ANSIBLE

* Ansible is an configuration management and provisioning tool. It is created using python.
* It is an automation tool, which automates all the actions in multiple servers at a time by just writing a script and pushing it to all the server from central server.
* For ex: if you have 100 servers and you want to install an application in all 100 servers. You don't need to login to all servers and do that work, just create an yaml script, run the script in the ansible server and it will do the remaining in all 100 servers.
* Python is mandatory for ansible v2.6.
* To install ansible, first we have to install **epel-release**.
* To Install the ansible….**yum install ansible**
* To check whether ansible is installed (or) not…..**ansible --version**.
* Main configuration file = **/etc/ansible/ansible.cfg**.

**FEAUTURES**

* **Agent less** = No need of creating an agent in client machines like in other CM tools chef, puppet. You just have install ansible in server and we are good to go.
* **SSH** = It uses ssh connections. You don't need to install any extra software to connect to client machines. So, Make sure ssh is working properly in all machines.
* **PUSH** = It uses push based architecture for sending configurations. Write the code in yaml script and execute the script. Ansible will push and executes it in all nodes.

**HOST INVENTORY**

* It contains list of your hosts and host groups.
* It will have all your host ip's. we can also create groups for the specific hosts like web servers are one group, db servers are one group.
* Befor adding hosts to inventory, make sure you have made an successful ssh connection with all the nodes to maintain without intereption.
* If you get an error like this while making ssh connection with node.

**“1 key(s) remain to be installed -- if you are prompted now it is to install the new keys”**

* Just set “**password** **authentication yes”** in /**etc/ssh/sshd\_config** file in **node** machine and restart the node ssh server.
* To add hosts, just write down the ip of hosts.
* To create a group in inventory, write grpname in square bracets and write the hosts ip's down the groupname.

**[host group name]**

**Host ip 1**

**Host ip 2**

**Host ip 3**

* So, whenever you want to make configuration changes just for db server, mention the dbserver grpname while executing the playbook and it will change only that specific grp.
* default location of inventory = **/etc/ansible/hosts**.
* To see how many hosts we configured in inventory = **ansible all --list-hosts**.
* After adding host inventory try to ping whether we configured correctly (or) not = **ansible test –m ping** (test=grp name).
* You can use different paths for inventory. But while executing playbooks you have to mention that path with **–i** option.

**ansible-playbook –I inventory-path file.yml**

**DYNAMIC INVENTORY**

* Normally, we use default ansible static host inventory **(/etc/ansible/hosts)** for nodes.
* If the nodes have static ip, there is no problem. But, if the nodes have dynamic ip, the ip will change after every reboot. Updating the hosts file with new ip’s after every reboot will be a headache.
* For this we use dynamic inventory, We have to write python scripts for dynamic inventory and the scripts will keep the hosts updated.
* In aws, we don’t need to write any python scripts for dynamic inventory. By default, ansible provides you scripts for dynamic inventory in aws.
* **ec2.py** = It is a script written in boto ec2 library. It will query your aws account for running ec2 instances.
* **ec2.ini** = It is the configuration for the ec2.py and it can be used to limit the reach of ansible. You can specify tags, regions etc.
* You can download these scripts from the internet and copy those files to **/etc/ansible**.
* To work with these scripts, you should have installed ansible with **pip**.
* You have to install boto (python interface with aws) to work with ansible and aws.
* Install **python** and **boto** (python interface with aws).

**pip install boto.**

* To link boto with ansible we have to set this in inventory file.
* Create a file called **~/.boto** and configure your credentials like this

**[credentials]**

**aws\_access\_key\_id = AKIAJNLYRAN2L2LFKSXA**

**aws\_secret\_access\_key = HbVeN6YZsuMeVpTDw0VifPLh75jCuEmC7CH0orx1**

* Go to /etc/ansible/

**./ec2.py** **--list** = It will show all your running instances in aws.

**ansible instance-id –m ping** = Ping instances with instance id.

**ansible tag\_Name\_ec2tag –m ping** = To ping instance with tag name.

* Mention the ec2.py with playbook command while executing any playbook.

**ansible-playbook –i ec2.py file.yml.**

* In the playbook, you can specify hosts as **all** (or) you can specify **instance** **tags**.

