Ansible Fundamentals

Working with Roles for Automation Reuse

Objectives

This module:

- Explains what a role is, how it is structured, and how you can use it in a playbook.
- Describes how to create a role in a playbook's project directory and run it as part of one of the plays in the playbook.
- Explains how to select and retrieve roles from Ansible Galaxy and use them in playbooks.

Creating Roles

Red Hat

Creating Roles

- Ansible **roles** allow you to make automation code more reusable.
- Provides packaged tasks that can be configured through variables.
- The playbook just calls the role and passes it the right values through its variables.
- Allows you to create generic code for one project and reuse it on other projects.

Benefits of Ansible Roles

- Roles group content, allowing easy sharing of code with others.
- Roles can be written in a way that define the essential elements of a system type: web server, database server, Git repository, and more.
- Roles make larger projects more manageable.
- Different administrators can develop roles in parallel.

Creating Ansible Roles

- You can write a role using the same tools you use to write playbooks
- Creating and using a role is a three step process:
  1. Create the role directory structure.
  2. Define the role content.
  3. Use the role in a play.
- One way to create a role is to start by writing a play and then refactoring it into a generic role.
- Do not store secrets in a role. Pass them as parameters from the play.

Creating the Role's Directory Structure

- Each role has its own directory with a standardized folder structure.
- The top-level directory defines the name of the role itself.
- Files are organized into subdirectories named according to the purpose of each file in the role, such as **tasks** and **handlers**.
- You can manually create the directory structure.
- On Linux, the **ansible-galaxy init *rolename*** command can create the "skeleton" directory for you.

Creating the Role Skeleton

```
role_example
├── defaults
│   └── main.yml
├── files
├── handlers
│   └── main.yml
├── meta
│   └── main.yml
├── README.md
├── tasks
│   └── main.yml
├── templates
├── tests
│   ├── inventory
│   └── test.yml
└── vars
    └── main.yml
```

Directory	Function
defaults	Default values of role variables that can be overwritten when the role is used. These variables have low precedence and are intended to be changed and customized by plays.
files	Static files that are referenced by role tasks.
handlers	The handler definitions used by the role.
meta	The **main.yml** file in this directory contains information about the role, including author, license, platforms, and optional role dependencies.
tasks	Tasks performed by the role, similar to a play's tasks section.
templates	Jinja2 templates that are referenced by role tasks.
tests	This directory can contain an inventory and **test.yml** playbook that can be used to test the role.
vars	Defines values of variables used internally by the role. These variables have high precedence (therefore difficult to override), and are not intended to be changed by the play.

Starting From a Playbook

- At right is an example of a simplified play to create an FTP server on all systems in the inventory group **ftpservers**
- The name of the package and configuration file are hard coded into the play

- A real play would probably have more tasks
  - A handler on **template** to restart vsftpd if the configuration changed and vsftpd was already running
  - Settings to open the firewall and so on

```
- name: Play to install vsftpd
  hosts: ftpservers
  tasks:
    - name: Software is installed
      yum:
        name: vsftpd
        state: present

    - name: Configuration file is installed
      template:
        src: vsftpd.conf.j2
        dest: /etc/vsftpd/vsftpd.conf
        owner: root
        group: root
        mode: '0600'
        setype: etc_t

    - name: Service is running
      service:
        name: vsftpd
        state: started
        enabled: yes
```

Use Variables as Parameters

- The play has now been rewritten so that variables control the name of the package to install, the service to start, and the location of the template and configuration file
- This allows these values to be easily changed

```
- name: Play to install vsftpd
  hosts: ftpservers
  vars:
    ftp_package: vsftpd
    ftp_service: vsftpd
    ftp_config_src: vsftpd.conf.j2
    ftp_config_dest: /etc/vsftpd/vsftpd.conf
  tasks:
    - name: Software is installed
      yum:
        name: "{{ ftp_package }}"
        state: present

    - name: Configuration file is installed
      template:
        src: "{{ ftp_config_src }}"
        dest: "{{ ftp_config_dest }}"
        owner: root
        group: root
        mode: '0600'
        setype: etc_t

    - name: Service is running
      service:
        name: "{{ ftp_service }}"
        state: started
        enabled: yes
```


