How to Include Variables in Ansible + Examples

Written by Percy Grunwald— Last Updated February 22, 2019

I'm aware of two ways to include variables from another file in Ansible:

1. The `include_vars` module (works in any list of tasks, such as a playbook or role)
2. The `vars_files` keyword (works on plays only)

There are a number of reasons you might want to move variables into separate files when using Ansible:

1. **Refactoring** - group long lists of variables together into smaller files
2. **Conditionally include variables** - you may wish to load different variables based on host-specific properties or in response to tasks

I have included the most common use cases I can think of for `include_vars` and `vars_files` in the sections below.

How to use `include_vars`

The `include_vars` module can be used in a playbook or role to load variables from a file. Simply set the value of `include_vars` to a local file to load the variables it contains:

```
---
# ./hello_world.yml

- name: print greeting
  hosts: "*"
  tasks:
    - include_vars: name_vars.yml

    - debug: msg="Hello, {{ name }}!"
```

`name_vars.yml` can be located in the same directory as `hello_world.yml`, or inside `./vars/` (I prefer to put variable files into `./vars/`). The `./vars/name_vars.yml` file looks like this:

```
---
# ./vars/name_vars.yml

name: World
```

The playbook above will produce the following output:

```
ok: [123.123.123.123] => {
    "msg": "Hello, World!"
}
```

Be careful: any variables you set with `set_fact` will **not** be overwritten with `include_vars` due to the variable precedence of different sources in Ansible.

For example, if you change the tasks file from above to be like this:

```
---
# ./hello_world.yml

- name: print greeting
  hosts: "*"
  tasks:
    - set_fact: name=Percy

    - include_vars: name_vars.yml

    - debug: msg="Hello, {{ name }}!"
```

Despite having `include_vars` **after** `set_fact`, the output will look like this:

```
ok: [123.123.123.123] => {
    "msg": "Hello, Percy!"
}
```

How to include variables conditionally

You can conditionally include variables by using the `when` keyword.

Similar to the example above, you can include variable files based on the OS family of the remote host:

```
---
# ./redis.yml

- name: print redis package
  hosts: "*"
  tasks:
    - include_vars: vars-Debian.yml
      when: ansible_os_family == 'Debian'

    - include_vars: vars-RedHat.yml
      when: ansible_os_family == 'RedHat'

    - debug: var=redis_package
```

Where the variable files look like this:

```
---
# ./vars/vars-Debian.yml

redis_package: redis-server
```

```
---
# ./vars/vars-RedHat.yml

redis_package: redis
```

The debug task from above will print the following:

```
ok: [centos_7] => {
    "redis_package": "redis"
}
ok: [ubuntu_bionic_1804] => {
    "redis_package": "redis-server"
}
```

How to include variables with a dynamic file name

You can make the previous example more terse by calling `include_vars` with a dynamic filename based on the `ansible_os_family` variable:

```
- include_vars: "vars-{{ ansible_os_family }}.yml"

- debug: var=redis_package
```

These tasks will behave in the same way as the previous example:

```
ok: [centos_7] => {
    "redis_package": "redis"
}
ok: [ubuntu_bionic_1804] => {
    "redis_package": "redis-server"
}
```

How to include variables from an Ansible vault file

Including variables from an Ansible Vault file works the same way as including a regular plaintext vars file. A common pattern I use in my playbooks is something like this:

```
---
# ./configure_app_servers.yml

- name: configure app servers
  hosts: app_servers
  pre_tasks:
    - name: include env vars
      include_vars: "{{ env }}.yml"
      tags: ["always"]

    - name: include vault for env
      include_vars: "{{ env }}.vault.yml"
      tags: ["always"]
  roles:
    ...
```

As you can see in the example above, the regular vars file **and** the vault file are included in the same way with `include_vars` . Ansible will automatically detect whether or not the file is an Ansible Vault file and decrypt it accordingly. **This same functionality can also be achieved using `vars_files` on the play**, please see the following sections to see how this is done.

You can set `env` by passing the `--extra-vars` option to `ansible-playbook` :

```
$ ansible-playbook configure_app_servers.yml --extra-vars "env=prod"
```

You can set `env` to one of `dev` , `staging` , `prod` etc. For the `prod` env, you could have variable like this:

```
---
# ./vars/prod.yml

domain: example.com
rails_env: production
https: true
```

The `prod.vault.yml` would look like this in plain text:

```
$ANSIBLE_VAULT;1.1;AES256
34623530653864623831656 26...
...
```

But would print this when viewed with `ansible-vault view ./vars/prod.vault.yml`:

```
---
# ./vars/prod.vault.yml

database_password: "94BqEabtebgzbQItkqPEVMyqjKbp57Gc"
api_key: "xVQKRARKgcwDktPCJjYRFiqmHGsvNFd9"
```

How to include variables for each item in a loop

The example from the previous section could be more terse if you used a loop with `include_vars` rather than a two separate tasks:

```
---
# ./configure_app_servers.yml

- name: configure app servers
  hosts: app_servers
  pre_tasks:
    - name: include vars and vault for env
      include_vars: "{{ item }}.yml"
      tags: ["always"]
      loop:
        - "{{ env }}"
        - "{{ env }}.vault"
  roles:
    ...
```

For `env: prod`, the loop above would include variables from `prod.yml` as well as `prod.vault.yml` just like in the previous section.

How to include variables using `vars_files` on a play

You could further improve the example above by using the `vars_files` keyword on the play. Note that you need to prepend `./vars/` to the file path when using the `vars_file` keyword:

```
---
# ./configure_app_servers.yml

- name: configure app servers
  hosts: app_servers
  vars_files:
    - "./vars/{{ env }}.yml"
    - "./vars/{{ env }}.vault.yml"
  roles:
    ...
```