegating a task to a host outside the play
- Delegating a task to a host that exists in the inventory
- Delegating a task to a host that does not exist in the inventory

Delegating tasks to the local machine

When any action needs to be performed on the node running Ansible, it can be delegated to localhost by using **delegate_to: localhost**.

Delegated Facts

Any facts gathered by a delegated task are assigned by default to the delegate_to host, instead of the host which actually produced the facts. The following example shows a task file that will loop through a list of inventory servers to gather facts.

```
- hosts: app_servers
  tasks:
    - name: gather facts from app servers
      setup:
      delegate_to: "{{item}}"
      with_items: "{{groups['lb_servers']}}"

    - debug:
        var: ansible_eth0['ipv4']['address']
```

References

Delegation — Delegation, Rolling Updates, and Local Actions — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_delegation.html#delegation
Delegated facts — Delegation, Rolling Updates, and Local Actions — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_delegation.html#delegated-facts

Guided Exercise: Configuring Delegation

In this exercise, you will configure the delegation of tasks in an Ansible playbook. The playbook will configure **robo2** as a proxy server and **robo** as an Apache web server. During the deployment of the website on **robo**, you will delegate the task of stopping the traffic coming to **robo** to **robo2** proxy server and later after deployment you will start the traffic coming to **robo** by delegating task to **robo2**.

#cd /home/ansible/playbook/configure-delegation

Create an inventory file named **hosts** under **~/configure-delegation/inventory**. The inventory file should have two groups defined: **webservers** and **proxyservers**.
The **robo** host should be part of the **webservers** group and **robo2** should be part of the **proxyservers** group.
cat inventory

```
[ansible@robo configure-delegation]$ cat inventory
[webservers]
robo

[proxyservers]
robo2
```

Create robo.conf.j2 & robo2.conf.j2 with following entries.
cat robo.conf.j2

```
<VirtualHost *:80>
    ServerAdmin webmaster@robo
    ServerName robo
    ErrorLog logs/robo-error.log
    CustomLog logs/robo-common.log common
    DocumentRoot /var/www/html/
    <Directory /var/www/html/index.html>
        Options +Indexes +FollowSymlinks +Includes
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
```

#cat robo2.conf.j2

```
<VirtualHost *:80>
    ServerAdmin webmaster@robo2
    ServerName robo2
    ErrorLog logs/robo2-error.log
    CustomLog logs/robo2-common.log common
    DocumentRoot /var/www/html/
    <Directory /var/www/html/robo2/>
        Options +Indexes +FollowSymlinks +Includes
        Order allow,deny
        Allow from all
    </Directory>
</VirtualHost>
```

Move the **robo.conf.j2**, template file that configures the virtual host to the **/configure - delegation/templates** directory. Later you will use an Ansible variable (inventory_hostname) to list the source of this file.
mkdir templates
mv robo.conf.j2 templates

Move the **configure-delegation/robo2.conf.j2** template file for configuring reverse proxy to the **configure-delegation/templates** directory.
mv robo2.conf.j2 templates

Create a template file, named **index.html.j2**, for the website to be hosted on **robo** under the **templates** directory. The file should contain the following content:

cat index.html.j2

```
[ansible@robo configure-delegation]$ cat index.html.j2
The webroot is {{ ansible_fqdn }}.
[ansible@robo configure-delegation]$
```

mv index.html.j2 templates

Create a playbook named **site.yml** in the main project directory, **~/configure-delegation**.The playbook should define a play to install and configure *httpd*. The play
should contain the following tasks:

• Install the *httpd* package and start and enable the service to **all** hosts defined in the inventory.
• Configure firewall to accept incoming **http** traffic.
• Copy the respective **httpd.conf** configuration file to the hosts serving as the web and proxy server. The resulting file should be named **myconfig.conf** under the **/etc/ httpd/conf.d/myconfig.conf** directory.

