Sample Ansible setup

You have learned about playbooks, inventory, roles, and variables. This section pulls all those elements together, outlining a sample setup for automating a web service. You can find more example playbooks illustrating these patterns in our ansible-examples repository. (NOTE: These may not use all of the features in the latest release, but are still an excellent reference!).

The sample setup organizes playbooks, roles, inventory, and variables files by function, with tags at the play and task level for greater granularity and control. This is a powerful and flexible approach, but there are other ways to organize Ansible content. Your usage of Ansible should fit your needs, not ours, so feel free to modify this approach and organize your content as you see fit.

Sample directory layout

This layout organizes most tasks in roles, with a single inventory file for each environment and a few playbooks in the top-level directory:

```
production                # inventory file for production servers
staging                   # inventory file for staging environment

group_vars/
   group1.yml             # here we assign variables to particular groups
   group2.yml
host_vars/
   hostname1.yml          # here we assign variables to particular systems
   hostname2.yml

library/                  # if any custom modules, put them here (optional)
module_utils/             # if any custom module_utils to support modules, put them here (optional)
filter_plugins/           # if any custom filter plugins, put them here (optional)

site.yml                  # main playbook
webservers.yml            # playbook for webserver tier
dbservers.yml             # playbook for dbserver tier
tasks/                    # task files included from playbooks
    webservers-extra.yml  # <-- avoids confusing playbook with task files

roles/
    common/               # this hierarchy represents a "role"
        tasks/            #
            main.yml      #  <-- tasks file can include smaller files if warranted
        handlers/         #
            main.yml      #  <-- handlers file
        templates/        #  <-- files for use with the template resource
            ntp.conf.j2   #  <------- templates end in .j2
        files/            #
            bar.txt       #  <-- files for use with the copy resource
            foo.sh        #  <-- script files for use with the script resource
        vars/             #
            main.yml      #  <-- variables associated with this role
        defaults/         #
            main.yml      #  <-- default lower priority variables for this role
        meta/             #
            main.yml      #  <-- role dependencies
        library/          # roles can also include custom modules
        module_utils/     # roles can also include custom module_utils
        lookup_plugins/   # or other types of plugins, like lookup in this case

    webtier/              # same kind of structure as "common" was above, done for the webtier role
    monitoring/           # ""
    fooapp/               # ""
```

Note

By default, Ansible assumes your playbooks are stored in one directory with roles stored in a sub-directory called `roles/`. As you use Ansible to automate more tasks, you may want to move your playbooks into a sub-directory called `playbooks/`. If you do this, you must configure the path to your `roles/` directory using the `roles_path`

setting in ansible.cfg.

Alternative directory layout

Alternatively you can put each inventory file with its `group_vars/host_vars` in a separate directory. This is particularly useful if your `group_vars/host_vars` don't have that much in common in different environments. The layout could look something like this:

```
inventories/
   production/
      hosts               # inventory file for production servers
      group_vars/
         group1.yml       # here we assign variables to particular groups
         group2.yml
      host_vars/
         hostname1.yml    # here we assign variables to particular systems
         hostname2.yml

   staging/
      hosts               # inventory file for staging environment
      group_vars/
         group1.yml       # here we assign variables to particular groups
         group2.yml
      host_vars/
         stagehost1.yml   # here we assign variables to particular systems
         stagehost2.yml

library/
module_utils/
filter_plugins/

site.yml
webservers.yml
dbservers.yml

roles/
    common/
    webtier/
    monitoring/
    fooapp/
```

This layout gives you more flexibility for larger environments, as well as a total separation of inventory variables between different environments. However, this approach is harder to maintain, because there are more files. For more information on organizing group and host variables, see :ref:`splitting_out_vars`.

System Message: ERROR/3 (`D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)sample_setup.rst`, **line 108**); *backlink*

Unknown interpreted text role "ref".

Sample group and host variables

These sample group and host variables files record the variable values that apply to each machine or group of machines. For instance, the data center in Atlanta has its own NTP servers, so when setting up ntp.conf, we should use them:

```
---
# file: group_vars/atlanta
ntp: ntp-atlanta.example.com
backup: backup-atlanta.example.com
```

Similarly, the webservers have some configuration that does not apply to the database servers:

```
---
# file: group_vars/webservers
apacheMaxRequestsPerChild: 3000
apacheMaxClients: 900
```

Default values, or values that are universally true, belong in a file called group_vars/all:

```
---
# file: group_vars/all
ntp: ntp-boston.example.com
backup: backup-boston.example.com
```

If necessary, you can define specific hardware variance in systems in a host_vars file:

```
---
# file: host_vars/db-bos-1.example.com
foo_agent_port: 86
bar_agent_port: 99
```

Again, if you are using :ref:`dynamic inventory <dynamic_inventory>`, Ansible creates many dynamic groups automatically. So a tag like "class:webserver" would load in variables from the file "group_vars/ec2_tag_class_webserver" automatically.

