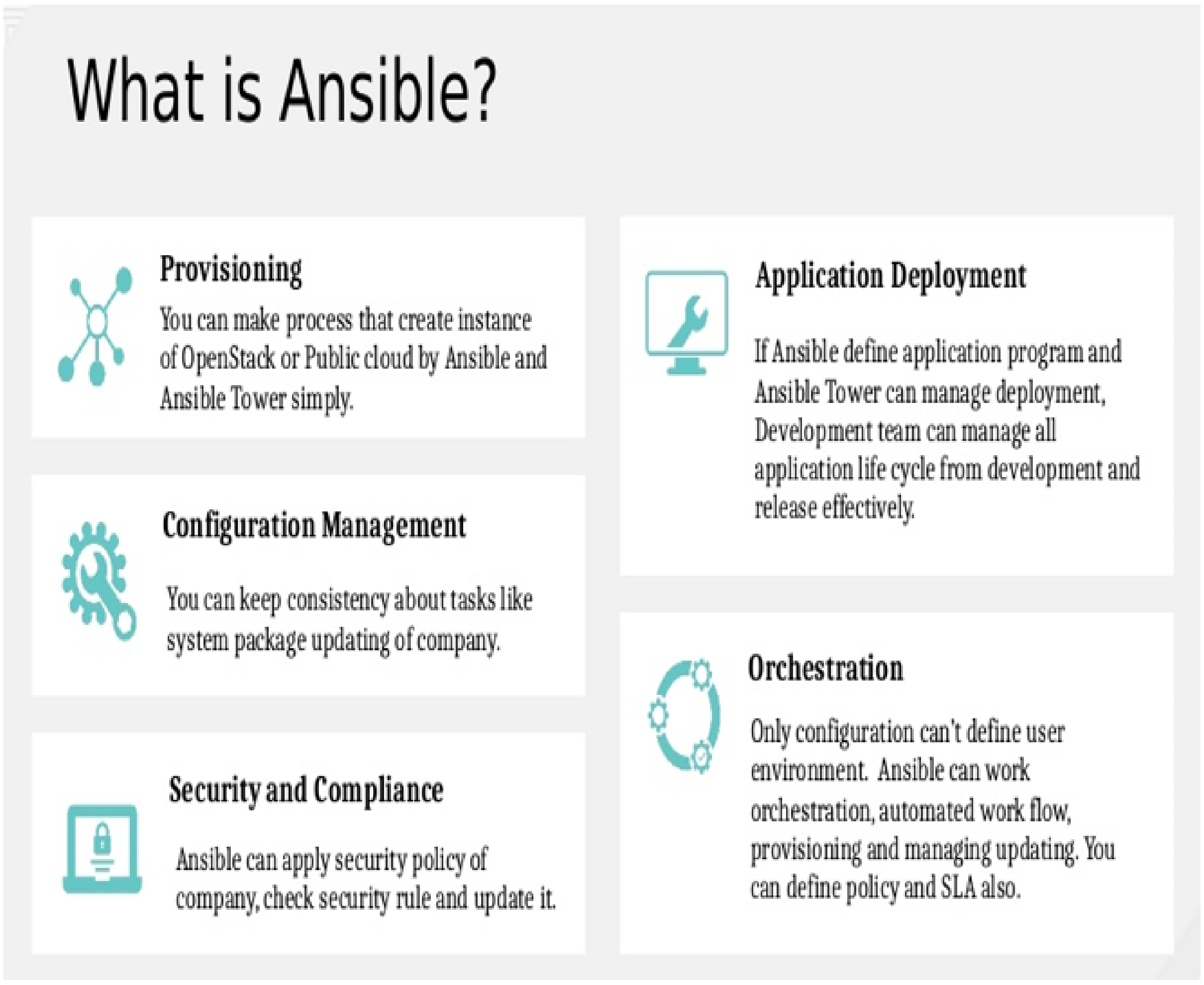
Ansible



What is Ansible?

Provisioning

You can make process that create instance of OpenStack or Public cloud by Ansible and Ansible Tower simply.

Application Deployment

If Ansible define application program and Ansible Tower can manage deployment, Development team can manage all application life cycle from development and release effectively.

Configuration Management

You can keep consistency about tasks like system package updating of company.