**MODULES**

* Modules (tasks or library plug-ins) are the ones which actually executed inside the playbooks.
* A playbook contains play, a play contains tasks, a task contains modules and whenever you run the playbook the modules will get executed.

**Ex : apt module, yum module, service module, copy module, fetch module** etc**.**

* If **SElinux** is enabled in your server and nodes, you must install **libselinux-python in all nodes and server** to use **file/copy/template** related functions in ansible.

**AD-HOC CMDS**

* These are simple one-line cmds in order to perform a quick task. If you want to do a task and you don't want to write a playbook, you can use these ad-hoc commands.
* You have to use the ansible commands with modules which are called ad-hoc commands.

Ex : **ansible all -s -m shell –a “uptime”**  = it will show the uptime of all hosts.

**ansible all –s –m shell –a “tail –f /var/log/access.log”** = To see logs.

**s =** run cmd using sudo**.**

**a =** pass arguments to module**.**

**C =** Check**.**

**m =** modulename**.**

* There are so many ad-hoc commands like these for daily purposes without writing playbooks.
* **ansible all –m setup** = To see nodes info.

**COPY**

* **ansible all –m copy –a “src=/path dest=/path-to-store”** = To copy file from server to hosts.

**FETCH**

* **ansible all –s –m fetch –a “src=/path dest=/path”** = To copy file from hosts to server.

**FILE**

* **ansible all –m file –a “dest=/file-path mode=0644 state=touch”** = To create an empty file.
* **ansible all –m file –a “dest=path file mode=0666 state=directory”** = To create dir.
* **ansible all –m file –a “dest=/file-path state=absent”** = To delete a file (or) dir.
* **ansible all –m file –a “src=/path dest=/path user=root group=root state=link”** = To create symlink.
* **ansible all –s –m file –a “dest=/path mode=0666 user=name group=name” =** To change permission.

**CRON**

* **ansible all –m cron –a “name=’/cron-name’ hour=4 job=/script-2-exe”** = To execute cron job.
* **ansible all –m cron –a “name=’/cron-name’ state=absent”** = To remove cron job**.**

**YUM**

* **ansible all –m yum –a “name=pkg state=installed”** = To install packages.
* **ansible all –m yum –a “name=pkg state=latest”** = To install a pkg. If the pkg is already installed, it will update the pkg to latest version.
* **ansible all –m yum –a “name=pkg-name state=absent”** = To remove a pkg.
* **ansible all –m yum –a “name=’\*’ state=latest” =** To update all pkgs.
* **ansible all –m yum –a “name=url state=present”** = To install pkg from an url.
* **Ansible all –m get\_url -a “name=url=link dest=/path ”** to download links

**SERVICE**

* **ansible all -m -s service –a “name=httpd state=started”** = To start a service.
* **ansible all -m -s service –a “name=httpd state=restarted”** = To restart the service.
* **ansible all -m -s service –a “name=httpd state=stopped”** = To stop the service.

Service modules= Started, stopped, restarted, reloaded.

**USER**

* **ansible all –m user “name=user-name password=password group=grp-name”** = To create an user in all hosts. You have to enter crypted password.
* **mkpasswd –method=sha-512 =** To generate crypted password**.**
* **ansible all –m user –a “name=user-name state=absent”** = To delete an user from all servers.

**PLAYBOOKS**

* Playbooks define your workflow. Playbooks are set of instructions that you send to run on hosts machines. If we want to configure something in nodes, we write tasks in playbooks, it actually gets executed in same order in all nodes as we written in playbooks.
* Playbooks are easy to write. These are written in yaml and should end with **.yml** format.
* Playbooks are divided into three sections.
* 1. **Targets (or) hosts =** servers where this playbooks should be executed.
* 2. **variables** (not mandatory) = You can define your variables.
* 3. **Tasks** = tasks to perform on hosts (or) targets.
* 4. **Handlers** = these are just like tasks, but only run when called bay another task.

