## Ansible Interview Question And Answers:

### Q. How Ansible Works?

There are many similar automation tools available like Puppet, Capistrano, Chef, Salt, Space Walk etc, but Ansible categorize into two types of server: controlling machines and nodes.

The controlling machine, where Ansible is installed and Nodes are managed by this controlling machine over SSH. The location of nodes are specified by controlling machine through its inventory.

The controlling machine (Ansible) deploys modules to nodes using SSH protocol and these modules are stored temporarily on remote nodes and communicate with the Ansible machine through a JSON connection over the standard output.

Ansible is agent-less, that means no need of any agent installation on remote nodes, so it means there are no any background daemons or programs are executing for Ansible, when it’s not managing any nodes.

Ansible can handle 100’s of nodes from a single system over SSH connection and the entire operation can be handled and executed by one single command ‘ansible’. But, in some cases, where you required to execute multiple commands for a deployment, here we can build playbooks.

Playbooks are bunch of commands which can perform multiple tasks and each playbooks are in YAML file format.

### Q.What’s the Use of Ansible.

Ansible can be used in IT infrastructure to manage and deploy software applications to remote nodes. For example, let’s say you need to deploy a single software or multiple software to 100’s of nodes by a single command, here ansible comes into picture, with the help of Ansible you can deploy as many as applications to many nodes with one single command, but you must have a little programming knowledge for understanding the ansible scripts.

We’ve compiled a series on Ansible, title ‘Preparation for the Deployment of your IT Infrastructure with Ansible IT Automation Tool‘, through parts 1-4 and covers the following topics.

### Q.How would you describe yourself in terms of what you do and how you’d like to be remembered?

Obviously I’d like to be remembered as a master of prose who forever changed the face of literature as we know it, but I’m going to have to settle for being remembered as a science fiction writer (and, more and more, critic) who wrote the occasional funny line and picked up a few awards.

### Q.Why are you attracted to science and science fiction?

Early imprinting, maybe, for the science fiction. When I was quite small a family friend let me read his 1950s run of ‘Galaxy’ magazine. My favourite aunt pressed John Wyndham’s ‘The Day of the Triffids’ on me; a more terrifying great-aunt gave me G.K. Chesterton’s fantastic novels; and so on.

The incurable addiction had begun. Meanwhile, science classes just seemed to be the part of school that made most sense, and I fell in love with Pelican pop-maths titles – especially Kasner’s and Newman’s ‘Mathematics and the Imagination’ and all those books of Martin Gardner’s ‘Scientific American’ columns.

### Q. Tell us about your software company and what sort of software it produced(s).

This goes back to the 1980s and the Apricot home computers, the early, pretty and non-PC-compatible ones. My pal Chris Priest and I both used them for word processing, and he persuaded me to put together a disk of utilities to improve the bundled ‘SuperWriter’ w/p, mostly written in Borland Turbo Pascal 3 and later 4: two-column printing, automated book index preparation, cleaning the crap out of the spellcheck dictionary, patching SuperWriter to produce dates in UK format, and so on.

Then I redid the index software (‘AnsibleIndex’) in CP/M for the Amstrad PCW and its Locoscript word processors. When the Apricot market collapsed, I wrote an Apricot emulator in assembler so that people could keep using their horrible but familiar old software on a PC. Eventually, in a fit of nostalgia, I collected all my columns for ‘Apricot File’ and various Amstrad PCW magazines as books unoriginally titled ‘The Apricot Files’ and ‘The Limbo Files’. (That’s probably enough self-promotion, but there’s lots more at <http://ansible.co.uk/>.)

### Q.Describe your newsletter Ansible and who it’s aimed at.

It appears monthly and has been called the ‘Private Eye’ of science fiction, but isn’t as cruel and doesn’t (I hope) recycle old jokes quite as relentlessly. Though I feel a certain duty to list some bread-and-butter material like conventions, award winners and deaths in the field, ‘Ansible’ skips the most boring SF news – the long lists of books acquired, books published, book sales figures, major new remainders – in favour of quirkier items and poking fun at SF notables. The most popular departments quote terrible lines from published SF/fantasy and bizarre things said about SF by outsiders (‘As Others See Us’). All the back issues of ‘Ansible’ since it started in 1979 can be read online:

What is Ansible?