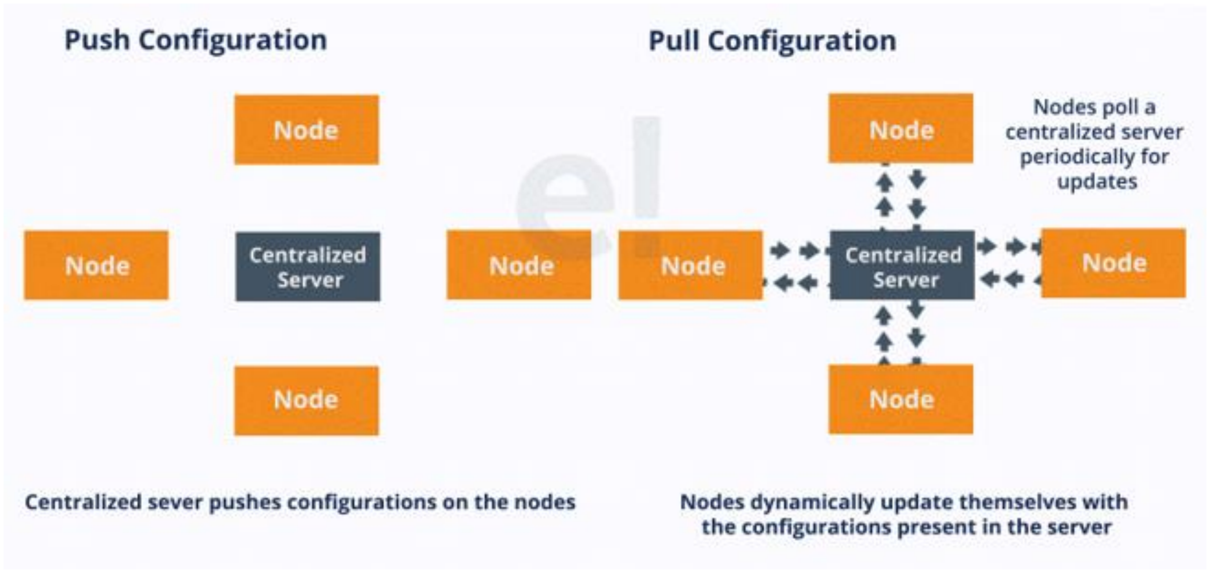
Orchestration

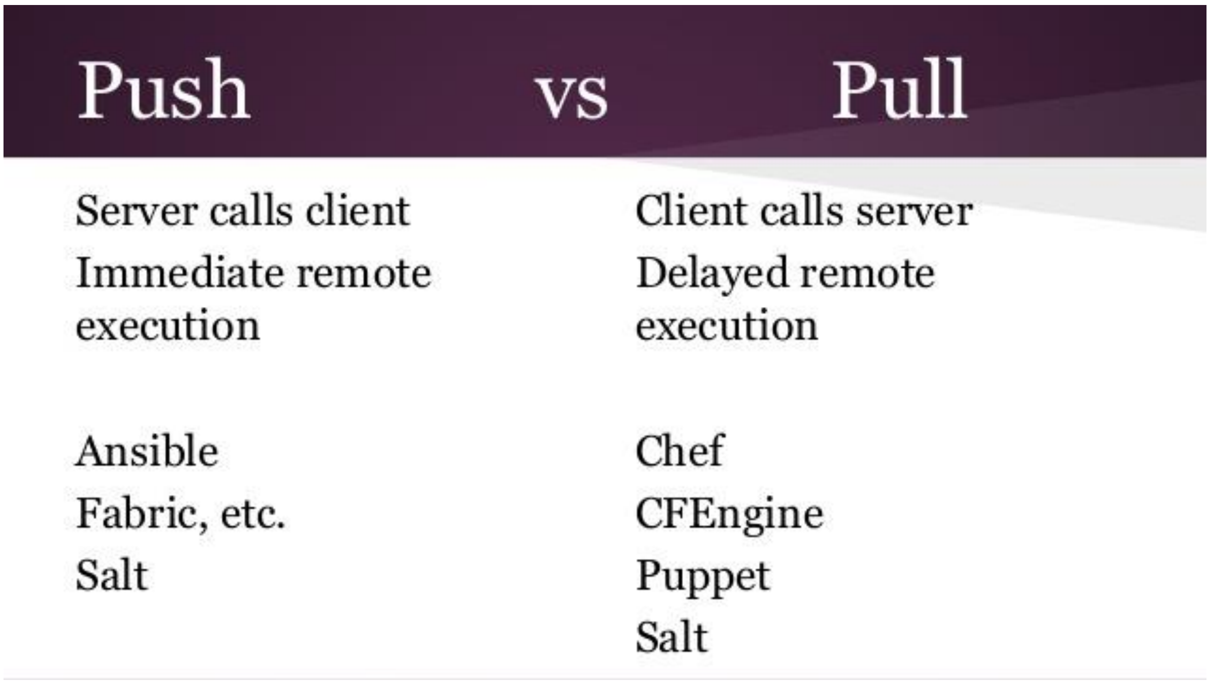
Only configuration can't define user environment. Ansible can work orchestration, automated work flow, provisioning and managing updating. You can define policy and SLA also.

Security and Compliance

Ansible can apply security policy of company, check security rule and update it.

Push vs Pull configuration

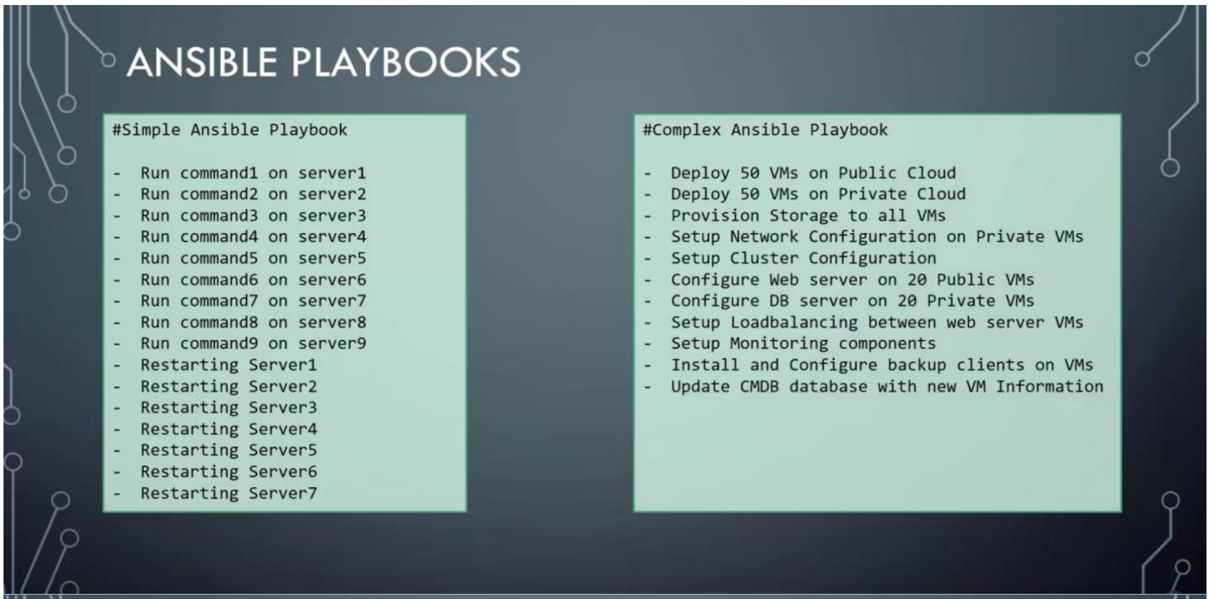

Push Configuration
Node
Node
Centralized Server
Node
Centralized sever pushes configurations on the nodes
Pull Configuration
Node
Nodes poll a centralized server periodically for updates
Node
Node
Centralized Server
Node
Node
Nodes dynamically update themselves with the configurations present in the server



Push	vs	Pull
Server calls client		Client calls server
Immediate remote execution		Delayed remote execution
Ansible		Chef
Fabric, etc.		CFEngine
Salt		Puppet
		Salt