EX:

**---**

**- hosts: test**

**sudo: yes**

**tasks:**

**- name: install ftp**

**yum: name=vsftpd state=installed**

**notify:**

**- start ftp**

**handlers:**

**- name: start ftp**

**service: name=vsftpd state=started**

* save and quit the file. Now, run this playbook.....**ansible-playbook file.yml** ..DONE...
* **ansible-playbook file.yml --syntax-check** = To check syntax errors in playbook.
* **ansible-playbook –v file.yml** = To see output of playbook.
* **ansible-playbook –v file.yml --step** = It will ask (y,n) to perform this action (or) not.
* **ansible-playbook –vvvv file.yml =** Shows you clear log output of tasks.
* Every lines we write after hosts line should be under it.
* Handlers are similar to tasks, but run only if we set nofity directive.
* You can limit hosts where ansible playbook should run with **“--limit”** option.
* **ansible-playbook file.yml --limit “host1”** - To limit tasks only to one host in group.
* **ansible-playbook file.yml --limit “host1,host2”** – To limit tasks to couple of hosts in group.
* If you are executing a playbook in ten nodes. By default, ansible uses linear strategy where it executes the playbook on a host and waits for the result, until then It won’t execute task on other hosts. This is not good for some situations, instead of waiting for the task to finish on host, we can run tasks on other hosts too with **strategy** **free**.
* Use strategy free in playbook after hosts line.

**strategy: free**

* Sometimes you want to skip a particular step on particular host. Like you don’t want to install a package on one of your host in your group. At that time, you can use **when**, which contains raw jinja2 expression without curly braces.

**when: ansible\_os\_family == “debain”.**

**when: (ansible\_distribution == “centos” and ansible\_distribution\_major\_version == “6”)**

**when: (ansible\_distribution == “debain” and ansible\_distribution\_major\_version == “7”)**

**ROLES**

* Roles are like predefined files, where we define all our requirements in roles and use those roles in playbooks.
* We can share & reuse these roles in anywhere. Roles are located at **/etc/ansible/roles** dir.
* You can create roles by using **ansible**-**galaxy** command. It will create a directory structure where you can define all your requirements.
* A role directory structure contains of **defaults**, **files**, **vars**, **handlers**, **meta**, **tests**, **templates** and **tasks**.
* **Defaults** = Conatins variables. But have lower priority.
* **Files =** Contains regular files where you need to copy to nodes**.**
* **Handlers =** Contains targets for notify directives and almost associated with the services.
* **Meta** = consists of atributes such as about the role, author of the role, platform, dependencies etc.
* **Tasks** = Contains all the actions that should be done, when you are using this role like install package, remove package etc.
* **Templates** = to change the contents of the file which has static and dynamic content. These are similar to files but it supports modification (dynamic files) as they are being provisioned to nodes. Modifications are done through jinja2 templating language.

For ex: configuration files.

* **Vars** = variables stores in the vars have higher priority, which are hard to override, whereas variables stores in defaults have low priority.
* Every directory contains its own file called main.yml. You have to write all the data in this file respective to their directories.
* **ansible-galaxy init apache**= To create a role for apache. If you want to create a role for nginx, type nginx in the place of apache.
* Go to **/etc/ansible/roles/apache/tasks**.
* Edit the **main**.**yml** file and add these following lines to install apache and copy the index.html file to nodes.

**- name: install apache**

**yum: name=httpd state=installed**

**- name: copy index file**

**copy: src=index.html dest=/var/www/html**

**notify:**

**- start apache**

* Go to **/etc/ansible/roles/apache/handlers**.
* Edit the **main**.**yml** file to give targets for notify directories.

**- name: start apache**

**service: name=httpd state=started**

* The Handlers and notify name should be the same.
* To copy files to node machines, the files should be there at files directory.
* Copy the index.html file to **/etc/ansible/roles/apache/files** dir.
* Whatever the files you want to copy to nodes, first you should mention them in tasks dir with copy module and those files should be present in the files dir.
* After completing these requirements, create a playbook for apache to install and copy the files. You just have to mention the hosts and role name in the playbook. It will take all the files from that role.

**---**

**- hosts: all**

**Roles:**

**- role-name**

* To copy dynamic content like configuration files(for ex:httpd.conf), we can use template module.
* In tasks dir. Add the following lines above notify line.

**- name: copy dynamic files**

**Template: src=httpd.conf dest=/etc/httpd/conf/httpd.conf**

* It will copy the data which is not there in destination machine not the entire file.
* Templates are based on **jinja2** templating language.
* You can define multiple no of roles in a single playbook. It will perform all the tasks.
* You can use include module in tasks, when you have created multiple yml files separately.