[Ansible](https://tekslate.com/tutorials/ansible/) is a software tool to deploy application using ssh without sny downtime.It is also used to manage and configure software applications. Ansible is developed by Python language.

What are the Advantages of Ansible?

-Agent-less

-Verylow overhead

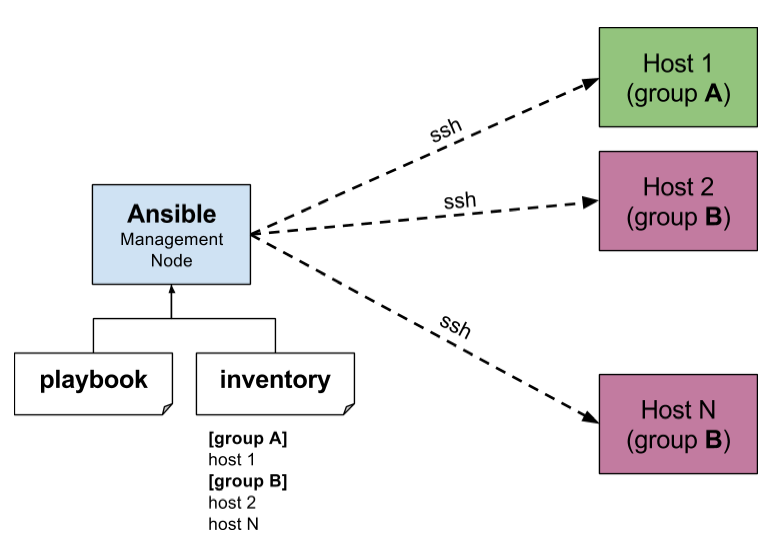
-Good performance

How Ansible Works?

There are many similar automation tools available like Puppet, Capistrano, Chef, Salt, Space Walk etc, but Ansible categorize into two types of server: controlling machines and nodes.

The controlling machine, where Ansible is installed and Nodes are managed by this controlling machine over SSH. The location of nodes are specified by controlling machine through its inventory.

The controlling machine (Ansible) deploys modules to nodes using SSH protocol and these modules are stored temporarily on remote nodes and communicate with the Ansible machine through a JSON connection over the standard output.



Ansible is agent-less, that means no need of any agent installation on remote nodes, so it means there are no any background daemons or programs are executing for Ansible, when it’s not managing any nodes.

Ansible can handle 100’s of nodes from a single system over SSH connection and the entire operation can be handled and executed by one single command ‘ansible’. But, in some cases, where you required to execute multiple commands for a deployment, here we can build playbooks.

Playbooks are bunch of commands which can perform multiple tasks and each playbooks are in YAML file format.

What’s the Use of Ansible?

Ansible can be used in IT infrastructure to manage and deploy software applications to remote nodes. For example, let’s say you need to deploy a single software or multiple software to 100’s of nodes by a single command, here ansible comes into picture, with the help of Ansible you can deploy as many as applications to many nodes with one single command, but you must have a little programming knowledge for understanding the ansible scripts.

We’ve compiled a series on [Ansible](https://tekslate.com/ansible-training/), title ‘Preparation for the Deployment of your IT Infrastructure with Ansible IT Automation Tool‘, through parts 1-4 and covers the following topics.

What are the Advantages of Ansible?

– Agent-less

– very overload

– Good Performance.

Is there a web interface / REST API / etc?

Yes, Ansible, Inc makes a great product that makes Ansible even more powerful and easy to use. See [Ansible Tower](http://docs.ansible.com/ansible/tower.html).

How do I submit a change to the documentation?

Documentation for Ansible is kept in the main project git repository, and complete instructions for contributing can be found in the docs.

When should I use {{ }}? Also, how to interpolate variables or dynamic variable names

A steadfast rule is ‘always use {{ }} except when when:‘. Conditionals are always run through Jinja2 as to resolve the expression, so when: failed\_when: and changed\_when: are always templated and you should avoid adding {{}}.

