Managing Host

Referencing Host Pattern

- Host patterns are used to specify the hosts to target 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 have already used 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, provide the host pattern as a command line argument to the **ansible** command.
- 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. Therefore, it is important to have a robust understanding of host patterns.
- *$ cat inventory_file_name*

Managed Hosts

- 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.
- When the playbook runs, the first **Gathering Facts** task should run on all managed hosts that match the host pattern. A failure during this task can cause the managed host to be removed from the play.
- If an IP address is listed explicitly in the inventory, instead of a host name, then it can be used as a host pattern. If the IP address is not listed in the inventory, then you cannot use it to specify the host even if the IP address resolves to that host name in the DNS.

```
---
- name: Managed host
  gather_facts: false
  hosts: 172.16.79.135
  tasks:
  - debug: msg="Managing Host Pattern"
...
```

Managing Host

Specifying Hosts Using a Group

- You have already used inventory host groups as host patterns. When a group name is used as a host pattern, it specifies that Ansible will act on the hosts that are members of the group.
- Remember that there is a special group named all that matches all managed hosts in the inventory.
- There is also a special group named ungrouped, which includes all managed hosts in the inventory that are not members of any other group:

```
---
- name: Managed host
  gather_facts: false
  hosts: active
  tasks:
  - debug: msg="Managing Host Pattern"
...
```

Matching Multiple Hosts with Wildcards

- Another method of accomplishing the same thing as the all host pattern is to use the asterisk (*) wildcard character, which matches any string.
- If the host pattern is just a quoted asterisk, then all hosts in the inventory will match.

```
---
- name: Managed host
  gather_facts: false
  hosts: ‘*’
  tasks:
  - debug: msg="Managing Host Pattern"
...
```

```
- hosts: '*.example.com'
- hosts: '192.168.2.*'
- hosts: 'datacenter*'
```

Managing Host

Lists

- Multiple entries in an inventory can be referenced using logical lists.
- A comma-separated list of host patterns matches all hosts that match any of those host patterns.
- You can also mix managed hosts, host groups, and wildcards,
- The colon character (:) can be used instead of a comma. However, the comma is the preferred separator, especially when working with IPv6 addresses as managed host names.

```
---
- name: Managed host
  gather_facts: false
  hosts: active:public:192.168.1.1
  tasks:
  - debug: msg="Managing Host Pattern"
...
```

Dynamic Inventory

- When working with numerous machines, however, or in an environment where machines come and go very quickly, it can be hard to keep the static inventory files up-to-date.
- Ansible supports **dynamic inventory** scripts that retrieve current information from these types of sources whenever Ansible executes, allowing the inventory to be updated in real time.
- These scripts are executable programs that collect information from some external source and output the inventory in **JSON** format.
- Dynamic inventory scripts are used just like static inventory text files. The location of the inventory is specified either directly in the current **ansible.cfg** file, or using the **-i** option.
- If the inventory file is executable, then it is treated as a dynamic inventory program and Ansible attempts to run it to generate the inventory. If the file is not executable, then it is treated as a static inventory.

Managing Host

- Dynamic inventory will fetch your IP address automatically
- Generally all cloud environments are dynamic environments.
- Ansible supports many cloud environments such as AWS EC2, OpenStack, Google Compute Engine, etc.
- The custom program can be written in any programming language, but must return inventory information in JSON format when passed appropriate options.

Working with Default AWS EC2 dynamic inventory file.

- https://docs.ansible.com/
- → Inventory → working with dynamic Inventory → inventory scripts → download this two file (ec2.ini and ec2.py) copy and paste.
- Provide execution permission (chmod +x ec2.py)
- Check python version (python - -version) (python3 is install you will get error)
- **sudo alternatives --set python /usr/bin/python3 (set the python path)**
- ./ec2.py (try to execute)

  Error Message you will get

  {

  [Traceback (most recent call last):

  File "./ec2.py", line 144, in <module>

  import boto

  ModuleNotFoundError: No module named 'boto']

  }
- cat ec2.py | grep boto
- **sudo pip3 install boto** [Installing boto with pip- python package]
- **Go to AWS account – EC2 instance:**
- Instance → Action→Instance Setting → Attach / Replace I am role → Create new I am role → Create role → Select EC2 → click on Next: Permission → type in Filter Policies “EC2fullpermission” and “AmazonElastCacheFullAcces” → Select and click Next : Tag→ Click Next: Review → Role name “Ansible_Access” → Create Role
- Go to Attaché / Replace I am page and refresh the role.
- Select you newly created role from drop down list → apply
- Run the ec2.py script [./ec2.py]

Writing Custom Dynamic inventory for AWS

```
#!/usr/bin/python
import json
import sys
try:
    import boto3
except Exception as e:
    print(e)
    print("Error, kinldy resolve")
    sys.exit(1)

def get_hosts(ec2_ob,fv):
    f={"Name":"tag:Env" , "Values": [fv]}
    hosts=[]
    for each_in in ec2_ob.instances.filter(Filters=[f]):
       # print(each_in.private_ip_address)
        hosts.append(each_in.private_ip_address)
    return hosts
def main():
    ec2_ob=boto3.resource("ec2","ap-south-1")
    db_group=get_hosts(ec2_ob,'db')
    app_group=get_hosts(ec2_ob,'app')
    all_groups= { 'db' : db_group, 'app' : app_group }
    print(json.dumps(all_groups))
    return None

if __name__=="__main__":
    main()
```