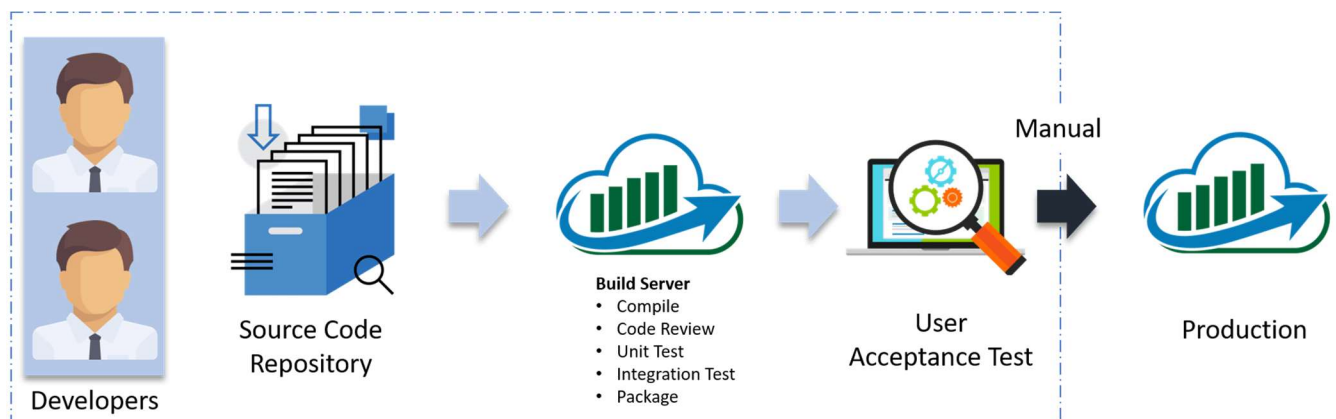


Q1. What is CI/CD?

Continuous Integration is a software development practice where members of a team integrate their work frequently, usually, each person integrates at least daily leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly.

Continuous Delivery is a process where you build software in such a way that it can be released to production at any time. Consider the diagram below:



Let me explain the above diagram:

- Automated build scripts will detect changes in Source Code Management (SCM) like Git.
- Once the change is detected, source code would be deployed to a dedicated build server to make sure build is not failing and all test classes and integration tests are running fine.
- Then, the build application is deployed on the test servers (pre-production servers) for User Acceptance Test (UAT).
- Finally, the application is manually deployed on the production servers for release.

Q2. What is Configuration Management and how does it help an organization?

Configuration Management is the practice of handling updates and changes systematically so that a system maintains its integrity over time. Configuration Management (CM) keeps a track of all the updates that are needed in a system and it ensures that the current design and build state of the system is up to date and functioning correctly.

Configuration Management can help an organization by overcoming the following challenges:

- Finding out what changes need to be implemented when user requirements change.
- Redoing and updating an implementation due to change in the requirements since the last implementation.
- Reverting to an older version of the component because the latest version is flawed.
- Replacing the wrong component because you couldn't accurately determine which component needed replacing.

To better understand this consider the NYSE example:

The New York Stock Exchange (NYSE) encountered a glitch in their software which prevented them from trading stocks for approx 90 minutes. On the night before, new software was installed on 8 of its 20 trading terminals. Unfortunately, the software failed to operate properly on the 8 terminals.

Therefore, by using Configuration Management tools such as Ansible and Puppet, they reverted back to the old software. Had they not implemented CM, they would've taken a much longer time to fix the issue which would lead to a much bigger loss.

Q3. What is Ansible and what makes it stand out from the rest of the Configuration Management tools?

[Ansible](#) is an open source IT Configuration Management, Deployment & Orchestration tool. It aims to provide large productivity gains to a wide variety of automation challenges.

Here's a list of features that makes Ansible such an effective Configuration Management and Automation tool:

1. Simple: Uses a simple syntax written in YAML called playbooks.
2. Agentless: No agents/software or additional firewall ports that you need to install on the client systems or hosts which you want to automate.
3. Powerful and Flexible: Ansible's capabilities allow you to orchestrate the entire application environment regardless of where it is deployed.
4. Efficient: Ansible introduces modules as basic building blocks for your software. So, you can even customize it as per your needs.

Q4. How does Ansible work?

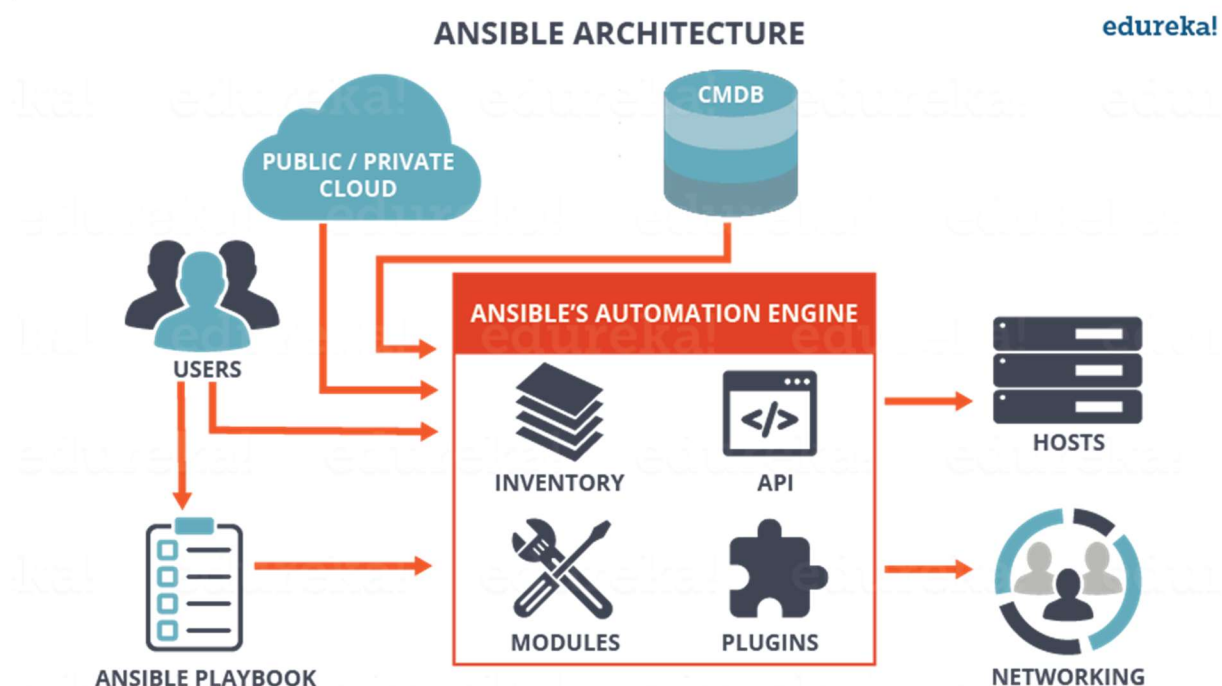
Ansible, unlike other configuration management tools, is categorized into two types of servers – Controlling machines and Nodes. Controlling machine is where Ansible is installed and nodes are the ones that are managed by the controlling machines through SSH. There is an inventory file in the controlling machine that holds the location of the node systems. Ansible deploys modules on the node systems by running the playbook on the controlling machine. Ansible is agentless, that means there is no need to have a third party tool to make a connection between one node and the other.