In most other cases you should always use the brackets, even if previouslly you could use variables without specifying (like with\_ clauses), as this made it hard to distinguish between an undefined variable and a string.

Another rule is ‘moustaches don’t stack’. We often see this:

{{ somevar\_{{other\_var}} }}

The above DOES NOT WORK, if you need to use a dynamic variable use the hostvars or vars dictionary as appropriate:

{{ hostvars[inventory\_hostname]['somevar\_' + other\_var] }}

How to install Ansible

Installation of Ansible Ubuntu 14.04

The best way to get Ansible for Ubuntu is to add the project’s PPA (personal package archive) to your system.

To do this effectively, we need to install the software-properties-common package, which will give us the ability to work with PPAs easily. (This package was called python-software-properties on older versions of Ubuntu.)

sudo apt-get update

sudo apt-get install software-properties-common

Once the package is installed, we can add the Ansible PPA by typing the following command:

sudo apt-add-repository ppa:ansible/ansible

Press ENTER to accept the PPA addition.

Next, we need to refresh our system’s package index so that it is aware of the packages available in the PPA. Afterwards, we can install the software:

sudo apt-get update

sudo apt-get install ansible

We now have all of the software required to administer our servers through Ansible.

How do I generate crypted passwords for the user module?

The mkpasswd utility that is available on most Linux systems is a great option:

mkpasswd --method=sha-512

If this utility is not installed on your system (e.g. you are using OS X) then you can still easily generate these passwords using Python. First, ensure that the [Passlib](https://code.google.com/p/passlib/) password hashing library is installed.

pip install passlib

Once the library is ready, SHA512 password values can then be generated as follows:

python -c "from passlib.hash import sha512\_crypt; import getpass; print sha512\_crypt.encrypt(getpass.getpass())"

Use the integrated [Hashing filters](http://docs.ansible.com/ansible/playbooks_filters.html#hash-filters) to generate a hashed version of a password. You shouldn’t put plaintext passwords in your playbook or host\_vars; instead, use [Vault](http://docs.ansible.com/ansible/playbooks_vault.html) to encrypt sensitive data.

Desired to gain proficiency on Ansible?

Explore the blog post on [Ansible training](https://tekslate.com/ansible-training/) to become a pro in Ansible.

How do I get ansible to reuse connections, enable Kerberized SSH, or have Ansible pay attention to my local SSH config file?

Switch your default connection type in the configuration file to ‘ssh’, or use ‘-c ssh’ to use Native OpenSSH for connections instead of the python paramiko library. In Ansible 1.2.1 and later, ‘ssh’ will be used by default if OpenSSH is new enough to support ControlPersist as an option.

Paramiko is great for starting out, but the OpenSSH type offers many advanced options. You will want to run Ansible from a machine new enough to support ControlPersist, if you are using this connection type. You can still manage older clients. If you are using RHEL 6, CentOS 6, SLES 10 or SLES 11 the version of OpenSSH is still a bit old, so consider managing from a Fedora or openSUSE client even though you are managing older nodes, or just use paramiko.

We keep paramiko as the default as if you are first installing Ansible on an EL box, it offers a better experience for new users.

What is the best way to make content reusable/redistributable?

If you have not done so already, read all about “Roles” in the playbooks documentation. This helps you make playbook content self-contained, and works well with things like git submodules for sharing content with others.

If some of these plugin types look strange to you, see the API documentation for more details about ways Ansible can be extended.

How do I see all the inventory vars defined for my host?

You can see the resulting vars you define in inventory running the following command:

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

How do I copy files recursively onto a target host?

The “copy” module has a recursive parameter, though if you want to do something more efficient for many files, look at the “synchronize” module instead, which wraps rsync. See the module index for info on both modules.

What is Ansible Role?

Ansible can interact with configured clients from the command line with the ansible command, and how you can automate configuration with playbooks run through the ansible-playbook command.