> **System Message: ERROR/3** (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)sample_setup.rst, line 151); *backlink*
>
> Unknown interpreted text role "ref".

Sample playbooks organized by function

With this setup, a single playbook can define all the infrastructure. The site.yml playbook imports two other playbooks, one for the webservers and one for the database servers:

```
---
# file: site.yml
- import_playbook: webservers.yml
- import_playbook: dbservers.yml
```

The webservers.yml file, also at the top level, maps the configuration of the webservers group to the roles related to the webservers group:

```
---
# file: webservers.yml
- hosts: webservers
  roles:
    - common
    - webtier
```

With this setup, you can configure your whole infrastructure by "running" site.yml, or run a subset by running webservers.yml. This is analogous to the Ansible "--limit" parameter but a little more explicit:

```
ansible-playbook site.yml --limit webservers
ansible-playbook webservers.yml
```

Sample task and handler files in a function-based role

Ansible loads any file called `main.yml` in a role sub-directory. This sample `tasks/main.yml` file is simple - it sets up NTP, but it could do more if we wanted:

```
---
# file: roles/common/tasks/main.yml

- name: be sure ntp is installed
  yum:
    name: ntp
    state: present
  tags: ntp

- name: be sure ntp is configured
  template:
    src: ntp.conf.j2
    dest: /etc/ntp.conf
  notify:
    - restart ntpd
  tags: ntp

- name: be sure ntpd is running and enabled
  service:
    name: ntpd
    state: started
    enabled: yes
  tags: ntp
```

Here is an example handlers file. As a review, handlers are only fired when certain tasks report changes, and are run at the end of each play:

```
---
# file: roles/common/handlers/main.yml
- name: restart ntpd
  service:
    name: ntpd
    state: restarted
```

See :ref:`playbooks_reuse_roles` for more information.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)sample_setup.rst, line 230); *backlink***
>
> Unknown interpreted text role "ref".

What the sample setup enables

The basic organizational structure described above enables a lot of different automation options. To reconfigure your entire infrastructure:

```
ansible-playbook -i production site.yml
```

To reconfigure NTP on everything:

```
ansible-playbook -i production site.yml --tags ntp
```

To reconfigure only the webservers:

```
ansible-playbook -i production webservers.yml
```

To reconfigure only the webservers in Boston:

```
ansible-playbook -i production webservers.yml --limit boston
```

To reconfigure only the first 10 webservers in Boston, and then the next 10:

```
ansible-playbook -i production webservers.yml --limit boston[0:9]
ansible-playbook -i production webservers.yml --limit boston[10:19]
```

The sample setup also supports basic ad hoc commands:

```
ansible boston -i production -m ping
ansible boston -i production -m command -a '/sbin/reboot'
```

To discover what tasks would run or what hostnames would be affected by a particular Ansible command:

```
# confirm what task names would be run if I ran this command and said "just ntp tasks"
ansible-playbook -i production webservers.yml --tags ntp --list-tasks

# confirm what hostnames might be communicated with if I said "limit to boston"
ansible-playbook -i production webservers.yml --limit boston --list-hosts
```

Organizing for deployment or configuration

The sample setup models a typical configuration topology. When doing multi-tier deployments, there are going to be some additional playbooks that hop between tiers to roll out an application. In this case, 'site.yml' may be augmented by playbooks like 'deploy_exampledotcom.yml' but the general concepts still apply. Ansible allows you to deploy and configure using the same tool, so you would likely reuse groups and keep the OS configuration in separate playbooks or roles from the app deployment.

Consider "playbooks" as a sports metaphor -- you can have one set of plays to use against all your infrastructure and situational plays that you use at different times and for different purposes.

Using local Ansible modules

If a playbook has a :file:`./library` directory relative to its YAML file, this directory can be used to add Ansible modules that will automatically be in the Ansible module path. This is a great way to keep modules that go with a playbook together. This is shown in the directory structure example at the start of this section.

> **System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)**

(user_guide)sample_setup.rst, line 302); *backlink*

Unknown interpreted text role "file".

System Message: ERROR/3 (D:\onboarding-resources\sample-onboarding-resources\ansible-devel\docs\docsite\rst\user_guide\(ansible-devel)(docs)(docsite)(rst)(user_guide)sample_setup.rst, line 306)

Unknown directive type "seealso".

```
.. seealso::

   :ref:`yaml_syntax`
       Learn about YAML syntax
   :ref:`working_with_playbooks`
       Review the basic playbook features
   :ref:`list_of_collections`
       Browse existing collections, modules, and plugins
   :ref:`developing_modules`
       Learn how to extend Ansible by writing your own modules
   :ref:`intro_patterns`
       Learn about how to select hosts
   `GitHub examples directory <https://github.com/ansible/ansible-examples>`_
       Complete playbook files from the github project source
   `Mailing List <https://groups.google.com/group/ansible-project>`_
       Questions? Help? Ideas?  Stop by the list on Google Groups
```