Q5. How is Ansible different from Puppet?

Metrics	Ansible	Puppet
1. Availability	<ul style="list-style-type: none">• Highly available	<ul style="list-style-type: none">• Highly available
2. Ease of set up	<ul style="list-style-type: none">• Easy	<ul style="list-style-type: none">• Comparatively hard to set up
3. Management	<ul style="list-style-type: none">• Easy management	
4. Scalability	<ul style="list-style-type: none">• Highly scalable	<ul style="list-style-type: none">• Not very easy
5. Configuration language	<ul style="list-style-type: none">• YAML(Python)	<ul style="list-style-type: none">• Highly scalable• DSL(PuppetDSL)
6. Interoperability	<ul style="list-style-type: none">• High	<ul style="list-style-type: none">• High
7. Pricing nodes	<ul style="list-style-type: none">• \$10,000	<ul style="list-style-type: none">• \$11200-\$19900

Q6. What are the different components of ansible? Explain Ansible architecture.

The below diagram depicts the Ansible architecture:



Ansible Architecture – Ansible Interview Questions

The main component of Ansible is the Ansible automation engine. This engine directly interacts with various cloud services, Configuration Management Database (CMDB) and different users who write various playbooks to execute the Ansible Automation engine.

The Ansible Automation engine consists of the following components:

Inventories: These are a list of nodes containing their respective IP addresses, servers, databases, etc. which needs to be managed.

APIs: Just like any other API, the Ansible APIs are used for commuting various Cloud services, public or private services.

Modules: The modules are used to manage system resources, packages, libraries, files, etc. Ansible modules can be used to automate a wide range of tasks. Ansible provides around 450 modules that automate nearly every part of your environment.

Plugins: If you want to execute Ansible tasks as a job, Ansible Plugins can be used. They simplify the execution of a task by building a job like an environment that basically contains pieces of code corresponding to some specific functionality. There are 100s of Plugins provided by Ansible. An example is the Action plugin, which acts as front ends to modules and can execute tasks on the controller before calling the modules themselves.

Networking: Ansible can also be used to automate different networks and services. It can do this by creating a playbook or an Ansible role that easily spans different network hardware.

Hosts: The Ansible Hosts/ Node systems are machines (Linux, Windows, etc) that are getting automated.

Playbooks: Playbooks are simple code files which describe the tasks that need to be executed. The Playbooks are written in YAML format. They can be used to automate tasks, declare configurations, etc.

CMDB: It is a database that acts as a storehouse for various IT installations. It holds data about various IT assets (also known as configuration items (CI)) and describes the relationships between such assets.

Cloud: It is a network of remote servers hosted on the Internet to store, manage, and process data, rather than a local server.

Q7. What are Ansible Server requirements?

If you are a windows user then you need to have a virtual machine in which Linux should be installed. It requires Python 2.6 version or higher. you fulfill these requirements and you're good to go!

Q8. How would you install Ansible on a CentOS system?

This can be done in two simple steps:

Step 1: Set up EPEL Repository

EPEL (Extra Packages for Enterprise Linux) is an open source and free community-based repository project from Fedora team which provides high-quality add-on software packages for Linux distribution including RHEL (Red Hat Enterprise Linux), CentOS, and Scientific Linux.

The Ansible package is not available in the default yum repositories, so we will enable EPEL repository by using the below command:

```
sudo rpm -ivh http://dl.fedoraproject.org/pub/epel/6/i386/epel-release-6-8.noarch.rpm
```

This will download all the necessary packages which will be required to install Ansible.

Step 2: Install Ansible

Now that your EPEL repository has been added, all you have to do now is install Ansible using the command below:

```
yum install ansible -y
```

That's all! It's a two-step process that barely takes a minute!

If you wish to check the version of Ansible installed on your system, use the command below:

```
ansible --version
```

Q9. Explain a few of the basic terminologies or concepts in Ansible.

Few of the basic terms that are commonly used while operating on Ansible are:

Controller Machine: The Controller machine is responsible for provisioning the servers that are being managed. It is the machine where Ansible is installed.

Inventory: An inventory is an initialization file that has details about the different servers you are managing.

Playbook: It is a code file written in the YAML format. A playbook basically contains the tasks that need to be executed or automated.

Task: Each task represents a single procedure that needs to be executed, e.g. Install a library.

Module: A module is a set of tasks that can be executed. Ansible has 100s of built-in modules, but you can also create custom ones.

Role: An Ansible role is a pre-defined way for organizing playbooks and other files in order to facilitate sharing and reusing portions of provisioning.

Play: A task executed from start to finish or the execution of a playbook is called a play.

Facts: Facts are global variables that store details about the system, like network interfaces or operating system.

Handlers: Are used to trigger the status of a service, such as restarting or stopping a service.

Q10. Explain the concept behind Infrastructure as Code (IaC).

Infrastructure as Code (IaC) is a process for managing and operating data servers, storage systems, system configurations, and network infrastructure.

In traditional configuration management practices, each minute configuration change required manual action by system administrators and the IT support team. But with IaC, all the configuration details are managed and stored in a standardized file system, wherein the system automatically manages infrastructure changes and deals with system configurations.

Therefore, we do not require most of the manual effort since everything is managed and automated by following the IaC approach. Tools such as Ansible can be used to implement IaC approach.

Q11. Compare Ansible with Chef.

