Automation

Ansible Roles

DE HOGESCHOOL
MET HET NETWERK

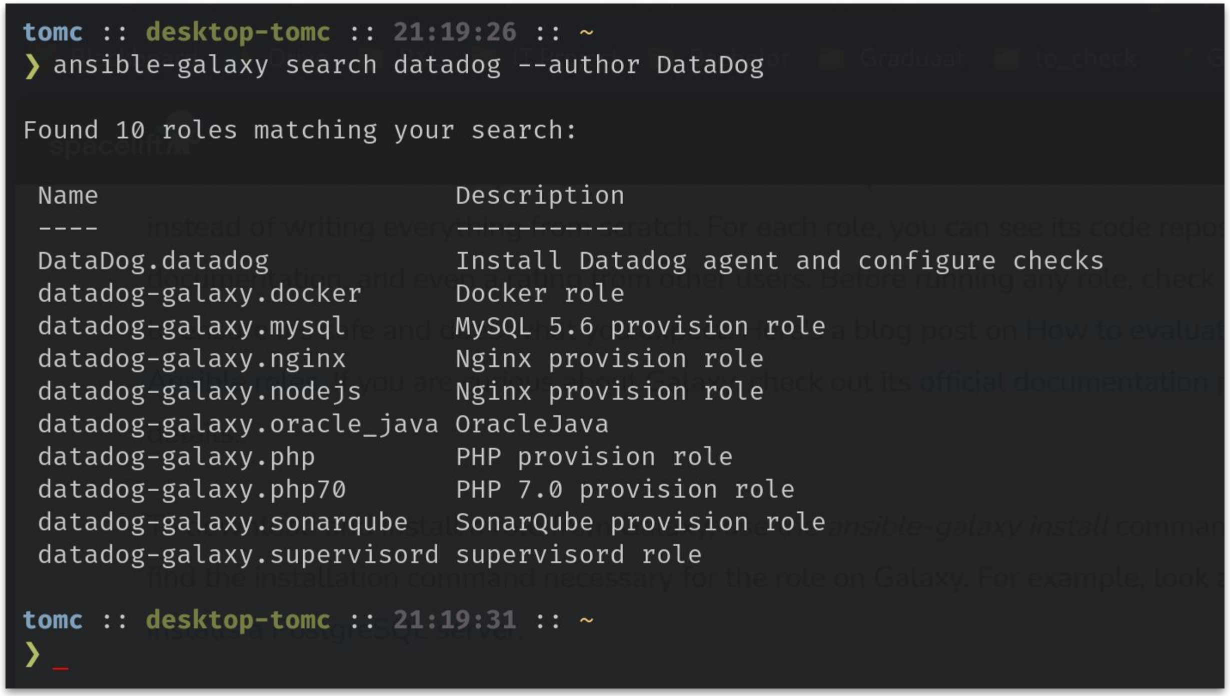
Elfde-Liniestraat 24, 3500 Hasselt, www.pxl.be

ansible-galaxy

- `ansible-galaxy` is a CLI tool that helps manage Ansible roles and their integration with the Ansible Galaxy platform.
  - provides a standardized way to create, search, install, and remove roles both locally and from the Ansible Galaxy platform.
  - simplifies the process of working with roles and promotes reusability and collaboration within the Ansible community.
  - enables users to interact with the Ansible Galaxy platform to download roles or publish their own roles for the community to use.