**- include: file.yml**

**- include: file2.yml**

* You can also specify tags for include modules, while executing playbooks, to run only specified tasks.

**- include file.yml tags:web**

**- include: file2.yml tags:db**

* While executing playbook, specify the tag to run only those tasks under that tag.
* ansible-playbook site.yml --tags “web”
* The above command will execute the tasks only which are under the web tag.

**LAUNCH EC2 WITH ANSIBLE**

* You can install ec2 instances with ansible by using ec2 module.
* You have to install **pip**, **boto** and configure your aws credentials in ec2.
* **yum install python-pip** = To install pip.
* You can also install pip with url.
* **wget** [**https://bootstrap.pypa.io/get-pip.py**](https://bootstrap.pypa.io/get-pip.py) **=** To download pip**.**
* **python get-pip.py** = To install pip (from downloaded file).
* **pip install boto** (or) **boto3** = to install boto.
* If boto doesn’t work install boto3 with pip.
* To link boto with ansible we have to set this in inventory file.
* Set inventory file = **localhost ansible\_connection=local ansible\_python\_interpreter=python**
* Create a file called **~/.boto** and configure your credentials like this

**[credentials]**

**aws\_access\_key\_id = AKIAJNLYRAN2L2LFKSXA**

**aws\_secret\_access\_key = HbVeN6YZsuMeVpTDw0VifPLh75jCuEmC7CH0orx1**

* Create an ec2 role and define all the variables and tasks and execute it to launch instances.
* You need ami, security group, subnet ID, keypair, region, instance type and other configurations as you want. It will create ec2 instance with the configurations you specified.
* If you are using a playbook to launch ec2 instances, you have to specify variables in playbook separately. You can’t use raw code to launch ec2 instances.
* Download e2.py and ec2.ini files to communicate and manage the nodes.
* Ex: playbook to create ec2

**---**

**- name: Create a new Demo EC2 instance**

**hosts: localhost**

**gather\_facts: False**

**vars:**

**region: ap-south-1**

**instance\_type: t2.micro**

**ami: ami-e60e5a89**

**keypair: docker**

**tasks:**

**- name: Create an ec2 instance**

**ec2:**

**key\_name: "{{ keypair }}"**

**group: RED # security group name**

**instance\_type: "{{ instance\_type}}"**

**image: "{{ ami }}"**

**wait: true**

**region: "{{ region }}"**

**count: 1**

**count\_tag:**

**Name: Demo**

**instance\_tags:**

**Name: Demo**

**vpc\_subnet\_id: subnet-cd1ec9a5**

**assign\_public\_ip: yes**

**register: ec2**

* The above playbook will create one ec2 instance with the configurations we mentioned in playbook.
* You can also use ansible roles to launch instances. The roles are easy to configure.
* Create an ec2 role with ansible galaxy command and configure the variables and tasks and create an playbook with that role to launch ec2 instances.
* Ansible-galaxy init ec2 = To create an ec2 role.
* Go to vars dir and edit the main.yml file, mention all the variables that you need to launch ec2.

**ec2:**

**region: ap-south-1**

**zone: ap-south-1a**

**keypair: keypairname**

**image: ami-id**

**instance\_type: t2.micro**

**group: security group for ec2**

**vpc\_subnet\_id: subnet to launch instance in.**

**public\_ip: yes**

**count: 1**

**instance\_tags:**

**Name: name for ec2**

**volumes:**

**- device\_name: /dev/xvda**

**volume\_type: standard**

**volume\_size: 10**

* The ec2 at the top of the script is nothing but the ec2 module we are using to launch ec2 instances with ansible.
* You can mention infinite number of variables in the vars dir. After mentioning variables add them to tasks to execute them.
* Go to tasks dir and edit the main.yml file and mention all these variables in that file.

**- name: launch ec2**

**ec2:**

**region: “{{ ec2.region }}”**

**zone: “{{ ec2.zone }}”**

**image: “{{ ec2.image }}”**

**keypair: “{{ ec2.keypair }}”**

**group: “{{ ec2.sg }}”**

**vpc\_subnet\_id: “{{ ec2.vpc\_subnet\_id }}”**

**assign\_public\_ip: “{{ ec2.public\_ip }}”**

**instance\_tags: “{{ ec2.instance\_tags }}”**

**wait: true**

* Create a playbook and specify this ec2 role to launch ec2 instances.