The first step in creating a role is creating its directory structure. To create the base directory structure, we’re going to use a tool bundled with Ansible called ansible-galaxy:

$ ansible-galaxy init azavea.packer

azavea.packer was created successfully

That command will create an azavea.packer directory with the following structure:

├── README.md

├── defaults

│ └── main.yml

├── files

├── handlers

│ └── main.yml

├── meta

│ └── main.yml

├── tasks

│ └── main.yml

├── templates

└── vars

└── main.yml

How do I access a variable name programmatically?

An example may come up where we need to get the ipv4 address of an arbitrary interface, where the interface to be used may be supplied via a role parameter or other input. Variable names can be built by adding strings together, like so:

{{ hostvars[inventory\_hostname]['ansible\_' + which\_interface]['ipv4']['address'] }}

The trick about going through hostvars is necessary because it’s a dictionary of the entire namespace of variables. ‘inventory\_hostname’ is a magic variable that indicates the current host you are looping over in the host loop.

How do I access shell environment variables?

If you just need to access existing variables, use the ‘env’ lookup plugin. For example, to access the value of the HOME environment variable on management machine:

---

# ...

vars:

local\_home: "{{ lookup('env','HOME') }}"

If you need to set environment variables, see the Advanced Playbooks section about environments.

Ansible 1.4 will also make remote environment variables available via facts in the ‘ansible\_env’ variable:

{{ ansible\_env.SOME\_VARIABLE }}

Our design of course tutorials and interview questions is practical and informative. At TekSlate, we offer resources to help you learn various IT courses. We avail both written material and demo video tutorials. For in-depth knowledge and practical experience explore [Online Ansible Training](https://tekslate.com/ansible-training/).

1. What is the difference between shell and command modules?
2. How would you copy a file from local to remote location? How would you the other way round? What is the difference between the copy and file module?
3. Why do you create roles in ansible?
4. What is the advantage/disadvantage of vagrant? When you would not use it with ansible?
5. Is it better to use only ansible and don't use the vagrant?
6. How would you connect to the Amazon machine through ansible?
7. Have you ever used unarchive module? How does it differ from using tar in shell command? Which one would you prefer and why?
8. How do you pass variable value from one inventory to another?
9. How can you find out the current running host in the ansible?

ANSIBLE

There are many similar automation tools available like Puppet, Capistrano, Chef, Salt, Space Walk etc, but [**Ansible**](http://mindmajix.com/ansible-interview-questions/) categorize into two types of server: controlling machines and nodes. The controlling machine, where Ansible is installed and Nodes are managed by this controlling machine over SSH. The location of nodes are specified by controlling machine through its inventory. The controlling machine (Ansible) deploys modules to nodes using SSH protocol and these modules are stored temporarily on remote nodes and communicate with the Ansible machine through a JSON connection over the standard output. Ansible is agent-less, that means no need of any agent installation on remote nodes, so it means there are no any background daemons or programs are executing for Ansible, when it’s not managing any nodes.  
  
Ansible can handle 100’s of nodes from a single system over SSH connection and the entire operation can be handled and executed by one single command ‘ansible’. But, in some cases, where you required to execute multiple commands for a deployment, here we can build playbooks. Playbooks are bunch of commands which can perform multiple tasks and each playbooks are in YAML file format.

## How can I set the PATH or any other environment variable for a task or entire playbook?

Setting environment variables can be done with the environment keyword. It can be used at the task or the play level:

environment:

PATH: "{{ ansible\_env.PATH }}:/thingy/bin"

SOME: value

Note

starting in 2.0.1 the setup task from gather\_facts also inherits the environment directive from the play, you might need to use the |default filter to avoid errors if setting this at play level.

## How do I handle different machines needing different user accounts or ports to log in with?

Setting inventory variables in the inventory file is the easiest way.

Note

Ansible 2.0 has deprecated the “ssh” from ansible\_ssh\_user, ansible\_ssh\_host, and ansible\_ssh\_port to become ansible\_user, ansible\_host, and ansible\_port. If you are using a version of Ansible prior to 2.0, you should continue using the older style variables (ansible\_ssh\_\*). These shorter variables are ignored, without warning, in older versions of Ansible.