Metrics	Ansible	Chef
1. Availability	• Highly available	• Highly available
2. Ease of set up	• Easy	• Not very easy
3. Management	• Easy management	• Not very easy
4. Scalability	• Highly scalable	• Highly scalable
5. Configuration language	• YAML(Python)	• DSL(Ruby)
6. Interoperability	• High	• High
7. Pricing nodes	• \$10,000	• \$13700

Q12. What is Ansible Galaxy?

Galaxy is a website that lets Ansible users share their roles and modules. The Ansible Galaxy command line tool comes packed with Ansible, and it can be used to install roles from Galaxy or directly from a Source Control Management system such as Git. It can also be used to build new roles, remove existing ones and perform tasks on the Galaxy website.

You can use the below command to download roles from the Galaxy website:

```
$ansible-galaxy install username.role_name
```

Q13. What are Ad-hoc commands? Give an example.

Ad-hoc commands are simple one-line commands used to perform a certain task. You can think of Adhoc commands as an alternative to writing playbooks. An example of an Adhoc command is as follows:

```
ansible host -m netcaler -a "nsc_host=nsc.example.com user=apiuser password=apipass"
```

The above Adhoc command accesses the netcaler module to disable the server.

Q14. What are the variables in Ansible?

Variables in Ansible are very similar to variables in any programming language. Just like any other variable, an Ansible variable is assigned a value which is used in computing playbooks. You can also use conditions around the variables. Here's an example:

```
1 - hosts: your hosts
2   vars:
3     port_Tomcat : 8080
```

Here, we've defined a variable called port_Tomcat and assigned the port number 8080 to it. Such a variable can be used in the Ansible Playbook.

Q15. What is the difference between a variable name and an environment variable?

Variable name	Environment variable
<ul style="list-style-type: none">• You need to add strings to create variable names• You can easily create multiple variable names by adding strings• We use the ipv4 address for variable names	<ul style="list-style-type: none">• You need existing variables to access environment variables.• To create environment variables we must refer advanced Ansible playbook• We use {{ ansible_env.SOME_VARIABLE }} for remote environment variables.

Q16. What are the Ansible Modules? Explain the different types.

Ansible modules are a small set of programs that perform a specific task. Modules can be used to automate a wide range of tasks. Modules in Ansible are considered to be idempotent or in other words, making multiple identical requests has the same effect as making a single request.

There are 2 types of modules in Ansible:

1. Core modules
2. Extras modules

Core Modules

These are modules that the core Ansible team maintains and will always ship with Ansible itself. They will also receive a slightly higher priority for all requests than those in the “extras” repos. The source of these modules is hosted by Ansible on GitHub in the Ansible-modules-core.

Extras Modules

These modules are currently shipped with Ansible but might be shipped separately in the future. They are also mostly maintained by the Ansible Community. Non-core modules are still fully usable but may receive slightly lower response rates for issues and pull requests.

Popular “extras” modules may be promoted to core modules over time. The source for these modules is hosted by Ansible on GitHub in the Ansible-modules-extras.

Q17. What is Ansible Task?

Ansible Tasks allow you to break up bits of configuration policy into smaller files. These are blocks of code that can be used to automate any process. For example, if you wish to install a package or update a software, you can follow the below code snippet:

```
Install <package_name>, update <software_name>
```

Q18. Can you explain what are playbooks in Ansible? Explain with some examples.

Playbooks in Ansible are written in the YAML format. It is a human-readable data serialization language. It is commonly used for configuration files. It can also be used in many applications where data is being stored.

For Ansible, nearly every YAML file starts with a list. Each item in the list is a list of key/value pairs, commonly called a “hash” or a “dictionary”. So, we need to know how to write lists and dictionaries in YAML.

All members of a list are lines beginning at the same indentation level starting with a “- ” (dash and space). More complicated data structures are possible, such as lists of dictionaries or mixed dictionaries whose values are lists or a mix of both.

For example, if you want a playbook containing details about the USA:


```
1 -USA
2 -continent: North America
3 -capital: Washington DC<a
4 name="AnsibleIntermediateLevelInterviewQuestions"></a>
   -population: 319 million
```

Now that you know the basic level questions, let's take a look at a little more advanced Ansible Interview Questions.

Intermediate Level Ansible Interview Questions

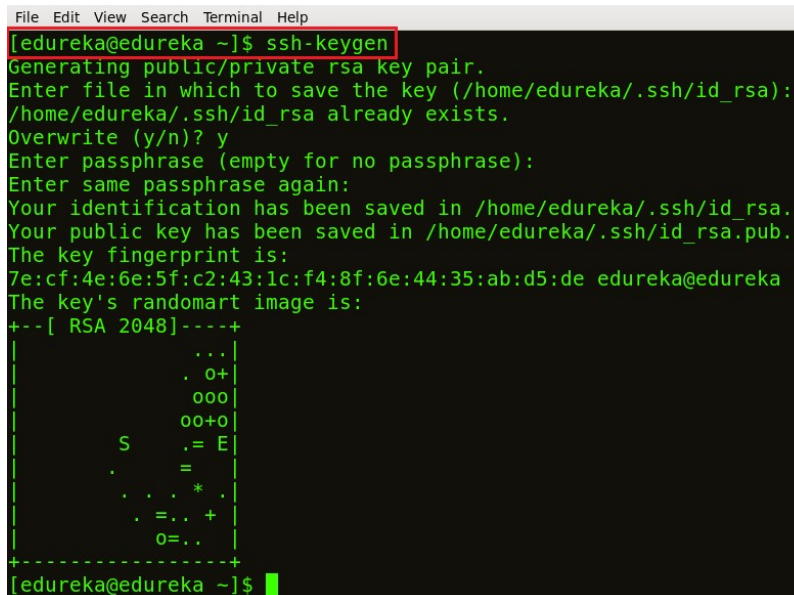
Once you've aced the basic conceptual questions, the interviewer will increase the difficulty level. So let's move on to the next section of this Ansible Interview Questions article. This section talks about the commands that are very common amongst docker users.

Q19. Can you write a simple playbook to install Nginx on a host machine?

Step 1: Generate a public SSH key and by using SSH connect to your host.