Red Hat

Defining the Role Content

- Create a new directory for your role with the directories you need for this role.
- We will only need **meta**, **tasks**, **templates**, and **defaults** directories, and a **README.md** file, in the role's directory for this example.
- We will put this in a **roles** directory in the same place as the existing playbook for now so that we can test it later.
- The contents of the **templates** directory (formerly in **project**) which will be used by the role have been moved into the **templates** directory for the role. Files would go into a **files** directory (not shown here).

```
project/
├── playbook.yml
├── templates
└── roles
    └── vsftpd_server
        ├── defaults
        │   └── main.yml
        ├── meta
        │   └── main.yml
        ├── README.md
        ├── tasks
        │   └── main.yml
        └── templates
            ├── vsftpd.conf.j2
            └── vsftpd_special.conf.j2
```

Defining the Role's Tasks

- Copy the tasks from your playbook into the **tasks/main.yml** file.
- Lines that start with **#** are comments.
- Indentation just needs to be consistent.

```
# tasks file for vsftpd_server role

    - name: Software is installed
      yum:
        name: "{{ ftp_package }}"
        state: present

    - name: Configuration file is installed
      template:
        src: "{{ ftp_config_src }}"
        dest: "{{ ftp_config_dest }}"
        owner: root
        group: root
        mode: '0600'
        setype: etc_t

    - name: Service is running
      service:
        name: "{{ ftp_service }}"
        state: started
        enabled: yes
```


Red Hat

Defining the Role's Defaults

- Copy the variables from your playbook into the **defaults/main.yml** file.
- This will set the default values for the role if no settings are specified.
- These variables can be overridden with different values when you call the role from a play.
- If you put them in the **vars/main.yml** file, they will be very hard to override, do not do this for parameters that should be adjustable.

```
# defaults file for vsftpd_server role

ftp_package: vsftpd
ftp_service: vsftpd
ftp_config_src: vsftpd.conf.j2
ftp_config_dest: /etc/vsftpd/vsftpd.conf
```


Red Hat

Documenting the Role

- Create a **meta/main.yml** file.
- This will include some basic information about this role. A simple example is at right.
- Look at other roles and the Ansible documentation for more complex examples.
- You may also create a **README.md** file in Markdown format as documentation for your role. See examples from roles at https://galaxy.ansible.com/

```
galaxy_info:
  author: your name
  description: your role description
  company: your company (optional)

  # This is an open source role
  license: MIT

  # You may specify minimum supported version
  # of Ansible that works for this role
  min_ansible_version: 2.8
```

Using a Role in a Playbook

```
- name: Play to create shared folder
  hosts: ftpservers

  roles:
    - vsftpd_server
```

- An easy way to call a role in a play is to list it in a roles section.
- This assumes the role's directory has been copied into the playbook's **roles** directory.
- This play calls the **vsftpd_server** role from the example in this presentation.
- Because no variables are specified, the role is applied with its default values.
- This combination does exactly what the original playbook did.
- Note that there are no tasks on this play. It can have tasks, but the roles run first.

Using a Role with Custom Parameters

```
- name: Play to create shared folder
  hosts: ftpservers

  roles:
    - vsftpd_server

    - role: vsftpd_server
      vars:
        ftp_config_src: vsftpd_special.conf.j2
```

- In this example play, the role is called twice.
- The first time it is called with its default options and creates the default directory and group.
- The second time it overrides the role's default variables and uses a different Jinja2 template for the configuration file.

Using a Role in a Playbook

```
- name: Play to create shared folder
  hosts: ftpservers

  tasks:
    - name: Execute role
      include_role:
        name: vsftpd_server

    - name: Role with non-default parameters
      include_role:
        name: vsftpd_server
      vars:
        ftp_config_src: vsftpd_special.conf.j2
```

- As an alternative, you can call the role as a task at any time by using the **include_role** module.
- This syntax lets you mix roles with normal tasks in the play.