**---**

**- hosts: localhost**

**connection: local**

**roles:**

**- ec2**

* Once you execute this playbook, it will launch ec2 instances with specified configurations.
* In the above example, we wrote the variables and tasks separately and mentioned the variables in tasks to perform actions.
* The values in the bracets represents the values in vars directory. As here in tasks dir, we mentioned the variables along with role (ec2).
* While executing the playbook, ansible goes to that role we specified in playbook, search for tasks dir to execute the actions and pulls the data from vars dir, if we mentioned any variables.

**VAULT**

* Ansible vault can encrypt anything inside an yaml file, with password of your choice.
* It can be ssh keys, ssl certificates, api tokens etc.
* You can’t see (or) edit the file once it is encrypted with vault. You can’t even run the playbook after encrypting, which gives you an error.
* With the help of ansible**-**vault command, you can encrypt and decrypt playbooks.
* The most common encrypted files are variable files which have sensitive data.

Ex: defaults/main.yml, vars/main.yml

* The encrypted files can be distributed (or) can place in version control. The file will be in encrypted mode even in version control.
* Whenever you are using an vault command, it will prompt you to give password for **encrypt**, **decrypt**, **create** and **edit** actions.
* **ansible-vault encrypt file.yml** = To encrypt an existing file.
* **ansible-vault decrypt file.yml** = To decrypt an file.
* **ansible-vault create file.yml** = To create a new file with encryption.
* **ansble-vault edit file.yml** = To edit an encrypted file.
* **ansible-playbook file.yml --ask-vault-pass** = To run an encrypted playbook. It will ask you vault password to execute the playbook.
* **ansible-vault rekey file.yml** = To change vault password. It will prompt you to type old password and give a new password to vault.

**PARALLELISM:**

**SERIAL:**

* When we execute a playbook ansible execute first task in all machines and continues to 2nd one, which will take more time to finish playbook and not good for deployments because it requires downtime.
* We can divide the servers for deployment with the help of “**serial**” keyword.
* You can set **number** for serial. So that ansible would complete the play before going to next hosts.

**serial: 3**

* In the above example, we specified serial as 3, it will divide 3 hosts per batch and completes the play in 3 hosts and jump into next 3 hosts.
* You can specify **percentage** also in serial. The percentage will be applied to total number of hosts in a play inorder to determine number of hosts per batch.

**serial: “50%”**

* In the above example, we have set 50%. So, ansible will divide hosts into batches by this percentage and run play in batches serial order. If the number of hosts does not divide equally the last batch will contain a remainder.
* We can also specify batch sizes, so it will divide the hosts based on the number.

**serial:**

**- 1**

**- 5**

**- 10**

* In the above example, we specified that
* **First batch will have 1 host**
* **2nd batch will have 5 hosts**
* **3rd and following batches will have 10 hosts until all hosts are finished.**
* We can also specify batch sizes as percentage.

**serial:**

**- “30”**

**- “50”**

**- “100”**

* You can mix and match the values.

**serial**

**- 3**

**- “40”**

* In some situations, we have to abort the play because of hosts failures. For this, we can set a threshold with serial parameter. Where ansible will abort the play if the certain threshold met.

**max\_fail\_percentage: 30**

**serial: 10**

* In the above example, if 30% of failures happen in 10 hosts, the play will be aborted.

**FORKS:**

* Forking means ansible executes first task parallelly on some hosts and waits for it to finish and go on to another hosts. By default **forks = 5**.
* But, If you have 100 hosts, it will execute one task on 5 hosts at a time. Which will take more time to complete the task.
* You can specify forks value in ansible configuration file (or) you can specify while excuting playbook.

**Edit /etc/ansible/ansible.cfg**

**Uncomment forks line and specify the value.**

* To specify while executing a playbook.

**ansible-playbook web.yml --forks=value**

* Based on the forks value, ansible executes the task on hosts.

Yum

Service file fetch

Copy

User

Unarchive

Shell

Command

Directory