Add another play to the **site.yml** playbook. The play should contain the following tasks:
• Stop the proxy server on **robo2** using delegation.
• Deploy the web page by copying the **index.html.j2** to the **/var/www/html/ index.html** directory on **robo**.
• Start the proxy server on **robo2** using delegation.

cat site.yml

```
---
- name: Install and configure httpd
  hosts: all
  remote_user: ansible
  become: true
  tasks:
    - name: Install httpd
      yum:
        name: httpd
        state: installed
    - name: Start and enable httpd
      service:
        name: httpd
        state: started
        enabled: yes
    - name: Install firewalld
      yum:
        name: firewalld
        state: installed
    - name: Start and enabled firewalld
      service:
        name: firewalld
        state: started
        enabled: yes
    - name: Enable firewalld
      firewalld:
        zone: public
        service: http
        permanent: true
        state: enabled
        immediate: true
    - name: template server configs
      template:
        src: "templates/{{ inventory_hostname }}.conf.j2"
        dest: /etc/httpd/conf.d/myconfig.conf
        owner: root
        group: root
        mode: 0644
      notify:
        - restart httpd
  handlers:
    - name: restart httpd
      service:
        name: httpd
        state: restarted

- name: Deploy web service and disable proxy server
  hosts: webservers
  remote_user: ansible
  become: true
  tasks:
    - name: Stop Apache proxy server
      service:
        name: httpd
        state: stopped
      delegate_to:  "{{ item }}"
      with_items: "{{ groups['proxyservers'] }}"
    - name: Deploy webpages
      template:
        src:  templates/index.html.j2
        dest: /var/www/html/index.html
        owner: apache
        group: apache
        mode: 0644
    - name: Start Apache proxy server
      service:
        name: httpd
        state: started
      delegate_to: "{{ item }}"
      with_items: "{{ groups['proxyservers'] }}"
```

ansible-playbook --syntax-check site.yml
#ansible-playbook site.yml

```
[ansible@robo configure-delegation]$ ansible-playbook site.yml
SUDO password:

PLAY [Install and configure httpd] *********************************

TASK [Gathering Facts] *********************************************
ok: [robo]
ok: [robo2]

TASK [Install httpd] ***********************************************
changed: [robo2]
changed: [robo]

TASK [Start and enable httpd] **************************************
changed: [robo]
changed: [robo2]

TASK [Install firewalld] *******************************************
changed: [robo2]
changed: [robo]

TASK [Start and enabled firewalld] *********************************
changed: [robo2]
changed: [robo]

TASK [Enable firewalld] ********************************************
ok: [robo2]
ok: [robo]

TASK [template server configs] *************************************
changed: [robo]
changed: [robo2]

RUNNING HANDLER [restart httpd] ************************************
changed: [robo]
changed: [robo2]

PLAY [Deploy web service and disable proxy server] *****************

TASK [Gathering Facts] *********************************************
ok: [robo]

TASK [Stop Apache proxy server] ************************************
changed: [robo -> robo2] => (item=robo2)

TASK [Deploy webpages] *********************************************
changed: [robo]

TASK [Start Apache proxy server] ***********************************
changed: [robo -> robo2] => (item=robo2)

PLAY RECAP *********************************************************
robo                       : ok=12   changed=9    unreachable=0    failed=0
robo2                      : ok=8    changed=6    unreachable=0    failed=0
```

Output: -

curl http://robo

The webroot is robo.

Configuring Parallelism

Configure parallelism in Ansible using forks

Ansible allows much more control over the execution of the playbook by running the tasks in parallel on all hosts. By default, Ansible only fork up to five times, so it will run a particular task on five different machines at once. This value is set in the Ansible configuration file **ansible.cfg**.

```
[student@demo ~]$ grep forks /etc/ansible/ansible.cfg
#forks          = 5
```

When there are many managed hosts (more than five), the **forks** parameter can be changed to something more suitable for the environment. The default value can be either overridden in the configuration file by specifying a new value for the **forks** key, or the value can be changed using the **--forks** option for the **ansible-playbook** or **ansible** commands.

Running tasks in parallel

For any specific play, you can use the **serial** keyword in a playbook to temporarily reduce the number of machines running in parallel from the fork count specified in the Ansible configuration file. The **serial** keyword is primarily used to control rolling updates.