Integrating with Ansible Galaxy platform

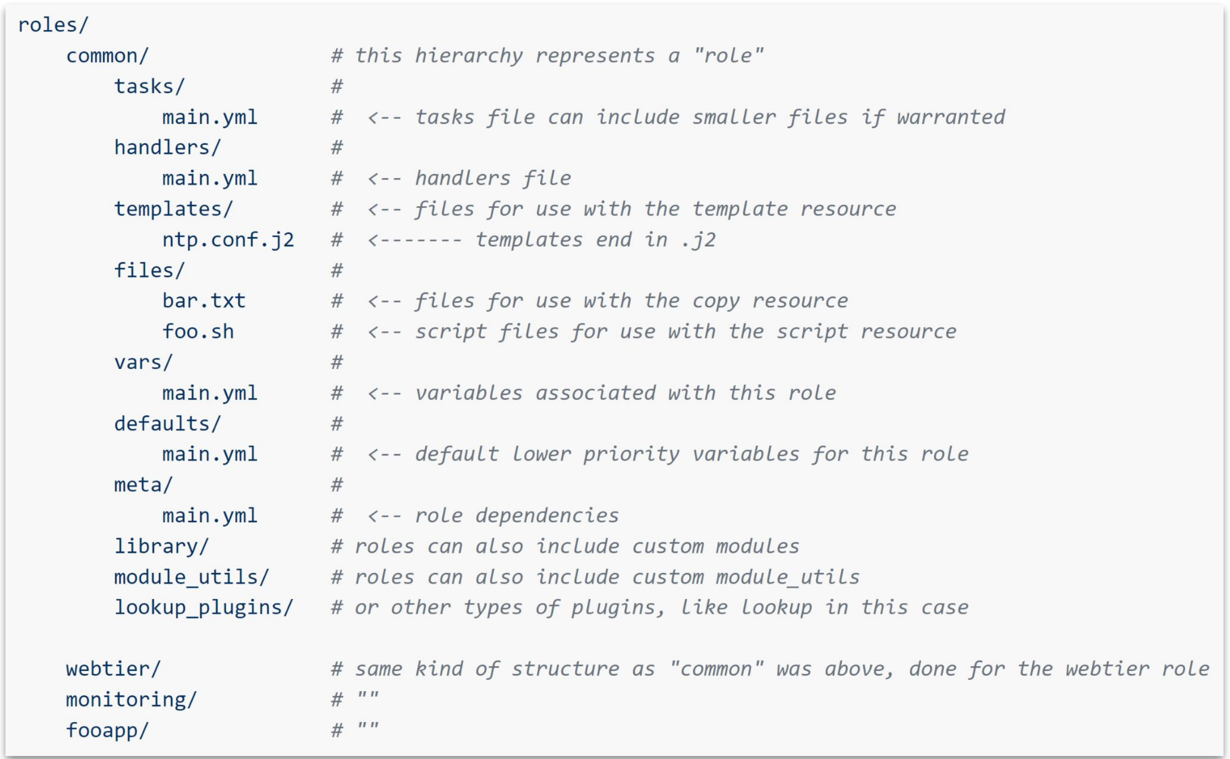
- The ansible-galaxy command-line tool allows users to seamlessly integrate with the Ansible Galaxy platform, a public repository for sharing and discovering Ansible roles. The platform encourages collaboration and reusability within the Ansible community.
- To search for roles use the following command:
  - **`ansible-galaxy search [options] keyword`**
  - This will display a list of roles that match the specified keyword, along with their descriptions, download counts, and ratings.



```
tomc :: desktop-tomc :: 21:19:26 :: ~
❯ ansible-galaxy search datadog --author DataDog

Found 10 roles matching your search:

 Name                          Description
 ----                          -----------
 DataDog.datadog               Install Datadog agent and configure checks
 datadog-galaxy.docker         Docker role
 datadog-galaxy.mysql          MySQL 5.6 provision role
 datadog-galaxy.nginx          Nginx provision role
 datadog-galaxy.nodejs         Nginx provision role
 datadog-galaxy.oracle_java    OracleJava
 datadog-galaxy.php            PHP provision role
 datadog-galaxy.php70          PHP 7.0 provision role
 datadog-galaxy.sonarqube      SonarQube provision role
 datadog-galaxy.supervisord    supervisord role

tomc :: desktop-tomc :: 21:19:31 :: ~
❯ _
```

Ansible Roles

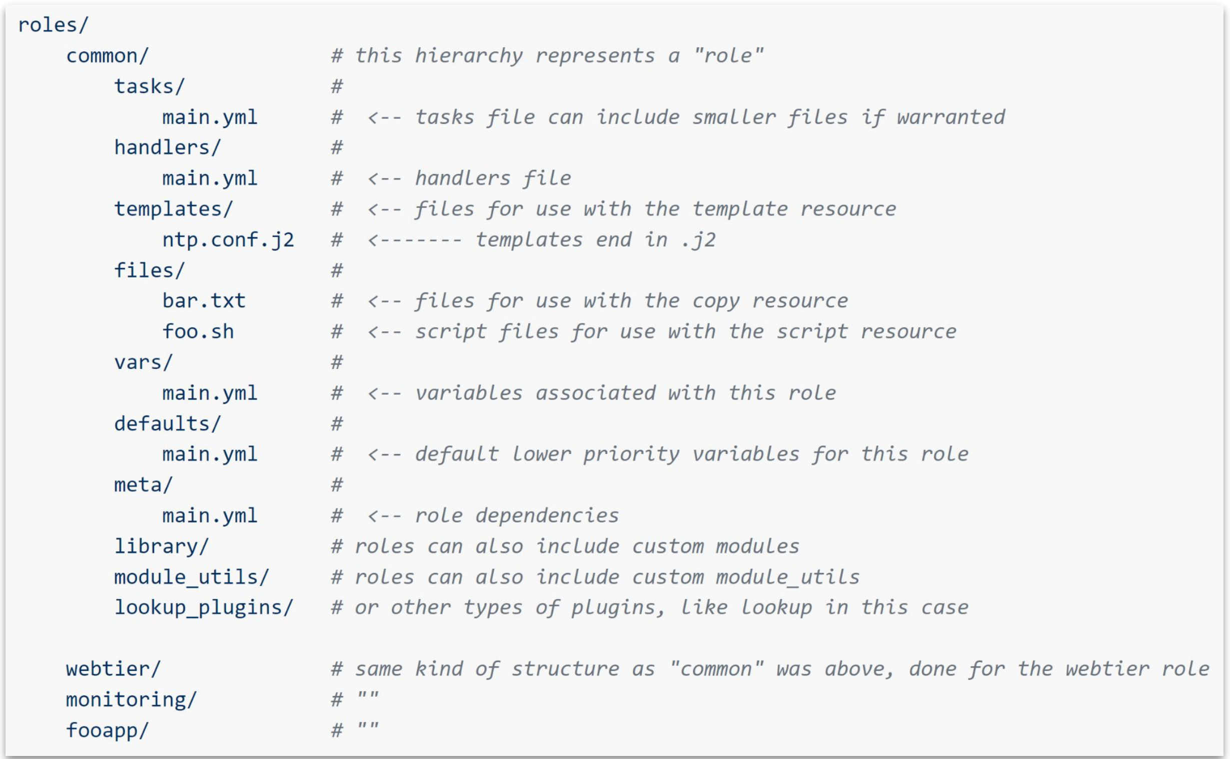
- Roles let you automatically load related vars, files, tasks, handlers, and other Ansible artifacts based on a known file structure.
- After you group your content in roles, you can easily reuse them and share them with other users.
- An Ansible role has a defined directory structure with standard directories. You must **include at least one of these directories** in each role. You can omit any directories the role does not use.



```
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/               # ""
```