Push Based vs. Pull Based

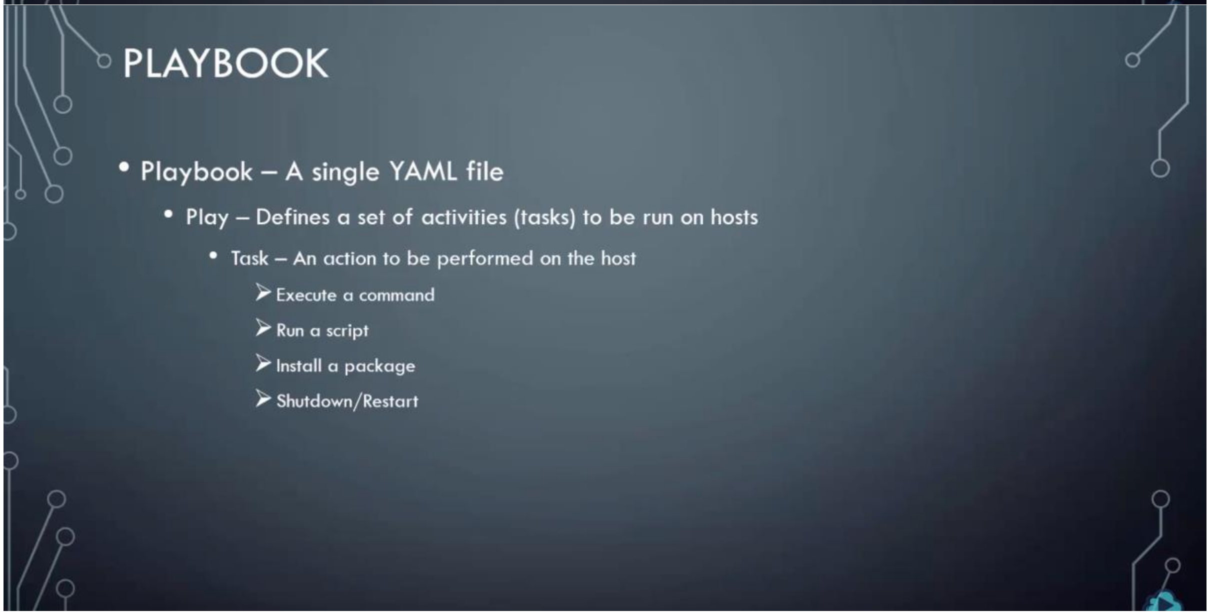
- Tool like Puppet and Chef are pull based
  - Agents on the server periodically checks for the configuration information from central server (Master)
- Ansible is push based
  - Central server pushes the configuration information on target servers
  - You control when the changes are made on the servers
  - Ansible has official support for pull mode, using a tool it ships with called *ansible-pull*.



ANSIBLE PLAYBOOKS

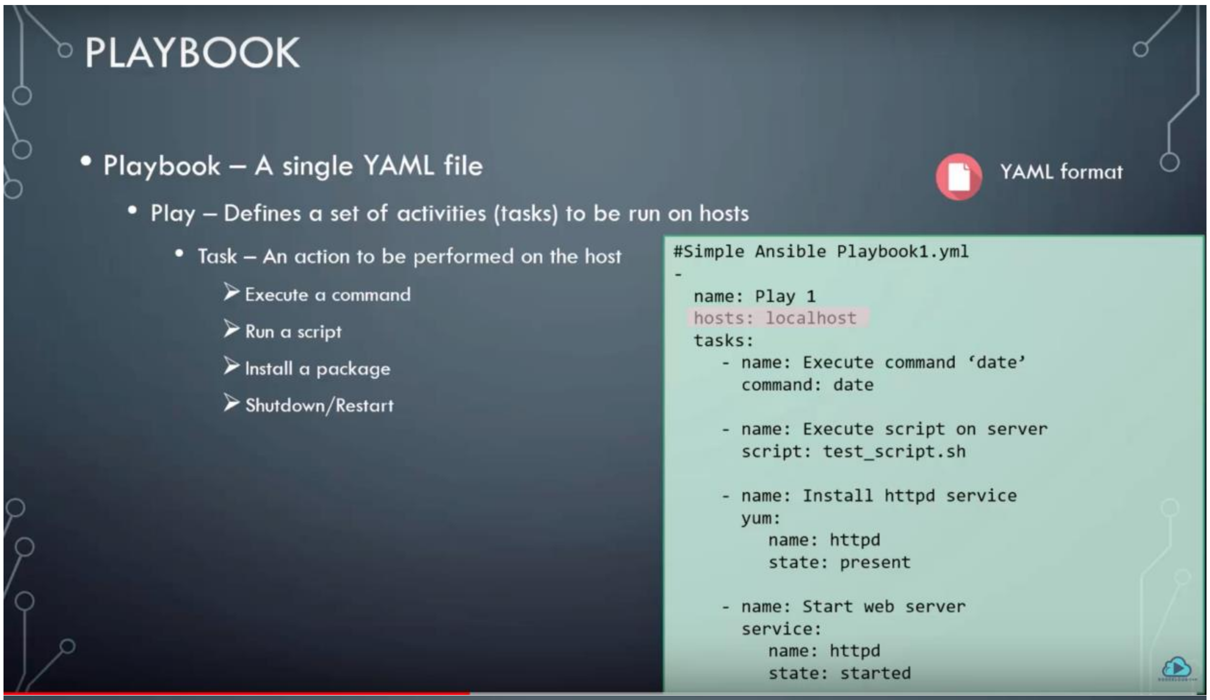
```
#Simple Ansible Playbook

-  Run command1 on server1
-  Run command2 on server2
-  Run command3 on server3
-  Run command4 on server4
-  Run command5 on server5
-  Run command6 on server6
-  Run command7 on server7
-  Run command8 on server8
-  Run command9 on server9
-  Restarting Server1
-  Restarting Server2
-  Restarting Server3
-  Restarting Server4
-  Restarting Server5
-  Restarting Server6
-  Restarting Server7
```

```
#Complex Ansible Playbook

-  Deploy 50 VMs on Public Cloud
-  Deploy 50 VMs on Private Cloud
-  Provision Storage to all VMs
-  Setup Network Configuration on Private VMs
-  Setup Cluster Configuration
-  Configure Web server on 20 Public VMs
-  Configure DB server on 20 Private VMs
-  Setup Loadbalancing between web server VMs
-  Setup Monitoring components
-  Install and Configure backup clients on VMs
-  Update CMDB database with new VM Information
```



PLAYBOOK

- Playbook – A single YAML file
  - Play – Defines a set of activities (tasks) to be run on hosts
    - Task – An action to be performed on the host
      - ➢ Execute a command
      - ➢ Run a script
      - ➢ Install a package
      - ➢ Shutdown/Restart