Follow the command below:

```
$ ssh-keygen
```



```
File Edit View Search Terminal Help
[edureka@edureka ~]$ ssh-keygen
Generating public/private rsa key pair.
Enter file in which to save the key (/home/edureka/.ssh/id_rsa):
/home/edureka/.ssh/id_rsa already exists.
Overwrite (y/n)? y
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/edureka/.ssh/id_rsa.
Your public key has been saved in /home/edureka/.ssh/id_rsa.pub.
The key fingerprint is:
7e:cf:4e:6e:5f:c2:43:1c:f4:8f:6e:44:35:ab:d5:de edureka@edureka
The key's randomart image is:
+--[ RSA 2048 ]-----+
|           .+..      |
|      .   .o+       |
|     .oo            |
|    .oo+o          |
|   S  .+= E        |
|      =            |
|     . . * .       |
|    . =. . +       |
|   o=..            |
+-----+
[edureka@edureka ~]$
```

Ansible Playbook – Ansible Interview Questions

As shown above, a public SSH key is generated.

Step 2: Next, copy the public SSH key on your hosts. Follow the below command to do it:

ssh-copy-id -i root@IP address of your host

```
[root@edureka edureka]# ssh-copy-id -i root@10.0.2.15
The authenticity of host '10.0.2.15 (10.0.2.15)' can't be es
RSA key fingerprint is 2d:af:5f:52:24:59:05:f0:1e:12:60:71:c
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added '10.0.2.15' (RSA) to the list of
root@10.0.2.15's password:
Now try logging into the machine, with "ssh 'root@10.0.2.15'
.ssh/authorized_keys
to make sure we haven't added extra keys that you weren't ex
[root@edureka edureka]#
```

Ansible Playbook – Ansible Interview Questions

Step 3: List the IP addresses of your hosts/nodes in your inventory.

Follow the below command:

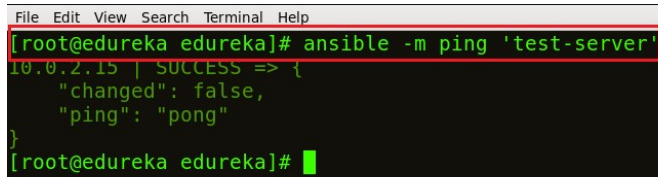
vi /etc/ansible/hosts

```
## db-[99:101]-node.example.com
[test-server]
172.17.0.2
INSERT
```

Ansible Playbook – Ansible Interview Questions

Once you run the command, the vi editor will open where you can list down the IP addresses of your hosts. This is now your inventory.

Step 4: To check if the connection has been established, let's ping:

A terminal window with a menu bar (File, Edit, View, Search, Terminal, Help) and a dark background. The prompt is [root@edureka edureka]#. The command 'ansible -m ping 'test-server'' is entered and highlighted with a red box. The output shows a successful ping to 10.0.2.15, returning 'pong'.

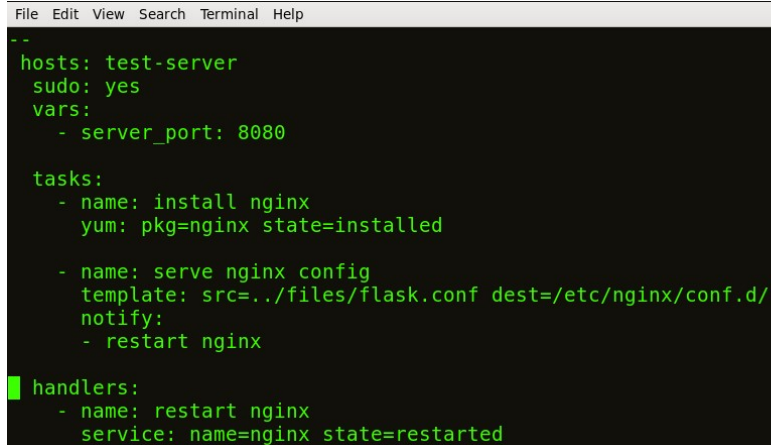
```
[root@edureka edureka]# ansible -m ping 'test-server'
10.0.2.15 | SUCCESS => {
  "changed": false,
  "ping": "pong"
}
[root@edureka edureka]#
```

Ansible Playbook – Ansible Interview Questions

The above image shows that a connection has been made between the control machine and the host.

Step 5: Create a playbook to install Nginx on the host machine. To create a playbook you just need to open a file with a yml extension, like shown below:

```
vi <Name of your file>.yml
```

A terminal window showing the content of a playbook file. The text is as follows:

```
--
hosts: test-server
  sudo: yes
  vars:
    - server_port: 8080

tasks:
  - name: install nginx
    yum: pkg=nginx state=installed

  - name: serve nginx config
    template: src=../files/flask.conf dest=/etc/nginx/conf.d/
    notify:
      - restart nginx

handlers:
  - name: restart nginx
    service: name=nginx state=restarted
```

Ansible Playbook – Ansible Interview

In an Ansible Playbook, the tasks are defined as a list of dictionaries and are executed from top to bottom.

Each task is defined as a dictionary that can have several keys, such as “name” or “sudo” which signify the name of the task and whether it requires sudo privileges.

A variable `server_port` is set that listens on TCP port 8080 for incoming requests.

Here, the first task is to get the necessary package for installation of Nginx and then install it. Internally, Ansible will check if the directory exists and create it if it’s not, otherwise, it will do nothing.

The next task is to configure Nginx. In Nginx, contexts contain configuration details.

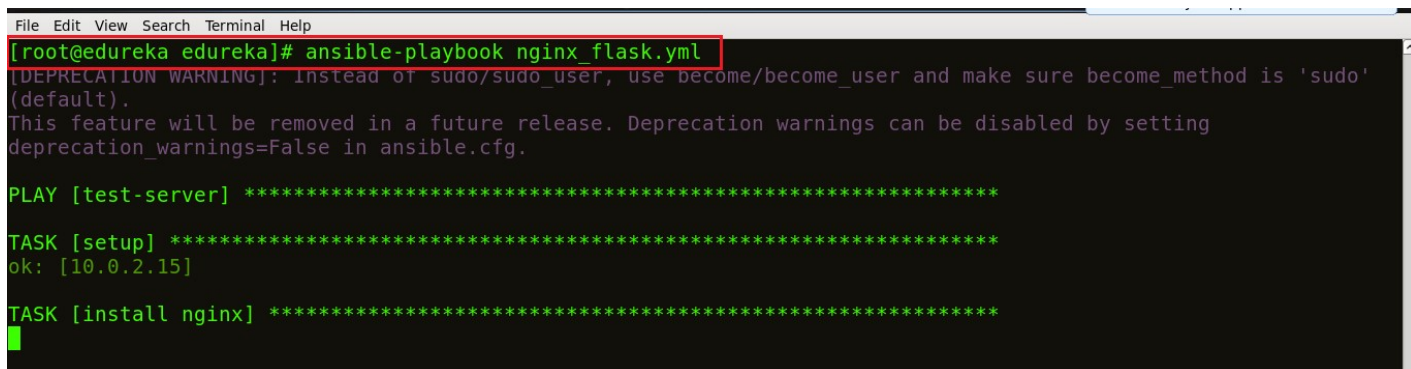
Here, the template is a file you can deploy on hosts. However, template files also include some reference variables which are pulled from variables defined as part of an Ansible playbook or facts gathered from the hosts. Facts containing the configuration details are being pulled from a source directory and being copied to a destination directory.