Using Roles with Ansible Galaxy

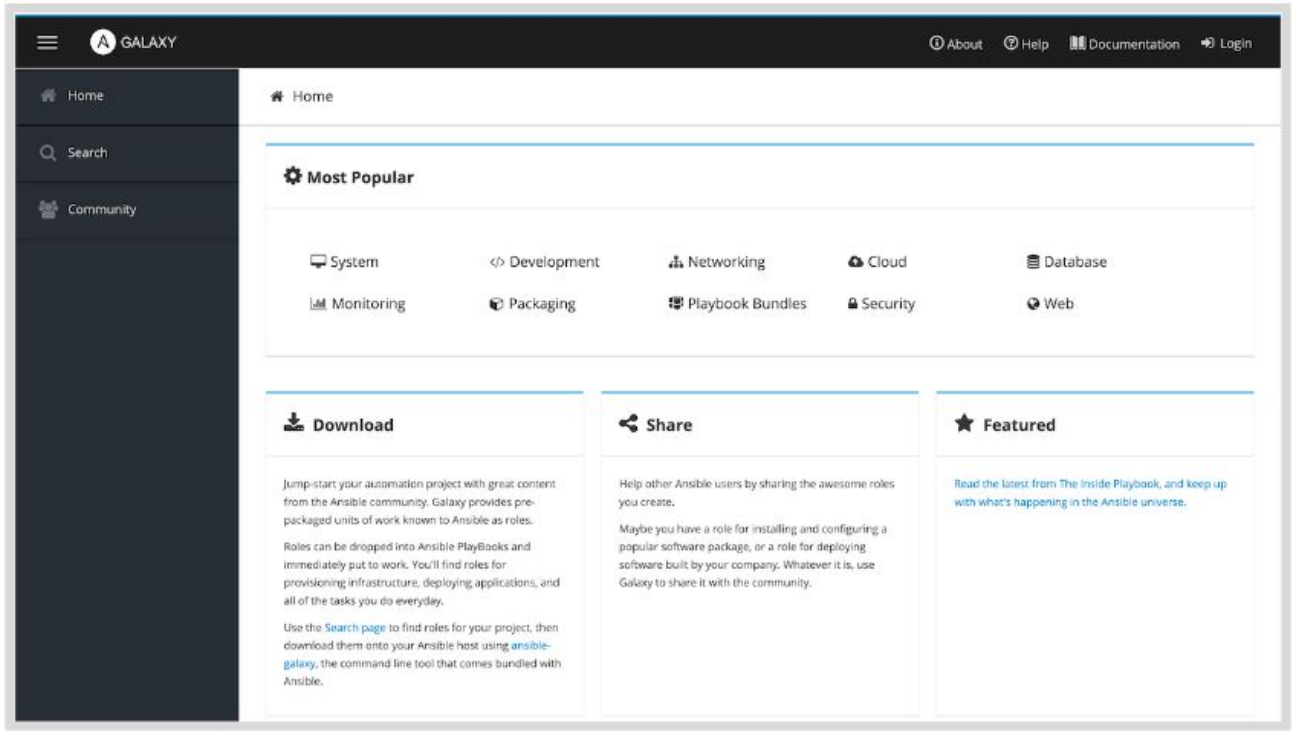
Red Hat

Obtaining and Using Roles

- Normally, roles are kept in their own Git repository separately from the playbook
- This helps avoid each playbook having a private copy of the role that has local edits
- But the role has to be available to the playbook when the playbook is run
- You might want to use your own roles, or reuse roles written by the open source community

About Ansible Galaxy

- https://galaxy.ansible.com
- Public library of Ansible content
- Written by a community of Ansible administrators and users
- Searchable database
- Links to documentation and videos for new Ansible user and role developers.
- Not officially supported by Red Hat, roles may have varying quality levels.