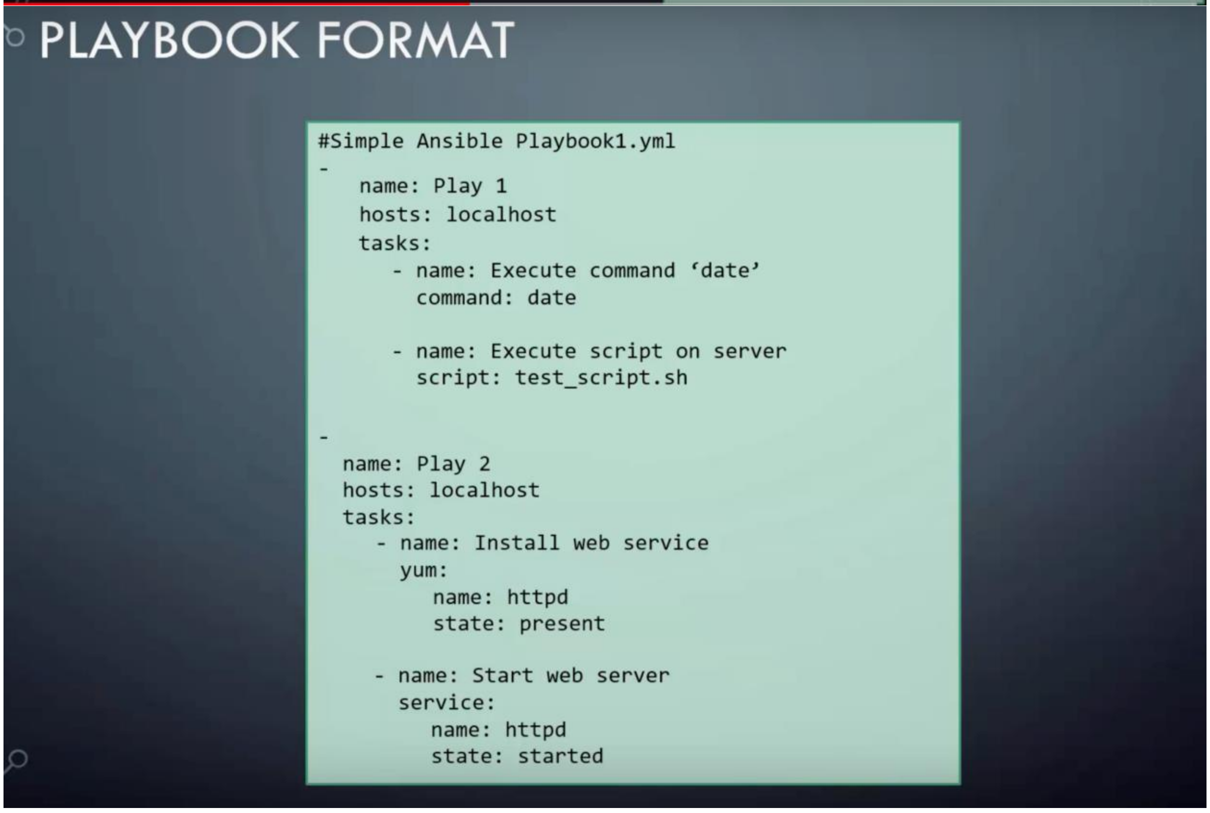


PLAYBOOK

- Playbook – A single YAML file
  - Play – Defines a set of activities (tasks) to be run on hosts
    - Task – An action to be performed on the host
      - ➢ Execute a command
      - ➢ Run a script
      - ➢ Install a package
      - ➢ Shutdown/Restart

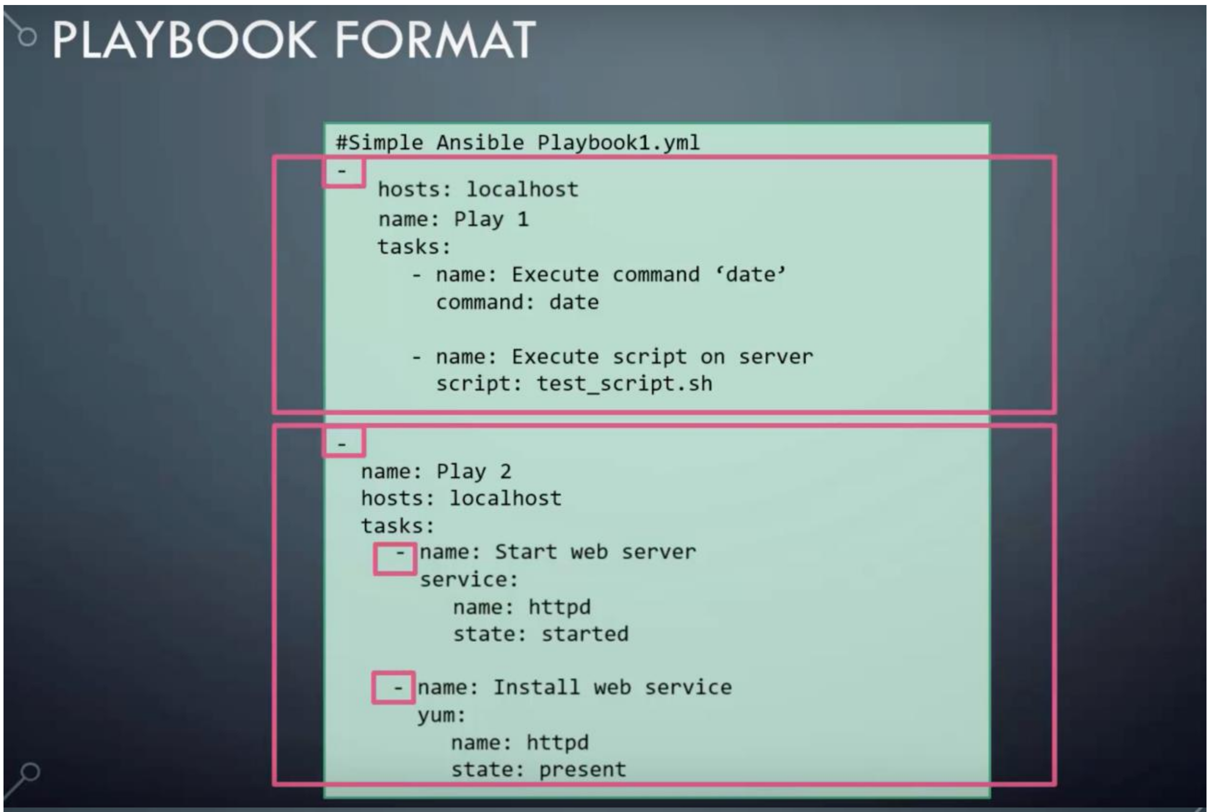
YAML format

```
#Simple Ansible Playbook1.yml
-
  name: Play 1
  hosts: localhost
  tasks:
    - name: Execute command ‘date’
      command: date

    - name: Execute script on server
      script: test_script.sh

    - name: Install httpd service
      yum:
        name: httpd
        state: present

    - name: Start web server
      service:
        name: httpd
        state: started
```