For instance, suppose these hosts have different usernames and ports:

[webservers]

asdf.example.com ansible\_port=5000 ansible\_user=alice

jkl.example.com ansible\_port=5001 ansible\_user=bob

You can also dictate the connection type to be used, if you want:

[testcluster]

localhost ansible\_connection=local

/path/to/chroot1 ansible\_connection=chroot

foo.example.com ansible\_connection=paramiko

You may also wish to keep these in group variables instead, or file them in a group\_vars/<groupname> file. See the rest of the documentation for more information about how to organize variables.

## How do I get ansible to reuse connections, enable Kerberized SSH, or have Ansible pay attention to my local SSH config file?

Switch your default connection type in the configuration file to ‘ssh’, or use ‘-c ssh’ to use Native OpenSSH for connections instead of the python paramiko library. In Ansible 1.2.1 and later, ‘ssh’ will be used by default if OpenSSH is new enough to support ControlPersist as an option.

Paramiko is great for starting out, but the OpenSSH type offers many advanced options. You will want to run Ansible from a machine new enough to support ControlPersist, if you are using this connection type. You can still manage older clients. If you are using RHEL 6, CentOS 6, SLES 10 or SLES 11 the version of OpenSSH is still a bit old, so consider managing from a Fedora or openSUSE client even though you are managing older nodes, or just use paramiko.

We keep paramiko as the default as if you are first installing Ansible on an EL box, it offers a better experience for new users.

## How do I configure a jump host to access servers that I have no direct access to?

With Ansible 2, you can set a ProxyCommand in the ansible\_ssh\_common\_args inventory variable. Any arguments specified in this variable are added to the sftp/scp/ssh command line when connecting to the relevant host(s). Consider the following inventory group:

[gatewayed]

foo ansible\_host=192.0.2.1

bar ansible\_host=192.0.2.2

You can create group\_vars/gatewayed.yml with the following contents:

ansible\_ssh\_common\_args: '-o ProxyCommand="ssh -W %h:%p -q user@gateway.example.com"'

Ansible will append these arguments to the command line when trying to connect to any hosts in the group gatewayed. (These arguments are used in addition to any ssh\_args from ansible.cfg, so you do not need to repeat global ControlPersist settings in ansible\_ssh\_common\_args.)

Note that ssh -W is available only with OpenSSH 5.4 or later. With older versions, it’s necessary to execute nc %h:%p or some equivalent command on the bastion host.

With earlier versions of Ansible, it was necessary to configure a suitable ProxyCommand for one or more hosts in ~/.ssh/config, or globally by setting ssh\_args in ansible.cfg.

## How do I speed up management inside EC2?

Don’t try to manage a fleet of EC2 machines from your laptop. Connect to a management node inside EC2 first and run Ansible from there.

## How do I handle python pathing not having a Python 2.X in /usr/bin/python on a remote machine?

While you can write ansible modules in any language, most ansible modules are written in Python, and some of these are important core ones.

By default, Ansible assumes it can find a /usr/bin/python on your remote system that is a 2.X version of Python, specifically 2.6 or higher.

Setting the inventory variable ‘ansible\_python\_interpreter’ on any host will allow Ansible to auto-replace the interpreter used when executing python modules. Thus, you can point to any python you want on the system if /usr/bin/python on your system does not point to a Python 2.X interpreter.

Some Linux operating systems, such as Arch, may only have Python 3 installed by default. This is not sufficient and you will get syntax errors trying to run modules with Python 3. Python 3 is essentially not the same language as Python 2. Python 3 support is being worked on but some Ansible modules are not yet ported to run under Python 3.0. This is not a problem though as you can just install Python 2 also on a managed host.

Do not replace the shebang lines of your python modules. Ansible will do this for you automatically at deploy time.

## What is the best way to make content reusable/redistributable?

If you have not done so already, read all about “Roles” in the playbooks documentation. This helps you make playbook content self-contained, and works well with things like git submodules for sharing content with others.