Handlers here define the action to be performed only upon notification of tasks or state changes. In this playbook, we defined, notify: restart Nginx handler which will restart Nginx once the files and templates are copied to hosts.

Now, save the file and exit.

Step 6: Run the playbook, using the command below:

```
ansible-playbook <name of your file>.yaml
```



```
File Edit View Search Terminal Help
[root@edureka edureka]# ansible-playbook nginx_flask.yml
[DEPRECATION WARNING]: Instead of sudo/sudo_user, use become/become_user and make sure become_method is 'sudo'
(default).
This feature will be removed in a future release. Deprecation warnings can be disabled by setting
deprecation_warnings=False in ansible.cfg.

PLAY [test-server] *****

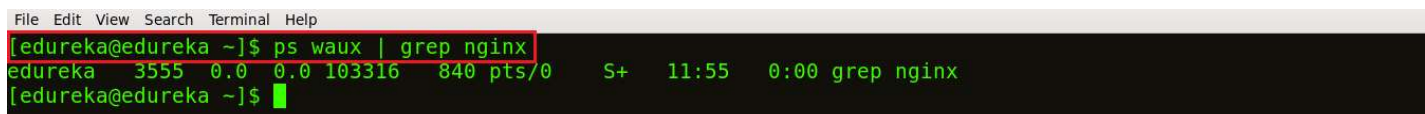
TASK [setup] *****
ok: [10.0.2.15]

TASK [install nginx] *****
```

Ansible Playbook – Ansible Interview Questions

Step 7: Check if Nginx is installed on the machine. Use the following command:

```
ps waux | grep nginx
```



```
File Edit View Search Terminal Help
[edureka@edureka ~]$ ps waux | grep nginx
edureka  3555  0.0  0.0 103316  840 pts/0    S+   11:55   0:00 grep nginx
[edureka@edureka ~]$
```

Ansible Playbook – Ansible Interview Questions

In the above image, the different process IDs 3555 and 103316 are running which shows that Nginx is running on your host machines.

Q20. How would you access a variable of the first host in a group?

This can be done by executing the below command:

```
{{ hostvars[groups['webservers'][0]]['ansible_eth0']['ipv4']['address'] }}
```

In the above command, we're basically accessing the hostname of the first machine in the webservers group. If you're using a template to do this, use the Jinja2 '#set' or you can also use set_fact, like shown below:

```
1         - set_fact: headnode={{ groups['webservers'][0] }}
2         - debug: msg={{ hostvars[headnode].ansible_eth0.ipv4.address }}
```

Q21. Why is '{{ }}' notation used? And how can one interpolate variables or dynamic variable names?

One basic rule is to 'always use {{}} except when:'. Conditionals are always run through Jinja2 as to resolve the expression. Therefore, 'when:failed_when:' and 'changed_when:' are always templated and we should avoid adding {{}}. In other cases, except when clause, we have to use brackets, otherwise, differentiating between an undefined variable and a string will be difficult to do.

Q22. What is Ansible role and how are they different from the playbook?

[Ansible Roles](#) is basically another level of abstraction used to organize playbooks. They provide a skeleton for an independent and reusable collection of variables, tasks, templates, files, and modules which can be automatically loaded into the playbook. Playbooks are a collection of roles. Every role has specific functionality.

Let's understand the difference between Ansible roles and playbook with an example.

Suppose you want your playbook to perform 10 different tasks on 5 different systems, would you use a single playbook for this? No, using a single playbook can make it confusing and prone to blunders. Instead, you can create 10 different roles, where each role will perform one task. Then, all you need to do is, mention the name of the role inside the playbook to call them.

Q23. How do I write an Ansible handler with multiple tasks?

Suppose you want to create a handler that restarts a service only if it is already running.

```
1         - name: Check if restarted
2           shell: check_is_started.sh
3             register: result
4           listen: Restart processes
5
6
7         - name: Restart conditionally step 2
8           service: name=service state=restarted
```

```

9             when: result
10            listen: Restart processes
11

```

Handlers can “listen” to generic topics, and tasks can notify those topics as shown below. This functionality makes it much easier to trigger multiple handlers. It also decouples handlers from their names, making it easier to share handlers among playbooks and roles:

Q24. How to keep secret data in a playbook?

Suppose you have a task that you don’t want to show the output or command given to it when using -v (verbose) mode, the following task can be used to do it:

```

1             - name: secret task
2               shell: /usr/bin/do something --value={{ secret value }}
3               no_log: True

```

This can be used to keep verbose output but hide sensitive information from others who would otherwise like to be able to see the output.

The no_log attribute can also apply to an entire play:

```

1             - hosts: all
2               no_log: True

```

Q25. What are Ansible Vaults and why are they used?

[Ansible Vault](#) is a feature that allows you to keep all your secrets safe. It can encrypt entire files, entire YAML playbooks or even a few variables. It provides a facility where you can not only encrypt sensitive data but also integrate them into your playbooks.

Vault is implemented with file-level granularity where the files are either entirely encrypted or entirely unencrypted. It uses the same password for encrypting as well as for decrypting files which makes using Ansible Vault very user-friendly.

Q26. How to create encrypted files using Ansible?

To create an encrypted file, use the ‘ansible-vault create’ command and pass the filename.

```
$ ansible-vault create filename.yaml
```

You’ll be prompted to create a password and then confirm it by re-typing it.