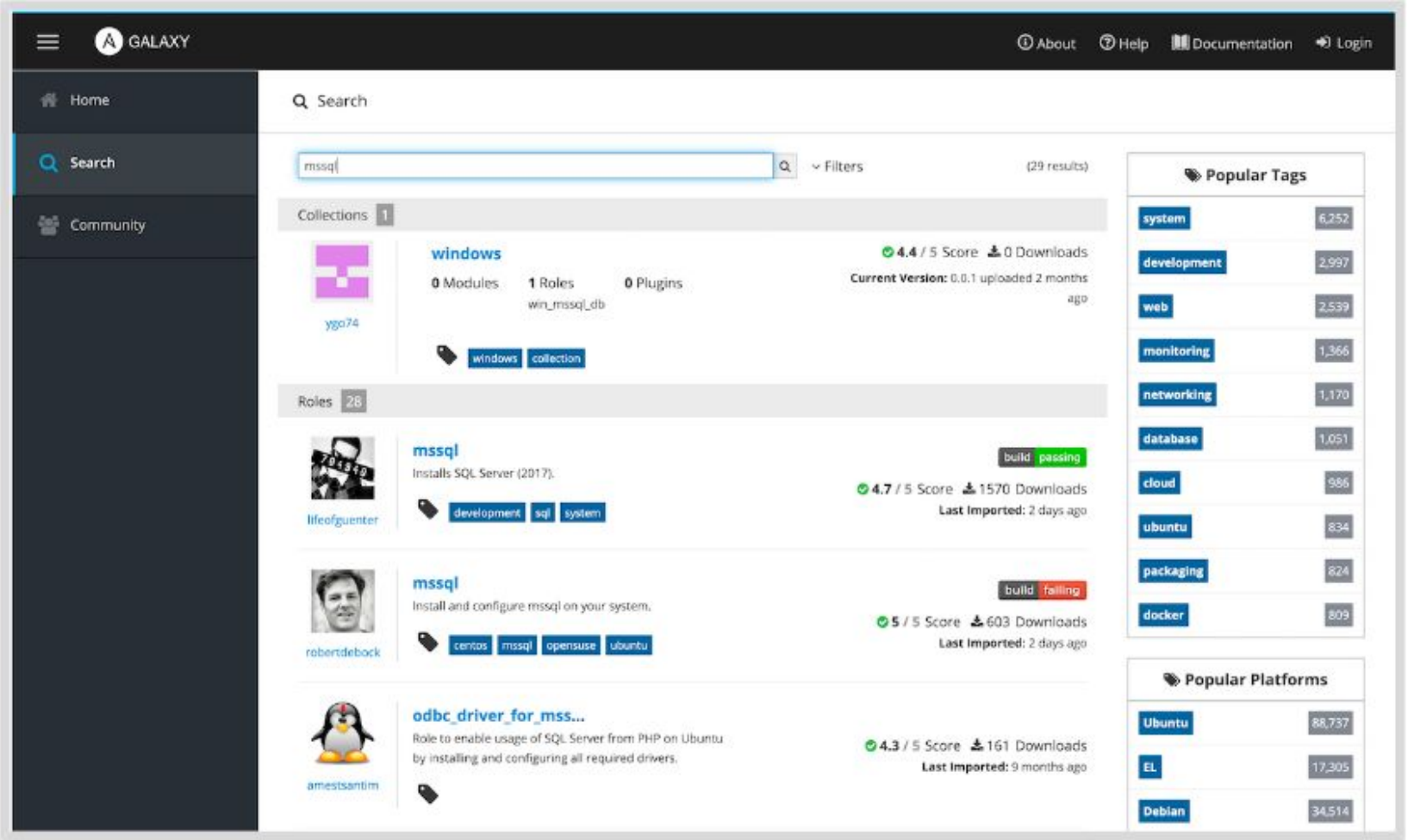


Red Hat

Introducing Ansible Galaxy

- Getting Help with Ansible Galaxy
  - **Documentation** tab on the Ansible Galaxy website home page
  - Provides information about downloading and implementing roles from Ansible Galaxy
  - Instructions on how to develop and upload roles to Ansible Galaxy
- Browsing Ansible Galaxy for Roles
  - The search tab gives users access to information about the roles published on Ansible Galaxy. You can search for an Ansible role by its name, tags, or other role attributes.
  - Many of the roles on Ansible Galaxy are designed for other operating systems or network devices. For example, you can use the **Search** tab to find Microsoft Windows-compatible roles.
  - Results are presented in descending order, based on the **Best Match** score.