If some of these plugin types look strange to you, see the API documentation for more details about ways Ansible can be extended.

## Where does the configuration file live and what can I configure in it?

See [Configuration file](http://docs.ansible.com/ansible/intro_configuration.html).

## How do I disable cowsay?

If cowsay is installed, Ansible takes it upon itself to make your day happier when running playbooks. If you decide that you would like to work in a professional cow-free environment, you can either uninstall cowsay, or set an environment variable:

export ANSIBLE\_NOCOWS=1

## How do I see a list of all of the ansible\_ variables?

Ansible by default gathers “facts” about the machines under management, and these facts can be accessed in Playbooks and in templates. To see a list of all of the facts that are available about a machine, you can run the “setup” module as an ad-hoc action:

ansible -m setup hostname

This will print out a dictionary of all of the facts that are available for that particular host. You might want to pipe the output to a pager.

## How do I see all the inventory vars defined for my host?

By running the following command, you can see vars resulting from what you’ve defined in the inventory:

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

## How do I loop over a list of hosts in a group, inside of a template?

A pretty common pattern is to iterate over a list of hosts inside of a host group, perhaps to populate a template configuration file with a list of servers. To do this, you can just access the “$groups” dictionary in your template, like this:

{% for host in groups['db\_servers'] %}

{{ host }}

{% endfor %}

If you need to access facts about these hosts, for instance, the IP address of each hostname, you need to make sure that the facts have been populated. For example, make sure you have a play that talks to db\_servers:

- hosts: db\_servers

tasks:

- debug: msg="doesn't matter what you do, just that they were talked to previously."

Then you can use the facts inside your template, like this:

{% for host in groups['db\_servers'] %}

{{ hostvars[host]['ansible\_eth0']['ipv4']['address'] }}

{% endfor %}

## How do I access a variable name programmatically?

An example may come up where we need to get the ipv4 address of an arbitrary interface, where the interface to be used may be supplied via a role parameter or other input. Variable names can be built by adding strings together, like so:

{{ hostvars[inventory\_hostname]['ansible\_' + which\_interface]['ipv4']['address'] }}

The trick about going through hostvars is necessary because it’s a dictionary of the entire namespace of variables. ‘inventory\_hostname’ is a magic variable that indicates the current host you are looping over in the host loop.

## How do I access a variable of the first host in a group?

What happens if we want the ip address of the first webserver in the webservers group? Well, we can do that too. Note that if we are using dynamic inventory, which host is the ‘first’ may not be consistent, so you wouldn’t want to do this unless your inventory is static and predictable. (If you are using [Ansible Tower](http://docs.ansible.com/ansible/tower.html), it will use database order, so this isn’t a problem even if you are using cloud based inventory scripts).

Anyway, here’s the trick:

{{ hostvars[groups['webservers'][0]]['ansible\_eth0']['ipv4']['address'] }}

Notice how we’re pulling out the hostname of the first machine of the webservers group. If you are doing this in a template, you could use the Jinja2 ‘#set’ directive to simplify this, or in a playbook, you could also use set\_fact:

- set\_fact: headnode={{ groups[['webservers'][0]] }}

- debug: msg={{ hostvars[headnode].ansible\_eth0.ipv4.address }}

Notice how we interchanged the bracket syntax for dots – that can be done anywhere.

## How do I copy files recursively onto a target host?

The “copy” module has a recursive parameter. However, take a look at the “synchronize” module if you want to do something more efficient for a large number of files. The “synchronize” module wraps rsync. See the module index for info on both of these modules.

## How do I access shell environment variables?

If you just need to access existing variables, use the ‘env’ lookup plugin. For example, to access the value of the HOME environment variable on the management machine:

---

# ...

vars:

local\_home: "{{ lookup('env','HOME') }}"

If you need to set environment variables, see the Advanced Playbooks section about environments.