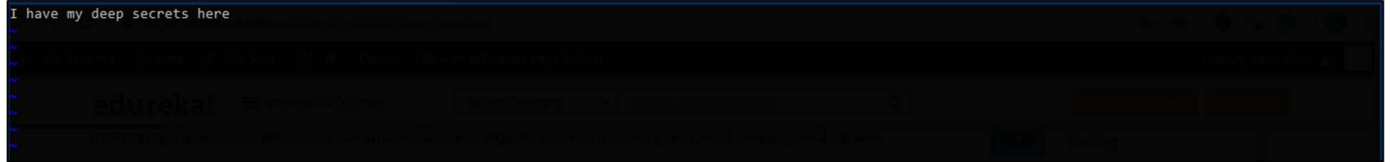
```

ubuntu@ip-10-0-2-54:~$ ansible-vault create secrets.txt
New Vault password:
Confirm New Vault password:

```

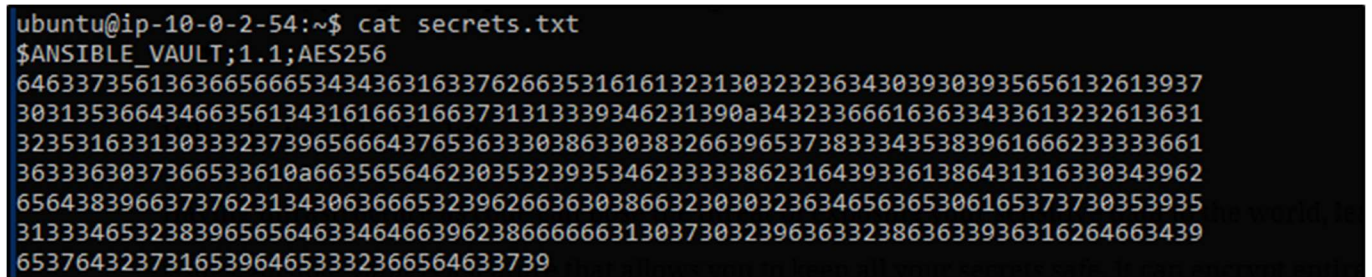

Ansible Playbook – Ansible Interview Questions

Once your password is confirmed, a new file will be created and will open an editing window. By default, the editor for Ansible Vault is vi. You can add data, save and exit.



Ansible Playbook – Ansible Interview Questions

This is your encrypted file:



Ansible Playbook – Ansible Interview Questions

Q27. 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.

Q28. What features does the Ansible Tower provide?

- **Ansible Tower Dashboard** – The Ansible Tower dashboard displays everything going on in your Ansible environment like the hosts, inventory status, the recent job activity and so on.
- **Real-Time Job Updates** – As Ansible can automate the complete infrastructure, you can see real-time job updates, like plays and tasks broken down by each machine either been successful or a failure. So, with this, you can see the status of your automation, and know what's next in the queue.

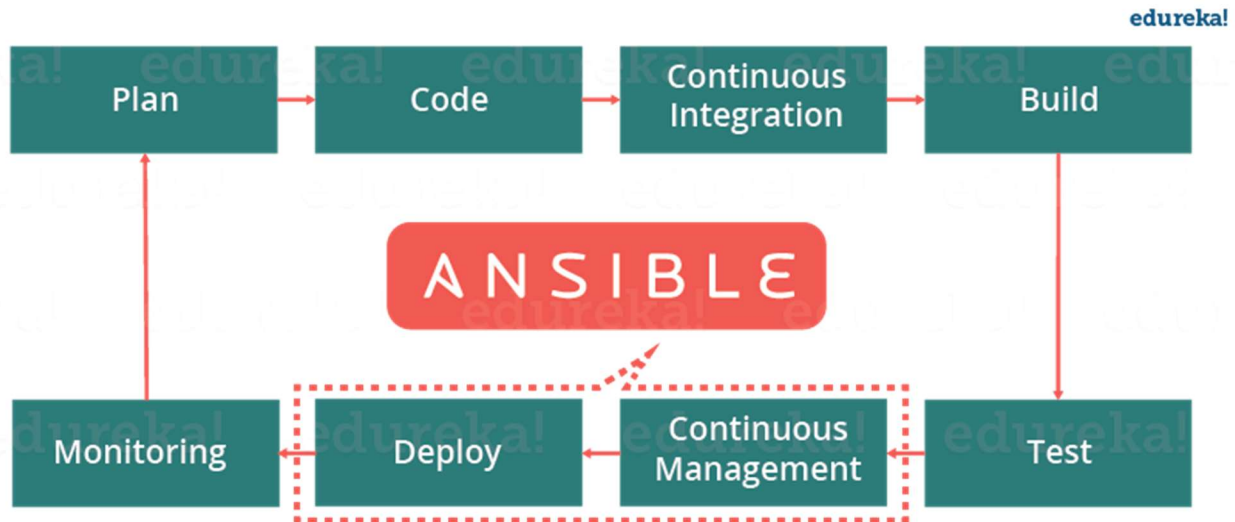
- **Multi-Playbook Workflows** – This feature allows you to chain any number of playbooks, regardless of the usage of different inventories, utilizes various credentials, or runs different users.
- **Who Ran What Job When** – As the name suggests, you can easily know who ran what job where and when as, all the automation activity is securely logged in Ansible Tower.
- **Scale Capacity With Clusters** – We can connect multiple Ansible Tower nodes into an Ansible Tower cluster as the clusters add redundancy and capacity, which allow you to scale Ansible automation across the enterprise.
- **Integrated Notifications** – This feature lets you notify a person or team when a job succeeds or fails across the entire organization at once, or customize on a per-job basis.
- **Schedule Ansible Jobs** – Different kinds of jobs such as Playbook runs, cloud inventory updates, and source control updates can be scheduled inside Ansible Tower to run according to the need.
- **Manage & Track Inventory** – Ansible Tower helps you manage your entire infrastructure by letting you easily pull inventory from public cloud providers such as Amazon Web Services, Microsoft Azure, and more.
- **Self-Service** – This feature of Ansible Tower lets you launch Playbooks with just a single click. It can also, let you choose from available secure credentials or prompt you for variables and monitor the resulting deployments.
- **REST API & Tower CLI Tool** – Every feature present in Ansible Tower is available via Ansible Tower's REST API, which provides the ideal API for a systems management infrastructure. The Ansible Tower's CLI tool is available for launching jobs from CI systems such as Jenkins, or when you need to integrate with other command-line tools.
- **Remote Command Execution** – You can run simple tasks such as add users, restart any malfunctioning service, reset passwords on any host or group of hosts in the inventory with Ansible Tower's remote command execution.

The following section will cover only the technical based Ansible Interview Questions. These questions are mostly based on the practical implementation of Ansible.