Rolling updates

If there is a website being deployed on 100 web servers, only 10 of them should be updated at the same time. The **serial** key can be set to 10 in the playbook to reduce the number of simultaneous deployments (assuming that the **fork** key was set to something higher). The **serial** keyword can also be specified as a percentage which will be applied to the total number of hosts in the play. If the number of hosts does not divide equally into the number of passes, the final pass will contain the modulus. Regardless of the percentage, the number of hosts per pass will always be 1 or greater.

```
---
- name: Limit the number of hosts this play runs on at the same time
  hosts: appservers
  serial: 2
```

Ansible, regardless of the number of forks set, only spins up the tasks based on the current number of hosts in a play.

Asynchronous tasks

There are some system operations that take a while to complete. For example, when downloading a large file or rebooting a server, such tasks takes a long time to complete. Using parallelism and forks, Ansible starts the command quickly on the managed hosts, then polls the hosts for status until they are all finished.

To run an operation in parallel, use the **async** and **poll** keywords. The **async** keyword triggers Ansible to run the job in the background and can be checked later, and its value will be the maximum time that Ansible will wait for the command to complete. The value of **poll** indicates to Ansible how often to poll to check if the command has been completed. The default **poll** value is **10** seconds.

In the example, the **get_url** module takes a long time to download a file and **async: 3600** instructs Ansible to wait for **3600** seconds to complete the task and **poll: 10** is the pollintime in seconds to check if the download is complete.

```
tasks:
  - name: Download big file
    get_url:
      url: http://demo.example.com/bigfile.tar.gz
    async: 3600
    poll: 10
```

Deferring asynchronous tasks

Long running operations or maintenance scripts can be carried out with other tasks, whereas checks for completion can be deferred until later using the **wait_for** module. To configure Ansible to not wait for the job to complete, set the value of **poll** to **0** so that Ansible starts the command and instead of polling for its completion it moves to the next tasks.

```
tasks:
  - name: restart machine
    shell: sleep 2 && shutdown -r now "Ansible updates triggered"
    async: 1
    poll: 0
```

For the running tasks that take an extremely long time to run, you can configure Ansible to wait for the job as long as it takes. To do this, set the value of **async** to **0**.

Asynchronous task status

While an asynchronous task is running, you can also check its completion status by using Ansible **async_status** module. The module requires the job or task identifier as its parameter.

References

Rolling Update Batch Size — Delegation, Rolling Updates, and Local Actions — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_delegation.html#rolling-update-batchsize
Asynchronous Actions and Polling — Ansible Documentation
http://docs.ansible.com/ansible/playbooks_async.html
async_status - Obtain status of asynchronous task — Ansible Documentation
http://docs.ansible.com/ansible/async_status_module.html
Ansible Performance Tuning (For Fun and Profit)
https://www.ansible.com/blog/ansible-performance-tuning

Guided Exercise: Configuring Parallelism

In this exercise, you will run a playbook which uses a script to performs a long-running process on **robo2** using an asynchronous task. Instead of waiting for the task to get completed, you will check the status using the **async_status** module.

#cd /home/ansible/playbook/configure-async

Create a script file named **longfiles.j2** under the **~/configure-async/templates** directory, with the following content.

#cd /home/ansible/playbook/configure-async/templates

cat longfiles.j2

```
#!/bin/bash
echo "emptying $2" > $2
for i in {00..30}; do
echo "run $i, $1"
echo "run $i for $1" >> $2
sleep 1
done
```

Create a playbook named **async.yml** under the lab's project directory. In the playbook, use the **webservers** inventory host group, the **ansible** remote user, and for privilege escalation, use the **root** user and **sudo** method.