Starting with Ansible 1.4, remote environment variables are available via facts in the ‘ansible\_env’ variable:

{{ ansible\_env.SOME\_VARIABLE }}

## How do I generate crypted passwords for the user module?

The mkpasswd utility that is available on most Linux systems is a great option:

mkpasswd --method=sha-512

If this utility is not installed on your system (e.g. you are using OS X) then you can still easily generate these passwords using Python. First, ensure that the [Passlib](https://bitbucket.org/ecollins/passlib/wiki/Home) password hashing library is installed:

pip install passlib

Once the library is ready, SHA512 password values can then be generated as follows:

python -c "from passlib.hash import sha512\_crypt; import getpass; print sha512\_crypt.using(rounds=5000).hash(getpass.getpass())"

Use the integrated [Hashing filters](http://docs.ansible.com/ansible/playbooks_filters.html#hash-filters) to generate a hashed version of a password. You shouldn’t put plaintext passwords in your playbook or host\_vars; instead, use [Vault](http://docs.ansible.com/ansible/playbooks_vault.html) to encrypt sensitive data.

## Can I get training on Ansible?

Yes! See our [services page](https://www.ansible.com/consulting) for information on our services and training offerings. Email [info@ansible.com](mailto:info%40ansible.com) for further details.

We also offer free web-based training classes on a regular basis. See our [webinar page](https://www.ansible.com/webinars-training) for more info on upcoming webinars.

## Is there a web interface / REST API / etc?

Yes! Ansible, Inc makes a great product that makes Ansible even more powerful and easy to use. See [Ansible Tower](http://docs.ansible.com/ansible/tower.html).

## How do I submit a change to the documentation?

Great question! Documentation for Ansible is kept in the main project git repository, and complete instructions for contributing can be found in the docs README [viewable on GitHub](https://github.com/ansible/ansible/blob/devel/docs/docsite/README.md). Thanks!

## How do I keep secret data in my playbook?

If you would like to keep secret data in your Ansible content and still share it publicly or keep things in source control, see [Vault](http://docs.ansible.com/ansible/playbooks_vault.html).

In Ansible 1.8 and later, if you have a task that you don’t want to show the results or command given to it when using -v (verbose) mode, the following task or playbook attribute can be useful:

- name: secret task

shell: /usr/bin/do\_something --value={{ secret\_value }}

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:

- hosts: all

no\_log: True

Though this will make the play somewhat difficult to debug. It’s recommended that this be applied to single tasks only, once a playbook is completed. Note that the use of the no\_log attribute does not prevent data from being shown when debugging Ansible itself via the ANSIBLE\_DEBUG environment variable.

## When should I use {{ }}? Also, how to interpolate variables or dynamic variable names

A steadfast rule is ‘always use {{ }} except when when:‘. Conditionals are always run through Jinja2 as to resolve the expression, so when:, failed\_when: and changed\_when: are always templated and you should avoid adding {{}}.

In most other cases you should always use the brackets, even if previously you could use variables without specifying (like with\_ clauses), as this made it hard to distinguish between an undefined variable and a string.Another rule is ‘moustaches don’t stack’. We often see this:

{{ somevar\_{{other\_var}} }}

The above DOES NOT WORK, if you need to use a dynamic variable use the hostvars or vars dictionary as appropriate:

{{ hostvars[inventory\_hostname]['somevar\_' + other\_var] }}

## Why don’t you ship in X format?

Several reasons, in most cases it has to do with maintainability, there are tons of ways to ship software and it is a herculean task to try to support them all. In other cases there are technical issues, for example, for python wheels, our dependencies are not present so there is little to no gain.

ANSIBLE ROLES

Within Ansible there are two techniques for reusing a set of configuration management tasks, [includes](http://docs.ansible.com/playbooks_roles.html#task-include-files-and-encouraging-reuse) and [roles](http://docs.ansible.com/playbooks_roles.html#roles). Although both techniques function in similar ways, roles appear to be the official way forward. [Ansible Galaxy](https://galaxy.ansible.com/) was built as a repository for roles, and as we’ll see in this post, ansible-galaxy exists to aid in installing and creating them.

# Creating a New Role

Let’s start off by creating a role for [Packer](http://www.packer.io/).