Ansible Technical Interview Questions

Q29. How is Ansible used in a Continuous Delivery pipeline? Explain.

It is well known that in DevOps development and operations work is integrated. This integration is very important for modern test-driven applications. Hence, Ansible integrates this by providing a stable environment to both development and operations resulting in a smooth delivery pipeline.



Ansible In DevOps – Ansible Interview Questions

When developers begin to think of infrastructure as part of their application i.e as Infrastructure as code (IaC), stability and performance become normative. Infrastructure as Code is the process of managing and provisioning computing infrastructure and their configuration through machine-processable definition files, rather than physical hardware configuration or the use of interactive configuration tools. This is where Ansible automation plays a major role and stands out among its peers.

In a Continuous Delivery pipeline, Sysadmins work tightly with developers, development velocity is improved, and more time is spent doing activities like performance tuning, experimenting, and getting things done, and less time is spent fixing problems.

Q30. How can you create a LAMP stack and deploy a webpage by using Ansible?

Suppose you're trying to deploy a website on 30 systems, every website deployment will require a base OS, web-server, Database, and PHP. We use ansible playbook to install these prerequisites on all 30 systems at once.

For this particular problem statement, you can use two virtual machines, one as a server where Ansible is installed and the other machine acts as the remote host. Also, I've created a simple static webpage saved in a folder index which has two files, index.html, and style.css.

In the below code I've created a single Ansible playbook to install Apache, MySql, and PHP:

```

1
2
3
4      ---
5      # Setup LAMP Stack
6      - hosts: host1
7        tasks:
8
9      - name: Add ppa repository
10        become: yes
11        apt_repository: repo=ppa:ondrej/php
12
13      - name: Install lamp stack
14        become: yes
15        apt:
16          pkg:
17            - apache2
18            - mysql-server
19            - php7.0
20            - php7.0-mysql
21          state: present
22          update_cache: yes
23
24      - name: start apache server
25        become: yes
26        service:
27          name: apache2
28          state: started
29          enabled: yes
30
31      - name: start mysql service
32        become: yes
33        services:
34          name: mysql
35          state: started
36          enabled: yes
37
38      - name: create target directory
39        file: path=/var/www/html state=directory mode=0755
40
41      - name: deploy index.html
42        become: yes
43        copy:
44          src: /etc/ansible/index/index.html
45          dest: var/www/html/index/index.html

```

Now, there are 6 main tasks, each task performs a specific function:

- The first task adds the repository required to install MySQL and PHP.
- The second task installs apache2, MySQL-server, PHP, and PHP-MySQL.

- The third and fourth task starts the Apache and MySQL service.
- The fifth task creates a target directory in the host machine and
- Finally, the sixth task executes the index.html file, it picks up the file from the server machine and copies it into the host machine.

To finally run this playbook you can use the following command:

```
$ ansible-playbook lamp.yml -K
```

Q31. How do I set the PATH or any other environment variable for a task?

The environment variables can be set by using the 'environment' keyword. It can be set for either a task or an entire playbook as well. Follow the below code snippet to see how:

```
1                                     environment:
2     PATH: "{{ ansible_env.PATH }}:/thingy/bin"
3     SOME: value
```

Q32. How can one generate encrypted passwords for the user module?

There are a couple of ways to do this. The simplest way is to use the ad-hoc command:

```
ansible all -i localhost, -m debug -a "msg={{ 'mypassword' |
password_hash('sha512', 'mysecretsalt') }}"
```

Another way is to use the mkpasswd functionality available on Linux systems:

```
mkpasswd --method=sha-512
```

However, if you're using a macOS then you can generate these passwords using Python. To do this you must first install the Passlib password hashing library:

```
pip install passlib
```

After installing it, the SHA512 password values can be generated in the following manner:

```
python -c "from passlib.hash import sha512_crypt; import getpass;
print(sha512_crypt.using(rounds=5000).hash(getpass.getpass()))"
```

Q33. How can looping be done over a list of hosts in a group, inside of a template?

An easy way to do this is to iterate over a list of hosts inside of a host group, in order to fill a template configuration file with a list of servers. This can be done by accessing the "\$groups" dictionary in your template, like so:

```

1           {% for host in groups['db_servers'] %}
2               {{ host }}
3           {% endfor %}

```

In order to access facts about these hosts, like, the IP address of each hostname, you need to make sure that the facts have been populated. For instance, make sure you have a play that talks to db_servers:

```

1           - hosts: db_servers
2             tasks:
3       - debug: msg="doesn't matter what you do, just that they were talked to
                previously."

```

Now you can use the facts within your template, like so:

```

1           {% for host in groups['db_servers'] %}
2               {{ hostvars[host]['ansible_eth0']['ipv4']['address'] }}
3           {% endfor %}

```

Q34. How can I display all the inventory vars defined for my host?

In order to check the inventory vars resulting from what you've defined in the inventory, you can execute the below command:

```
ansible -m debug -a "var=hostvars['hostname']" localhost
```

This will list down all the inventory vars.

Q35. How should one configure a jump host to access servers that I have no direct access to?

The first step would be to set a ProxyCommand in the ansible_ssh_common_args inventory variable. All arguments that are defined in this variable are added to the sftp/scp/ssh command line when connecting to the relevant host. Let's look at an example, consider the below inventory group:

```

1           [gatewayed]
2           foo ansible_host=192.0.2.1
3           bar ansible_host=192.0.2.2

```

Next, you can create group_vars/gatewayed.yml containing the following:

```
ansible_ssh_common_args: '-o ProxyCommand="ssh -W %h:%p -q user@gateway.example.com"'
```

Ansible will then append these arguments to the command line while trying to connect to any hosts in the group gatewayed.

Q36. How can you handle different machines needing different user accounts or ports to log in with?

The simplest way to do this is by setting inventory variables in the inventory file.

Let's consider that these hosts have different usernames and ports:

```
1                                     [webservers]
2      asdf.example.com ansible_port=5000 ansible_user=alice
3      jkl.example.com ansible_port=5001 ansible_user=bob
```

Also, if you wish to, you can specify the connection type to be used:

```
1                                     [testcluster]
2      localhost ansible_connection=local
3      /path/to/chroot1 ansible_connection=chroot
4      foo.example.com ansible_connection=paramiko
```

To make this more clear it is best to keep these in group variables or file them in a `group_vars/<group-name>` file.