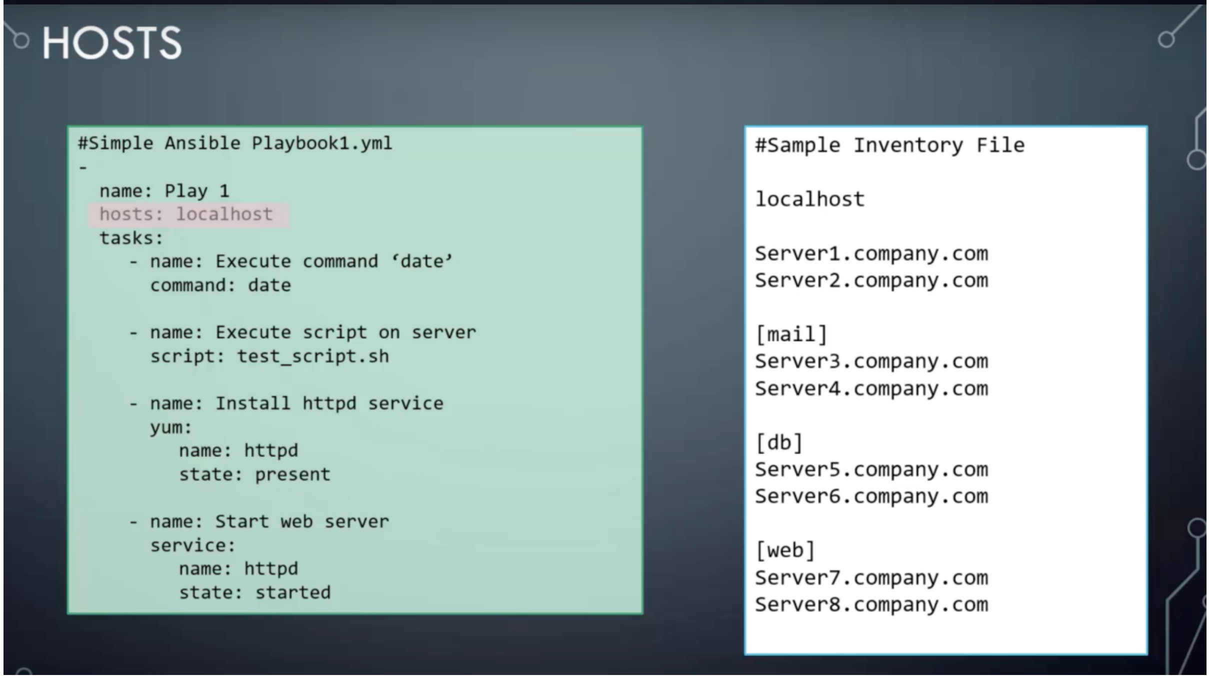
PLAYBOOK FORMAT

```
#Simple Ansible Playbook1.yml
-
   name: Play 1
   hosts: localhost
   tasks:
      - name: Execute command ‘date’
        command: date

      - name: Execute script on server
        script: test_script.sh

-
  name: Play 2
  hosts: localhost
  tasks:
     - name: Install web service
       yum:
          name: httpd
          state: present

     - name: Start web server
       service:
          name: httpd
          state: started
```



PLAYBOOK FORMAT

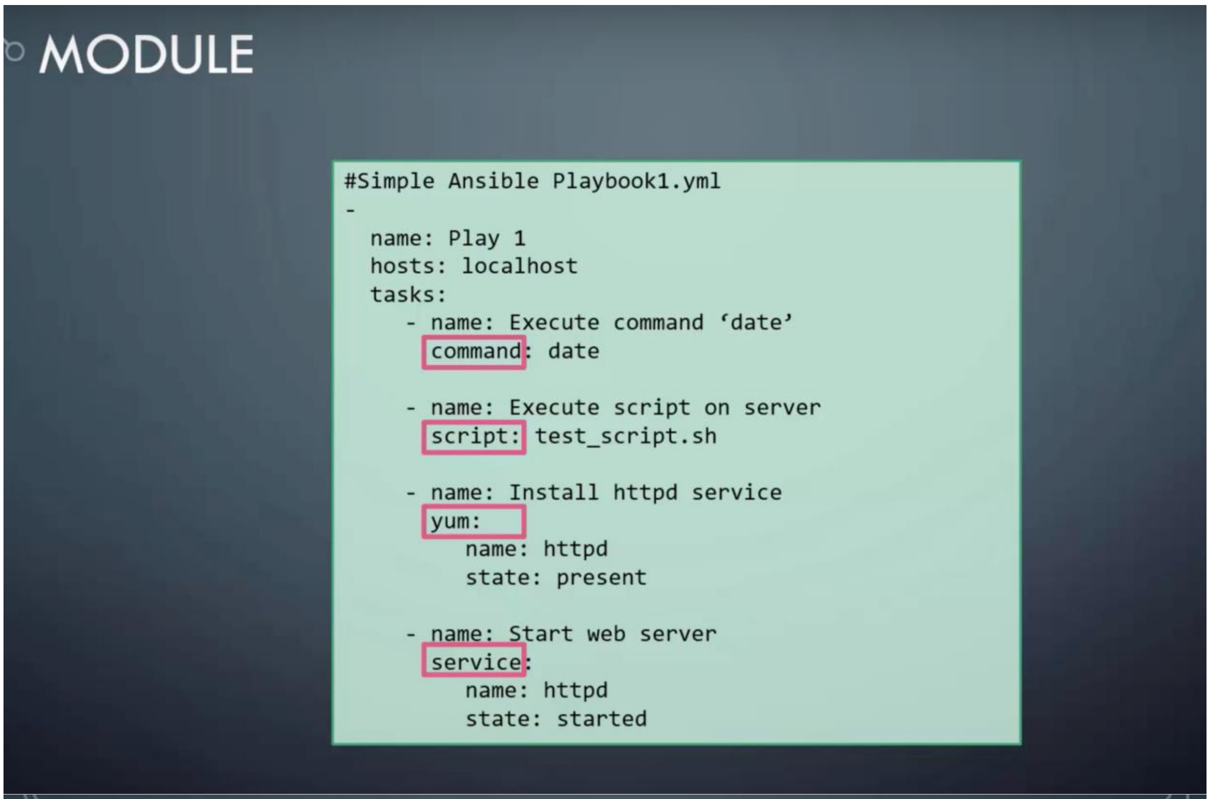
```
#Simple Ansible Playbook1.yml
-
   hosts: localhost
   name: Play 1
   tasks:
      - name: Execute command ‘date’
        command: date

      - name: Execute script on server
        script: test_script.sh
-
  name: Play 2
  hosts: localhost
  tasks:
    - name: Start web server
      service:
         name: httpd
         state: started

    - name: Install web service
      yum:
         name: httpd
         state: present
```



HOSTS

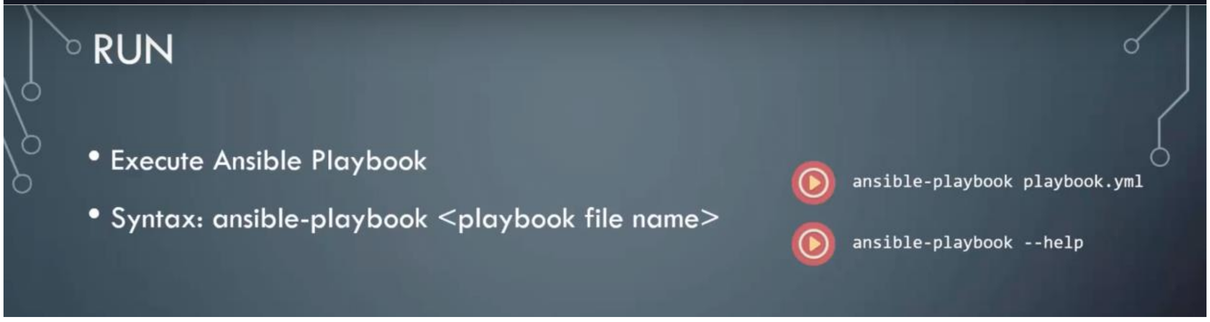
```
#Simple Ansible Playbook1.yml
-
  name: Play 1
  hosts: localhost
  tasks:
     - name: Execute command ‘date’
       command: date

     - name: Execute script on server
       script: test_script.sh

     - name: Install httpd service
       yum:
          name: httpd
          state: present

     - name: Start web server
       service:
          name: httpd
          state: started
```

```
#Sample Inventory File

localhost

Server1.company.com
Server2.company.com

[mail]
Server3.company.com
Server4.company.com

[db]
Server5.company.com
Server6.company.com

[web]
Server7.company.com
Server8.company.com
```