Packer is a useful tool for producing different machine image types with the same set of configuration management tasks. For example, Packer can be used to take a set of Ansible instructions, funnel them through itself, and produce both an AMI and Docker image.

Enough about Packer though, let’s get back to creating an Ansible role for installing Packer.

The first step in creating a role is creating its directory structure. In order to create the base directory structure, we’re going to use a tool bundled with Ansible (since 1.4.2) called ansible-galaxy:

1. $ ansible-galaxy init azavea.packer
2. azavea.packer was created successfully

That command will create an azavea.packer directory with the following structure:

1. ├── README.md
2. ├── defaults
3. │ └── main.yml
4. ├── files
5. ├── handlers
6. │ └── main.yml
7. ├── meta
8. │ └── main.yml
9. ├── tasks
10. │ └── main.yml
11. ├── templates
12. └── vars
13. └── main.yml

# Explaining the Role Directory Structure

A role’s directory structure consists of defaults, vars, files, handlers, meta, tasks, and templates. Let’s take a closer look at each:

### defaults

Within defaults, there is a main.yml file with the default variables used by a role. For the Packer role, there is only a packer\_version default variable. As of this post, the most recent version of Packer is 0.7.1, so we’ll set it to that:

1. ---
2. packer\_version: "0.7.1"

### vars

vars and defaults house variables, but variables in vars have a higher priority, which means that they are more difficult to override. Variables in defaults have the lowest priority of any variables available, which means they’re easy to override. Placing packer\_version in defaults instead of vars is desirable because now it is easier to override when you want to install an older or newer version of Packer:

1. ---
2. - hosts: all
3. sudo: yes
4. roles:
5. - { role: "azavea.packer", packer\_version: "0.7.0" }

All of that said, we’re set with packer\_version in defaults, so the vars directory is not needed either.

### files

files is where you put files that need to be added to the machine being provisioned, without modification. Most of the time, files in files are referenced by copy tasks.

The Packer role has no need for files, so we’ll delete that directory.

### handlers

handlers usually contain targets for notify directives, and are almost always associated with services. For example, if you were creating a role for NTP, you might have an entry in handlers/main.yml for restarting NTP after a task finishes altering the NTP configuration file.

Packer isn’t a service, so there is no need for the handlers directory.

### meta

meta/main.yml houses one of the biggest differences between includes from roles: metadata. The metadata of an Ansible role consists of attributes such as author, supported platforms, and dependencies. Most of this file is commented out by default, so I usually go through and fill in or uncomment relevant attributes, then delete anything else.

For the Packer role, I trimmed things down to:

1. ---
2. galaxy\_info:
3. author: Hector Castro
4. description: An Ansible role for installing Packer.
5. company: Azavea Inc.
6. license: Apache
7. min\_ansible\_version: 1.2
8. platforms:
9. - name: Ubuntu
10. versions:
11. - trusty
12. categories:
13. - cloud
14. - system
15. dependencies:
16. - { role: "azavea.unzip" }

Ignore the dependencies bit for right now. We’ll come back to it later.

### tasks

tasks houses a series of Ansible plays to install, configure, and run software. For Packer, we need to download a specific version, and since it’s packaged as a compiled binary in a ZIP archive, extract it. Accomplishing that with Ansible’s built-in get\_url and unarchive modules looks like this:

1. ---
2. - name: Download Packer
3. get\_url: >
4. url=https://dl.bintray.com/mitchellh/packer/packer\_{{ packer\_version }}\_linux\_amd64.zip
5. dest=/usr/local/src/packer\_{{ packer\_version }}\_linux\_amd64.zip
6. - name: Extract and install Packer
7. unarchive: src=/usr/local/src/packer\_{{ packer\_version }}\_linux\_amd64.zip
8. dest=/usr/local/bin
9. copy=no

### templates

templates is similar to files except that templates support modification as they’re added to the machine being provisioned. Modifications are achieved through the [Jinja2](http://jinja.pocoo.org/docs/dev/) templating language. Most software configuration files become templates.

Packer takes most of its configuration parameters via command-line arguments, so the templates directory is not needed.