Directory structure of Ansible Roles

- `tasks/main.yml` - the main list of tasks that the role executes.
- `handlers/main.yml` - handlers, which may be used within or outside this role.
- `library/my_module.py` - modules, which may be used within this role.
- `defaults/main.yml` - default variables for the role. These variables have the lowest priority of any variables available, and can be easily overridden by any other variable, including inventory variables.
- `vars/main.yml` - other variables for the role.
- `files/main.yml` - non-templated files that the role deploys.
- `templates/main.yml` - templates that the role deploys.
- `meta/main.yml` - metadata for the role, including role dependencies and optional Galaxy metadata such as platforms supported.



```
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/               # ""
```

Directory structure

- You can add other YAML files in some directories. For example, you can place platform-specific tasks in separate files and refer to them in the `tasks/main.yml` file.

```yaml
# roles/example/tasks/main.yml
- name: Install the correct web server for RHEL
  import_tasks: redhat.yml
  when: ansible_facts['os_family']|lower == 'redhat'

- name: Install the correct web server for Debian
  import_tasks: debian.yml
  when: ansible_facts['os_family']|lower == 'debian'
```

```yaml
# roles/example/tasks/redhat.yml
- name: Install web server
  ansible.builtin.yum:
    name: "httpd"
    state: present
```

```yaml
# roles/example/tasks/debian.yml
- name: Install web server
  ansible.builtin.apt:
    name: "apache2"
    state: present
```

Using roles

You can use roles in three ways:

- at the play level with the **`roles`** option: This is the classic way of using roles in a play.
- at the tasks level with **`include_role:`** You can reuse roles dynamically anywhere in the **`tasks:`** section of a play using **`include_role`**.
- at the tasks level with **`import_role:`** You can reuse roles statically anywhere in the **`tasks:`** section of a play using **`import_role`**.

Using roles at the play level

When you use the roles option at the play level, for each role 'x', if the following files exist:

- `roles/x/tasks/main.yml`, adds the tasks in that file to the play.
- `roles/x/handlers/main.yml`, adds the handlers in that file to the play.
- `roles/x/vars/main.yml` exists, adds the variables in that file to the play.
- `roles/x/defaults/main.yml` exists, adds the variables in that file to the play.
- `roles/x/meta/main.yml` exists, adds any role dependencies in that file to the list of roles.
- Any copy, script, template or include tasks (in the role) can reference files in `roles/x/{files,templates,tasks}/` (directory depends on task) without having to path them relatively or absolutely.

```yaml
---
- hosts: webservers
  roles:
    - common
    - webservers
```

Using roles at the play level

- Roles added in a roles section run before any other tasks in a play.
- You can define variables for roles.
- When you add a tag to the role option, Ansible applies the tag to ALL tasks within the role.
- When using `vars:` within the `roles:` section of a playbook, the variables are added to the play variables, making them available to all tasks within the play, before and after the roles.

```yaml
---
- hosts: webservers
  roles:
    - common
    - role: foo_app_instance
      vars:
        dir: '/opt/a'
        app_port: 5000
      tags: typeA
    - role: foo_app_instance
      vars:
        dir: '/opt/b'
        app_port: 5001
      tags: typeB
```