Introducing Ansible Galaxy



Red Hat

Ansible Galaxy Command-Line Tool

- The **`ansible-galaxy`** command line tool can be used to search for, display information about, install, list, remove or initialize roles.
- **`ansible-galaxy search`** searches for roles.
  - Use the **`--author`**, **`--platforms`**, and **`--galaxy-tags`** options to narrow search results.
- **`ansible-galaxy info`** displays more detailed information about a role.
- **`ansible-galaxy install`** downloads a role from Ansible Galaxy and install it.
  - By default roles are installed into the first directory that is writable in the user's **`role_path`**. Use the **`-p`** option to specify a different directory to install the role.

Some Role Security Considerations

- You do not have to use Ansible Galaxy to store your roles
- You might want to keep certain roles private and store them in a private Git repository
- It is important to never put sensitive data like passwords in a role itself
- Sensitive data should be set through variables passed to the role by the play

Installing Roles Using a Requirements File

- You can install a list of roles for a project based on definitions in a text file.
- If your playbook requires specific roles, create a **roles/requirements.yml** file in the project directory
- That file is a YAML list of roles to install
- For each role
  - Use the **name** keyword to override the local name of the role.
  - Use the **version** keyword to specify the version of the role.
  - The **src** attribute specifies the source of the role.

- The **requirements.yml** entry at right downloads and installs version 1.6.0 of the **geerlingguy.redis** role from Ansible Galaxy

```
- src: geerlingguy.redis
  version: "1.6.0"
```

Installing Roles Using a Requirements File

- The **ansible-galaxy install** command can be used to download the roles listed in your roles/requirements.yml file:

```
[user@host project]$ ansible-galaxy install -r roles/requirements.yml \
> -p roles
- downloading role 'redis', owned by geerlingguy
- downloading role from https://github.com/geerlingguy/ansible-role-redis/
archive/1.6.0.tar.gz
- extracting geerlingguy.redis to /opt/project/roles/geerlingguy.redis
- geerlingguy.redis (1.6.0) was installed successfully
```

- See https://galaxy.ansible.com/docs/using/installing.html for more examples.

Installing Roles Using a Requirements File

Here are four examples from a **roles/requirements.yml** file:

1. Grabs the latest version of **geerlingguy.redis** from Ansible Galaxy

2. Gets a specific version of **geerlinguy.redis** from Ansible Galaxy. This is a better practice to avoid unexpected changes.

3. Gets a role from a Git repository and selects a specific commit. It also renames the role locally.

4. Gets a role from a Git repository using SSH and selects the latest version on a specific branch.

```
# from Ansible Galaxy, using the latest version
- src: geerlingguy.redis

# from Ansible Galaxy, specific version
- src: geerlingguy.redis
  version: "1.6.0"

# from a Git repo using HTTPS and selecting a specific commit
- src: https://gitlab.example.com/automation/shared_directory.git
  scm: git
  version: 56e00a54
  name: linux_shared_directory

# from a Git repo using SSH and selecting the master branch
- src: git@gitlab.example.com:automation/shared_directory.git
  scm: git
  version: master
```


Red Hat

Managing Downloaded Roles

- The **ansible-galaxy list** subcommand list the roles found locally:

```
[user@host project]$ ansible-galaxy list
- geerlingguy.redis, 1.6.0
- myrole, (unknown version)
- nginx-acme, 56e00a54
- nginx-acme-ssh, master
- redis_prod, 1.5.0
```

- The **ansible-galaxy remove** subcommand removes a local role:

```
[user@host ~]$ ansible-galaxy remove nginx-acme-ssh
- successfully removed nginx-acme-ssh
```