Q37. Is it unsafe to bulk-set task arguments from a variable?

To set all the arguments in a task you can use the dictionary-typed variable. Even though this is usually good for dynamic executions, it induces a security risk. Therefore, when this happens, Ansible issues a warning. For example, consider the below code:

```
1      vars:
2      usermod_args:
3      name: testuser
4      state: present
5      tasks:
6      - user: '{{ usermod_args }}'
```

This example is safe but creating similar tasks is risky because the parameters and values passed to `usermod_args` could be overwritten by malicious values in the host facts on a compromised target machine.

Q38. Suppose you're using Ansible to configure the production environment and your playbook uses an encrypted file. Encrypted files prompt the user to enter passwords. But since Ansible is used for automation, can this process be automated?

Yes, Ansible uses a feature called password file, where all the passwords to your encrypted files can be saved. So each time the user is asked for the password, he can simply make a call to the password file. The password is automatically read and entered by Ansible.

```
$ ansible-playbook launch.yml --vault-password-file ~/ .vault_pass.txt
```

Having a separate script that specifies the passwords is also possible. You need to make sure the script file is executable and the password is printed to standard output for it to work without annoying errors.

```
$ ansible-playbook launch.yml --vault-password-file ~/ .vault_pass.py
```

Q39. Have you worked with Ansible before? Please share your experience.

Be very honest here. If you have used ansible before then talk about your experience. Talk about the projects that required ansible. You can tell the interviewer about how Ansible has helped you in provisioning and configuration management. If you haven't used Ansible before then just talk about any related tools that you've used. These related tools could be anything like Git, Jenkins, Puppet, Chef, Saltstack, etc.

Be very honest because they know if you're lying.

Q40. Is Ansible an Open Source tool?

Yes, Ansible is open source. That means you take the modules and rewrite them. Ansible is an open-source automated engine that lets you automate apps.

Q41. How can you connect other devices within Ansible?

Once Ansible is installed on the controlling machines, an inventory file is created. This inventory file specifies the connection between other nodes. A connection can be made using a simple SSH. To check the connection to a different device, you can use the ping module.

```
ansible -m ping all
```

The above command checks the connection to all the nodes specified in the inventory file.

Q42. Is it possible to build our modules in Ansible?

Yes, we can create our own modules within Ansible. It's an open-source tool which basically works on python. You can start creating your own modules. The only requirements would be to be amazingly good at programming.

Q43. What does Fact mean in Ansible?

When any new variable about the system has been discovered it's considered to be a "fact" in the playbook. Facts are mainly used to implement conditional executions. It can also be used to get the ad-hoc information about the system.

You can get facts with the following command:

```
$ ansible all- m setup
```

So when you want to extract only a part of the information, you use the setup module to filter out only the needed information.

Q44. What is the ask_pass module in Ansible?

Ask_pass is the control module in an Ansible playbook. This controls the prompting of the password when the playbook is getting executed. By default, it's always set to True. If you are using SSH keys for authentication purposes then you really don't have to change this setting at all.

Q45. Explain the callback plugin in Ansible?

Callback plugins are enable adding new behaviors to Ansible when responding to events. By default, callback plugins control most of the output you see when running the command line program. It can also be used to add additional output, integrate with other tools, etc.

Q46. Does Ansible support AWS?

Ansible has hundreds of modules supporting AWS and some of them include:

- Autoscaling groups
- CloudFormation
- CloudTrail
- CloudWatch
- DynamoDB
- ElastiCache
- Elastic Cloud Compute (EC2)
- Identity Access Manager (IAM)
- Lambda
- Relational Database Service (RDS)
- Route53
- Security Groups
- Simple Storage Service (S3)
- Virtual Private Cloud (VPC)

Q47. Does Ansible support hardware provisioning?

Yes, Ansible can provision hardware. A lot of companies are still stuck on to massive data centers of hardware. There are a few requirements. You must set up some services before you go ahead. Some of them are – DHCP, PXE, TFTP, Operating System Media, Web Server, etc.

Q48. Write an Ansible playbook to automate the starting of EC2 instance.

```
1
2      ---
3      - name: Create an ec2 instance
4          hosts: web
5          gather_facts: false
6
7          vars:
8              region: us-east-1
9              instance_type: t2.micro
10             ami: ami-05ea7729e394412c8
11             keypair: priyajdm
12
13             tasks:
14
15                 - name: Create an ec2 instance
16                     ec2:
17                         aws_access_key: '*****'
18                         aws_secret_key: '*****'
19                         key_name: "{{ keypair }}"
20                         group: launch-wizard-26
21                         instance_type: "{{ instance_type }}"
22                         image: "{{ ami }}"
23                         wait: true
24                         region: "{{ region }}"
25                         count: 1
26                         vpc_subnet_id: subnet-02f498e16fd56c277
27                         assign_public_ip: yes
28                         register: ec2
```


- We start by mentioning AWS access key id and secret key using the parameters **aws_access_key** and **aws_secret_key**.
- **key_name**: pass the variable that defines the keypair being used here
- **group**: mention the name of the security group. This defines the security rules of the EC2 instance we're trying to bring up
- **instance_type**: pass the variable that defines the type of instance we're using here
- **image**: pass the variable that defines the AMI of the image we're trying to start.
- **wait**: This has a boolean value of either true or false. If true, it waits for the instance to reach the desired state before returning
- **region**: pass the variable that defines the region in which an EC2 instance needs to be created.
- **count**: This parameter specifies the number of instances that need to be created. In this case, I've only mentioned only one but this depends on your requirements.
- **vpc_subnet_id**: pass the subnet id in which you wish to create the instance
- **assign_public_ip**: This parameter has a boolean value. If true like in our case, a public IP will be assigned to the instance when provisioned within VPC.

Q49. Can you copy files recursively onto a target host? If yes, how?

Yes, you can copy files recursively onto a target host using the copy module. It has a recursive parameter which copies files from a directory. There is another module called synchronize which is specifically made for this.

```

1           - synchronize:
2             src: /first/absolute/path
3             dest: /second/absolute/path
4             delegate_to: "{{ inventory_hostname }}"

```

Q50. Write a playbook to create a backup of a file in the remote servers before copy.

This is pretty simple. You can use the below playbook:

```

1           - hosts: blocks
2             tasks:
3               - name: ansible copy file backup example
4                 copy:
5                   src: ~/helloworld.txt
6                   dest: /tmp
7                   backup: yes

```