#cat async.yml

```
---
- name: longfiles async playbook
  hosts: webservers
  remote_user: ansible
  become: true

  tasks:
    - name: template longfiles script
      template:
        src: templates/longfiles.j2
        dest: /usr/local/bin/longfiles
        owner: root
        group: root
        mode: 0755

    - name: run longfiles script
      command: "/usr/local/bin/longfiles {{ item }} /tmp/{{ item }}.file"
      async: 110
      poll: 0
      with_items:
        - foo
        - bar
        - baz
      register: script_sleeper

    - name: show script_sleeper value
      debug:
        var: script_sleeper

    - name: check the status of longfiles script
      async_status: "jid={{ item.ansible_job_id }}"
      register: job_result
      until: job_result.finished
      retries: 30
      with_items: "{{ script_sleeper.results }}"
```

#ansible-playbook --syntax-check async.yml
#ansible-playbook async.yml

```
[ansible@robo configure-async]$ ansible-playbook async.yml
SUDO password:

PLAY [longfiles async playbook] ************************************************************************************************************************

TASK [Gathering Facts] *********************************************************************************************************************************
ok: [robo2]

TASK [template longfiles script] ***********************************************************************************************************************
changed: [robo2]

TASK [run longfiles script] ****************************************************************************************************************************
changed: [robo2] => (item=foo)
changed: [robo2] => (item=bar)
changed: [robo2] => (item=baz)

TASK [show script_sleeper value] ***********************************************************************************************************************
ok: [robo2] => {
    "script_sleeper": {
        "changed": true,
        "msg": "All items completed",
        "results": [
            {
                "_ansible_ignore_errors": null,
                "_ansible_item_label": "foo",
                "_ansible_item_result": true,
                "_ansible_no_log": false,
                "_ansible_parsed": true,
                "ansible_job_id": "829283374466.4129",
                "changed": true,
                "failed": false,
                "finished": 0,
                "item": "foo",
                "results_file": "/root/.ansible_async/829283374466.4129",
                "started": 1
            },
            {
                "_ansible_ignore_errors": null,
                "_ansible_item_label": "bar",
                "_ansible_item_result": true,
                "_ansible_no_log": false,
                "_ansible_parsed": true,
                "ansible_job_id": "277438377795.4209",
                "changed": true,
                "failed": false,
                "finished": 0,
                "item": "bar",
                "results_file": "/root/.ansible_async/277438377795.4209",
                "started": 1
            },
            {
                "_ansible_ignore_errors": null,
                "_ansible_item_label": "baz",
                "_ansible_item_result": true,
                "_ansible_no_log": false,
                "_ansible_parsed": true,
                "ansible_job_id": "857637495695.4290",
                "changed": true,
                "failed": false,
                "finished": 0,
                "item": "baz",
                "results_file": "/root/.ansible_async/857637495695.4290",
                "started": 1
            }
        ]
    }
}

TASK [check the status of longfiles script] ************************************************************************************************************
FAILED - RETRYING: check the status of longfiles script (30 retries left).
FAILED - RETRYING: check the status of longfiles script (29 retries left).
FAILED - RETRYING: check the status of longfiles script (28 retries left).
FAILED - RETRYING: check the status of longfiles script (27 retries left).
FAILED - RETRYING: check the status of longfiles script (26 retries left).
changed: [robo2] => (item={'_ansible_parsed': True, '_ansible_item_result': True, '_ansible_item_label': u'foo', u'ansible_job_id': u'
829283374466.4129', 'failed': False, u'started': 1, 'changed': True, 'item': u'foo', u'finished': 0, u'results_file': u'/root/.ansible
_async/829283374466.4129', '_ansible_ignore_errors': None, '_ansible_no_log': False})
changed: [robo2] => (item={'_ansible_parsed': True, '_ansible_item_result': True, '_ansible_item_label': u'bar', u'ansible_job_id': u'
277438377795.4209', 'failed': False, u'started': 1, 'changed': True, 'item': u'bar', u'finished': 0, u'results_file': u'/root/.ansible
_async/277438377795.4209', '_ansible_ignore_errors': None, '_ansible_no_log': False})
changed: [robo2] => (item={'_ansible_parsed': True, '_ansible_item_result': True, '_ansible_item_label': u'baz', u'ansible_job_id': u'
857637495695.4290', 'failed': False, u'started': 1, 'changed': True, 'item': u'baz', u'finished': 0, u'results_file': u'/root/.ansible
_async/857637495695.4290', '_ansible_ignore_errors': None, '_ansible_no_log': False})

PLAY RECAP *********************************************************************************************************************************************
robo2                      : ok=5    changed=3    unreachable=0    failed=0
```