ANSIBLE REALTIME QUESTIONS

1. **How to install multiple tools at once with single task, instead of installing as different tasks?**
   with_items:
   Sometimes we want to do many things with single tasks like installing many packages with the same tasks just by changing the arguments. This can be achieved using the with_items clause.
   By using the with_items, ansible creates a temporary variable called {{item}} which consist the value for the current iteration. Let's have some example to understand this. We will install few packages with below playbook.

```
# Installing packages individually (Slower Process)
- name: Installing Git
  apt:
    name: git
    update_cache: yes
- name: Installing nginx
  apt:
    name: nginx
    update_cache: yes
- name: Installing memcached
  apt:
    name: memcached
    update_cache: yes
```

   The above playbook will run 3 tasks each for installing individual package. Rather than specifying three different tasks, we can use with_items and specify the list of packages that we need to install.

```
# Installing Packages with one Task (Faster Process)
- name: Installing Packages
  apt:
    name: "{{ item }}"
    update_cache: yes
  with_items:
    - git
    - nginx
    - Memcached
```

   Here, while executing the task "Installing packages" Ansible will read the list from with_items and install packages one by one. You can also use with_items with roles as well. So, if you have any custom role defined and you want to execute that role multiple times, rather than defining it multiple times you can use with_items and just pass your elements.

2. **run_once**
   There are condition where we have to write our playbook in such a way that will run some tasks or perform some action only on single host from group. If you are thinking of Handlers do the same thing. Think twice because even though multiple tasks notify to perform some action, handlers will only gets executed after all tasks completed in play but on all hosts and not on single.

```
---
- hosts: web
tasks:
  - name: Initiating Database
    script: initalize_wp_database.sh
    run_once: true
```

To achieve this, Ansible provided run_once module, which will run only on single host from group of hosts. By default, Ansible will select the first host from the group to execute. This can also be used with delegate_to to run the task on specific server. When executed with serial, task marked as run_oncegets executed on one host from each batch.

Check whether a package is installed or not. verify that package (wget) is installed on the remote computer and if not install it from the distribution repos.

3. **Install wget**

We want to verify that wget is installed on the remote computer and if not install it from the distribution repos. To install packages, there are modules available, yum and apt (There are others but we will focus on these).
To be able to run the correct task depending on the ansible_pkg_mgr value we can use only_id:

```
- name: Install wget package (Debian based)
  action: apt pkg='wget' state=installed

  only_if: "'$ansible_pkg_mgr' == 'apt'"



- name: Install wget package (RedHat based)
  action: yum name='wget' state=installed

  only_if: "'$ansible_pkg_mgr' == 'yum'"
```

4. **Handling Different Operating Systems with Variables**

I mentioned that variables within a role should not be used, but that's only half the story. Specifically, variables should not be used for parameters – if you are going to define those variables later within a playbook or via group_vars/ or host_vars/, it is better to make them defaults.
But on the other hand, they are better suited as variables if they are not intended for assignment outside of the scope of the role. This is perfect for what we are planning on using them for here.
Variables can be defined in a role pretty easy – see roles. Note that in here there is a vars/directory, from which vars/main.yml is loaded with high priority.
This is not the file I am looking for though, as variable files are not very flexible: no additional files or logic can go into it, just values.
So how can variables be included logically then?
include_vars and with_first_found
include_vars is a module that allows you to include variables from within a task. Perfect for the kind of logic flow we are looking for, and allows you to specify a file based off either a jinja2 template handler or the special {{ item }} token.

That's only part of it though. Logic is needed to handle different operating systems, and can get pretty messy if all that is being used is a bunch of when: clauses. Enter with_first_found:
# tasks/main.yml

- name: set distro-specific variables
  include

To keep your playbooks cleaner, you could do this within the roles, for example, you could have a single tasks/main.yml the one based on the OS it will include the tasks matching the condition used with when, for example, if using ansible_os_family the contents of main.ymlcould be something like:

```
---
- include_tasks: freebsd.yml
  when: ansible_os_family == "FreeBSD"
  tags:
    - unbound

- include_tasks: debian.yml
  when: ansible_os_family == "Debian"
  tags:
    - unbound

- include_tasks: redhat.yml
  when: ansible_os_family == "RedHat"
  tags:
    - unbound
```

Please note that the modules for apt and yum have different options that make it impossible to use the method above of using the action: In order to do this, one must use the when: to perform their associated module per OS.
yum does not have a update module option, so it basically check to see if all packages are up to date.apt upgrades to dist.

```
- name: install system updates for centos systems
  yum: name=* state=latest update_cache=yes
  when: ansible_distribution == "CentOS"

- name: install system updates for ubuntu systems
  apt: upgrade=dist update_cache=yes
  when: ansible_distribution == "Ubuntu"
```

5. **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.

6. **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.

7. **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.

8. **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.

9. **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.

10. **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.

11. **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.

12. **Where does the configuration file live and what can I configure in it?**
    See Configuration file.

13. **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 the ANSIBLE_NOCOWS environment variable:

    ```
    export ANSIBLE_NOCOWS=1
    ```

14. **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.

15. 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
```

16. 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 %}
```

17. 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.

18. 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, 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.

19. 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.

20. 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 }}
```

21. 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 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 to generate a hashed version of a password. You shouldn’t put plaintext passwords in your playbook or host_vars; instead, use Using Vault in playbooks to encrypt sensitive data.

22. 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.

23. 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 Using Vault in playbooks.

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.

24. 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] }}
```

25. 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.