MODULE

```
#Simple Ansible Playbook1.yml
-
  name: Play 1
  hosts: localhost
  tasks:
     - name: Execute command ‘date’
       command: date

     - name: Execute script on server
       script: test_script.sh

     - name: Install httpd service
       yum:
          name: httpd
          state: present

     - name: Start web server
       service:
          name: httpd
          state: started
```



RUN

- Execute Ansible Playbook
- Syntax: ansible-playbook <playbook file name>

```
ansible-playbook playbook.yml
ansible-playbook --help
```

Example

Note that we can do the same for "ubuntu" or "ec2-user" for AWS instance with the inventory file (./hosts) like this:

```
[myservers]
3.93.171.48 ansible_user=ubuntu ansible_ssh_private_key_file=/Users/kihyuckhong/.ssh/einsteinish.pem
```

```
---
- hosts: webservers
  vars:
    http_port: 80
    max_clients: 200
  remote_user: root
  tasks:
  - name: ensure apache is at the latest version
    yum: name=httpd state=latest
  - name: write the apache config file
    template: src=/srv/httpd.j2 dest=/etc/httpd.conf
    notify:
    - restart apache
  - name: ensure apache is running (and enable it at boot)
    service: name=httpd state=started enabled=yes
  handlers:
    - name: restart apache
      service: name=httpd state=restarted
```

HOST INVENTORY:
#/etc/ansible/hosts/ (default Location)
just provide list of IP address & can be grouped

```
**

[local]
localhost ansible_connection=local

[appserver]
1.2.3.4
2.3.4.5

[webserver]
3.4.5.6

[dbserver]
5.6.7.8
**
```

Types of Ansible Inventories:

- **Static Inventory**
- **Dynamic Inventory**

Static Inventory

Static inventory is default inventory and is defined in the /etc/ansible/ansible.cfg file. Default file can be changed inside the ansible.cfg file. If you want to use the custom file as inventory input can specify it using" -i /path/to/file " with Ansible command line.

Example of Static Inventory:

```
# cat /etc/ansible/hosts
```

```
[all]

```

```
app-apache-finance  ansible_host=appserver1.opensky.home

db-mysql-marketing ansible_host=dbserver1.opensky.home

[appservers]

app-apache-finance ansible_host=appserver1.opensky.home

[dbservers]

db-mysql-marketing  ansible_host=dbserver1.opensky.home

[webserver-apache]

apache[01:50].example.com
```

Inventory table:

```
[all:vars]
ansible_user=student1
ansible_ssh_pass=PASSWORD
ansible_port=22

[web]
node1 ansible_host=<X.X.X.X>
node2 ansible_host=<Y.Y.Y.Y>
node3 ansible_host=<Z.Z.Z.Z>

[control]
ansible ansible_host=44.55.66.77
```

The below example Ansible Ad-Hoc command will check the ping status of all servers part of group called "all"

Example

```
# ansible -i /etc/Ansible/hosts.ini -m ping all
```

<u>Dynamic Inventory</u>

If you have the setup where you add and remove the hosts very frequently, then keeping your inventory always up-to-date become a little bit problematic. In such case Dynamic inventory comes into picture, generally are scripts (Python/Shell) for dynamic environments (for example cloud environments)

With Ansible, as aforementioned, can use "-i" to specify the custom inventory file.

For example, if you use AWS cloud and you manage EC2 inventory using its Query API, or through command-line tools such as awscli, then you can make use of dynamic inventory,

Dynamic inventory got benefits over static inventories:

- Reduces human error, as information is collected by scripts.
- Very less manual efforts for managing the inventories.

Ansible has inventory collection scripts for the below platforms

- AWS EC2 External Inventory Script, Collber, OpenStack, BSD Jails, Google Compute Engine, and Spacewalk.

Information Source: http://docs.ansible.com/ansible/intro_dynamic_inventory.html
For the quick demo, I have downloaded the Ansible Dynamic inventory script and related ec2.ini file from Ansible website.

dynamic-inven-ec.py	–	EC2 external inventory script (Python Script)
ec2.ini	–	Contains Ansible EC2 external inventory script settings

In the dynamic inventories, grouping can be managed by “tags, regions etc inside the ec2.ini file.

Example:

```
# ansible -i dynamic-inven-ec.py   -u ec2-user tag_Name_awslab*  -m ping
```

In the above example, Ansible will do the ping test for servers collected by dynamic inventory with tag key Name and Value with suffix “awslab”.

What is Ansible Tower?
Ansible Tower is Ansible at a more enterprise level. It is a web-based solution for managing your organization with a very easy user interface that provides a dashboard with all of the state summaries of all the hosts, allows quick deployments, and monitors all configurations.

The tower allows you to share the SSH credentials without exposing them, logs all the jobs, manage inventories graphically and syncs them with a wide variety of cloud providers.

AdHoc Commands

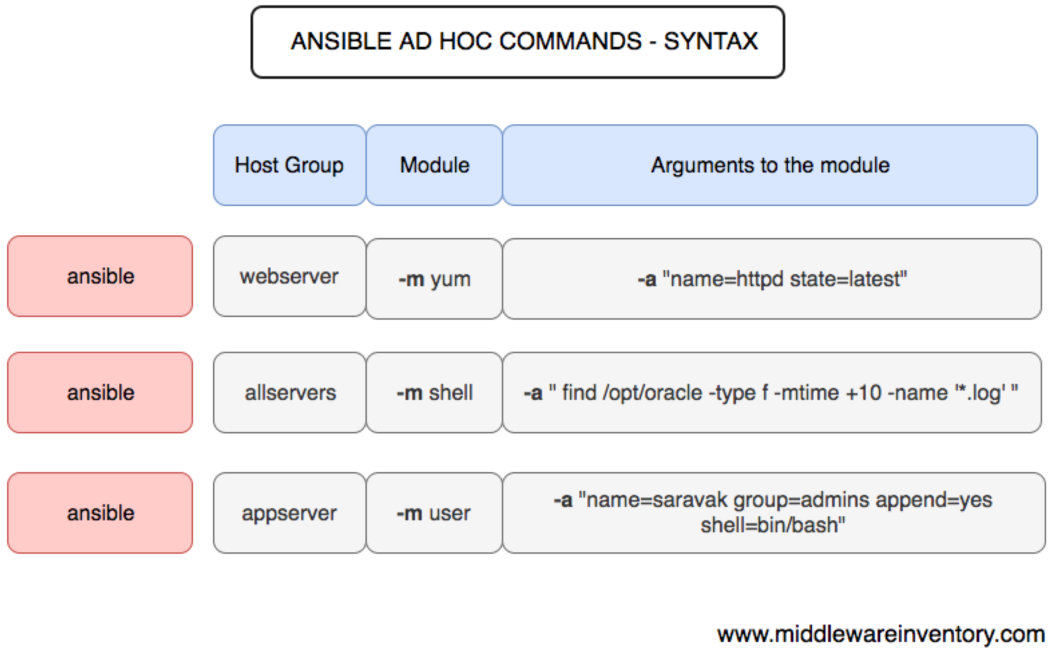
Adhoc commands are simple one line command to perform some action. Running modules with Ansible commands are adhoc commands.

E.g:

ansible Host Group –m Module -a Arguments to the module

ansible webserver -m netscaler -a "nsc_host=nsc.example.com user=apiuser password=apipass"
The above adhoc command uses the netscaler module to disable the server. There are hundreds of modules available in Ansible from where you can refer to and write adhoc commands.

Ansible ad_hoc commands Syntax


ANSIBLE AD HOC COMMANDS - SYNTAX
Host Group
Module
Arguments to the module
ansible
webserver
-m yum
-a "name=httpd state=latest"
ansible
allservers
-m shell
-a " find /opt/oracle -type f -mtime +10 -name '*.log' "
ansible
appserver
-m user
-a "name=saravak group=admins append=yes shell=bin/bash"
www.middlewareinventory.com

To run an ad hoc command, the command must be framed or have the following syntax.

```
ansible <host-pattern> [options]
```

For example. The command should be written as follows.

```
ansible appserverhostgroup -m <modulename> -a <arguments to the module>
```

A single ansible ad hoc command can have multiple options. -m and -a are one amongst them and widely used.

https://www.howtoforge.com/ansible-guide-ad-hoc-command/

1. Basic ad-Hoc Command

The basic command of ansible ad-hoc against 'all' hosts on the inventory file and using the 'ping' module.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible all -m ping
```

- The first parameter 'all' for all hosts on the inventory file.
- The second parameter inside the '-m' option for the module, running the ping module.

Now you will get the result as below.

An Ad-Hoc command against the provisioning server has been 'SUCCESS' without any changes made on the server and we get the result of the 'ping' module from the provisioning server 'pong'.

2. Filter Hosts Group and Single Host

- For default inventory file

Now you can use the Ad-Hoc command against a group of hosts that are already defined on the inventory file. You can use your custom inventory file or using the default inventory file '/etc/ansible/hosts'.

Below is an example to run the ad-hoc command against the group of hosts called 'hakase-testing' that are already defined on the default inventory configuration file.

Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m setup -a "filter=ansible_distribution*"
```

- For custom inventory file

Syntax: ansible Host Group –m Module -a Arguments to the module

If you're using the custom inventory file, add the '-i' option followed the inventory file name.

```
ansible hakase-testing -i hosts -m setup -a "filter=ansible_distribution*"
```

You will get the same result.

- single host on the inventory

Now if you want to run against single host on the inventory configuration, you can use the name of the host such as below.

Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible provision -m setup -a "filter=ansible_distribution*"
```

And the ad-hoc command will run on the 'provision' server only.

3. Using SSH Password

Now we will perform an Ad-Hoc command using the prompted ssh password authentication. And in order to do this, you need to install the additional package called 'sshpass' on the 'ansible-node'.

Install sshpass package using the apt command below.

```
sudo apt install sshpass -y
```

Now run the ad-hoc command and add the '--ask-pass' option to the end.

Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m ping --ask-pass
```

And you will be asked the 'SSH Password' for the server.

Type your ssh password and the ad-hoc command will be run against the server.

4. Privilege Escalation

The ansible provides features for the privilege escalation against servers. If you want to run the ad-hoc command as a non-root user, you can use the '--become' option to get the root privileges and the '-K' option to prompt the password.
Run the ad-hoc command 'fdisk -l' as a user 'hakase' with the privilege option '--become' and the '-K' to prompt the 'SUDO Password'.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m shell -a 'fdisk -l' -u hakase --become -K
```

Below is the result.

File Transfer

Now we're going to use the Ad-Hoc command for File Transfer to and from the server. We can transfer a file to the provisioning server with the 'copy' module, and download file from the server using 'fetch' module.

1. Upload File to Host

For this example, we're going to run the ad-hoc command and using the 'copy' module to upload the sudoers configuration for user hakase to the '/etc/sudoers.d' directory on the group 'hakase-testing'.
Run the ad-hoc command below.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m copy -a 'src=/home/hakase/hakase-sudo.conf dest=/etc/sudoers.d/ha
kase owner=root mode=0644' -u hakase --become -K
```

Now you will be asked the 'SUDO Password' for the hakase user. Type the password and you will get the result as below.

The file has been uploaded to the 'dest' destination directory, and you get the 'changed' result as 'true'.

2. Download File from the Host

Now we're going to use an ad-hoc command with the 'fetch' module for downloading the file from the provisioning server to the local 'ansible-node' server.
Download the configuration file '/etc/sudoers.d/hakase' from the 'provision' server to the local directory called 'backup'.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible provision -m fetch -a 'src=/etc/sudoers.d/hakase dest=/home/hakase/backup/hakase-su
doers flat=yes'
```

And you will get the file called 'hakase-sudoers' on the 'backup' directory.

Step 3 - Update Repository and Upgrade Packages

To update and upgrade the repository of Ubuntu servers, we can use the ad-hoc command with the apt module.
Update repository on the group hakase-testing.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m apt -a 'update_cache=yes' --become
```

Now update repositories and upgrade all packages to the latest version using the 'upgrade=dist' option.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m apt -a 'upgrade=dist update_cache=yes' --become
```

Wait for all packages is being upgraded.

Manage Packages

This is very useful when you're trying to build and debug your own playbook. Because sometimes you need an additional package to be installed on the system. So, this ad-hoc command will give you an easy way to install that package without login to each server.

1. Install Package

Install a single package using the ad-hoc command with the apt module as below.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m apt -a 'name=nginx state=latest' --become
```

2. Remove Package

Remove the package and purge all configuration related to the package.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m apt -a 'name=nginx state=absent purge=yes' --become
```

3. Autoremove

The example below is removing the nginx package and purge all configuration related and then remove all unused packages on the system.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m apt -a 'name=nginx state=absent purge=yes autoremove=yes' --beco
me
```

Manage Services

In this step, we're going to use the service module on the ad-hoc command for managing the system service on the provisioning server.
1. Start Services

Start the nginx service and add it to the boot time.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m service -a 'name=nginx state=started enabled=yes' --become
```

You will get 'changed' and 'enabled' result as 'true'.
2. Restart Service

If you want to restart the service, you can use the following command.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m service -a 'name=nginx state=restarted' --become
```

The nginx service has been restarted.
3. Stop a Service

To stop the service, change the 'state' value to 'stopped'.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m service -a 'name=nginx state=stopped' --become
```

The nginx service on the servers 'hakase-testing' has been stopped.
Checking the System

Now we're going to use the 'shell' module inside the ad-hoc command. And we will do simple system monitoring using simple Linux command through the Ansible ad-hoc.
Firstly, install the 'sysstat' package to all servers using the ad-hoc command below.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m apt -a 'name=sysstat state=latest' --become
```

Wait for the 'sysstat' package installation.
Once it's complete, you're ready to check all servers.
1. Disk Available

Check the disk available on the root partition using the fdisk command.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m shell -a 'df -h /dev/sda2' --become
```

Change the '/dev/sda2' with your own path.

2. RAM Memory Usage

Now check the RAM Memory usage on all servers using the 'free -m' command.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m shell -a 'free -m' --become
```

And you will be shown the result as below.

3. CPU Usage

Checking the CPU usage of all servers using the mpstat command.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m shell -a 'mpstat -P ALL' --become
```

The mpstat command is part of the 'sysstat' package.

4. Open Ports

Checking the open ports on all system using the netstat through the ad-hoc command.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m shell -a 'netstat -plntu' --become
```

5. Uptime

Now check the uptime of each server.
Syntax: ansible Host Group –m Module -a Arguments to the module

```
ansible hakase-testing -m shell -a 'uptime' --become
```

Some more examples

Ping test
Let’s start with a ping test and make sure we can connect to our servers.

Syntax: ansible Host Group –m Module -a Arguments to the module

```
$ ansible all -m ping

192.168.33.20 | SUCCESS => {
"changed": false,
"ping": "pong"
}
192.168.33.30 | SUCCESS => {
"changed": false,
"ping": "pong"
}
192.168.33.10 | SUCCESS => {
"changed": false,
"ping": "pong"
}
```

This ping is not the typical ICMP ping. It verifies the ability to login to the server and that a usable python is configured or not. We got the SUCCESS msg for all the servers, so we are good.

Parallel nature of Ansible

Let’s make sure the host names are set correct for all the servers.

Syntax: ansible Host Group –m Module -a Arguments to the module

```
$ ansible all -a "hostname"

192.168.33.30 | SUCCESS | rc=0 >>
db.dev

192.168.33.10 | SUCCESS | rc=0 >>
app1.dev

192.168.33.20 | SUCCESS | rc=0 >>
app2.dev
```

We can see the host names of the servers in random order. By default ansible runs commands in parallel using multiple process forks. The default value is 5. You can change this based upon server capacity

Syntax: ansible Host Group –m Module -a Arguments to the module

```
$ ansible all -a "hostname" -f 1
```

```
192.168.33.10 | SUCCESS | rc=0 >>
app1.dev

192.168.33.20 | SUCCESS | rc=0 >>
app2.dev

192.168.33.30 | SUCCESS | rc=0 >>
db.dev
```

'-f' denotes the number of forks (simultaneous process) to be used. Setting the value to 1 makes it to run in the sequence. It's fairly rare that you will ever need to do this, but it's much more frequent that you'll want to increase the value (like -f 10, or -f 25… depending on how much your system and network connection can handle)

Note: For the above example, we did not specify the command module as "-m command" since that is the default module.

Check Disk space

Now that we have made sure our host names are correct, lets check the disk space of each server.

Syntax: ansible Host Group –m Module -a Arguments to the module

```
$ ansible all -a 'df -h'
192.168.33.10 | SUCCESS | rc=0 >>
Filesystem                   Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVol00   38G  1.2G   37G   4% /
devtmpfs                      107M     0  107M   0% /dev
tmpfs                       119M     0  119M   0% /dev/shm
tmpfs                       119M  4.4M  115M   4% /run
tmpfs                       119M     0  119M   0% /sys/fs/cgroup
/dev/sda2                    1014M   89M  926M   9% /boot
tmpfs                        24M     0   24M   0% /run/user/1000

192.168.33.20 | SUCCESS | rc=0 >>
Filesystem                   Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVol00   38G  1.2G   37G   4% /
devtmpfs                      107M     0  107M   0% /dev
tmpfs                       119M     0  119M   0% /dev/shm
tmpfs                       119M  4.4M  115M   4% /run
tmpfs                       119M     0  119M   0% /sys/fs/cgroup
/dev/sda2                    1014M   89M  926M   9% /boot
tmpfs                        24M     0   24M   0% /run/user/1000

192.168.33.30 | SUCCESS | rc=0 >>
Filesystem                   Size  Used Avail Use% Mounted on
/dev/mapper/VolGroup00-LogVol00   38G  1.2G   37G   4% /
```

```
devtmpfs                  107M     0  107M   0% /dev
tmpfs                    119M     0  119M   0% /dev/shm
tmpfs                    119M  4.4M  115M   4% /run
tmpfs                    119M     0  119M   0% /sys/fs/cgroup
/dev/sda2               1014M   89M  926M   9% /boot
tmpfs                     24M     0   24M   0% /run/user/1000
```

Check for server memory

Syntax: ansible Host Group –m Module -a Arguments to the module

```
$ ansible all -a 'free -m'

192.168.33.30 | SUCCESS | rc=0 >>
              total        used        free      shared  buff/cache   available
Mem:            236          73          20           4         143         129
Swap:          1535           0        1535

192.168.33.10 | SUCCESS | rc=0 >>
              total        used        free      shared  buff/cache   available
Mem:            236          73          19           4         143         129
Swap:          1535           0        1535

192.168.33.20 | SUCCESS | rc=0 >>
              total        used        free      shared  buff/cache   available
Mem:            236          73          20           4         143         129
Swap:          1535           0        1535
```

Ensure date and time of the server are in sync

Syntax: ansible Host Group –m Module -a Arguments to the module

```
$ ansible all -a 'date'

192.168.33.20 | SUCCESS | rc=0 >>
Mon Feb 20 14:50:31 UTC 2017

192.168.33.10 | SUCCESS | rc=0 >>
Mon Feb 20 14:50:31 UTC 2017

192.168.33.30 | SUCCESS | rc=0 >>
Mon Feb 20 14:50:31 UTC 2017
```

So far we have just checked the status of the servers. Let us make some changes in the servers using various Ansible modules