Including roles: dynamic reuse

- You can reuse roles dynamically anywhere in the tasks section of a play using **`include_role`**.
- Included roles run in the order they are defined. If there are other tasks before an **`include_role`** task, the other tasks will run first.
- You can pass other keywords, including variables and tags, when including roles.
- When using **`include_role`** in a playbook or another role, Ansible will execute the tasks defined in the included role as a separate play within the playbook or role. This means that the tasks defined in the included role are not available directly within the playbook

```yaml
---
- hosts: webservers
  tasks:
    - name: Print a message
      ansible.builtin.debug:
        msg: "this task runs before the example role"

    - name: Include the example role
      include_role:
        name: example
      vars:
        dir: '/opt/a'
        app_port: 5000
      tags: typeA
      when: "ansible_facts['os_family'] == 'RedHat'"

    - name: Print a message
      ansible.builtin.debug:
        msg: "this task runs after the example role"
```

Including roles: static reuse

- You can reuse roles statically anywhere in the tasks section of a play using `import_role`.
- Included roles run in the order they are defined. If there are other tasks before an `import_role` task, the other tasks will run first.
- You can pass other keywords, including variables and tags, when importing roles.
- `import_role` is used to import a role and its tasks, handlers, variables, and defaults into a playbook or another role. Unlike `include_role`, import_role allows you to use the tasks and handlers from the imported role directly within the playbook or role, as if they were defined locally.

```yaml
---
- hosts: webservers
  tasks:
    - name: Print a message
      ansible.builtin.debug:
        msg: "this task runs before the example role"

    - name: Include the example role
      import_role:
        name: example
      vars:
        dir: '/opt/a'
        app_port: 5000
      tags: typeA
      when: "ansible_facts['os_family'] == 'RedHat'"

    - name: Print a message
      ansible.builtin.debug:
        msg: "this task runs after the example role"
```

Customizing Role Metadata

Role metadata provides information about the role, such as author details, dependencies, and role version. Customizing role metadata is important for proper documentation, collaboration, and maintenance. Role metadata is stored in the `meta/main.yml` file within the role directory.

`galaxy_info:`

- `author:` The role author's name.
- `description:` A brief description of the role.
- `company:` The company or organization associated with the role (optional).
- `license:` The license under which the role is distributed.
- `min_ansible_version:` The minimum version of Ansible required for the role to function.
- `platforms:` A list of supported platforms for the role (e.g., operating systems, distributions).
- `galaxy_tags:` A list of keywords that help users find your role on Ansible Galaxy.

Customizing Role Metadata

There are two ways to specify dependencies:

By role name: list the role names as items in the dependencies list. Ansible will search for the role in the configured roles path or on Ansible Galaxy.

Using a dictionary format: If you need to provide more details about the dependency, such as specifying a particular version or source.

When your role is included in a playbook, and the playbook is executed, Ansible will first check and install the dependencies, if not already installed, and then execute them in the order listed before running your role.

```yaml
dependencies:
  - yourusername.common
  - yourusername.ssl
```

```yaml
role: The name of the dependent role.
version: The desired version of the dependent role (if
applicable).
src: The source location of the dependent role, such as
a GitHub repository URL.
dependencies:
  - role: yourusername.common
  - role: yourusername.ssl
    version: 1.2.3
    src:
https://github.com/yourusername/ansible-ssl.git
```

Defining Role Variables

Global variables: defined outside any task or block, in vars/main.yml or vars/ directory.

Task-specific variables: defined within a task or block using vars keyword.

```yaml
# vars/main.yml
nginx_port: 80
nginx_server_name: example.com
nginx_root_dir: /var/www/html
```

```yaml
# tasks/main.yml
- name: Install Nginx
  apt:
    name: nginx
    state: present

- name: Copy Nginx configuration file
  template:
    src: templates/nginx.conf.j2
    dest: /etc/nginx/nginx.conf
  vars:
    port: "{{ nginx_port }}"
    server_name: "{{ nginx_server_name }}"
    root_dir: "{{ nginx_root_dir }}"
```

Role Tasks

- Tasks define the actions of an Ansible role and are defined in tasks/main.yml file within the role directory.
- Best practices for writing role tasks:
    - Keep tasks small and focused.
    - Use idempotent tasks.
    - Use conditionals sparingly.
    - Use loops for repetitive tasks.

```yaml
# tasks/main.yml
- name: Copy Nginx configuration file
  template:
    src: templates/nginx.conf.j2
    dest: /etc/nginx/nginx.conf
  vars:
    port: "{{ nginx_port }}"
    server_name: "{{ nginx_server_name }}"
    root_dir: "{{ nginx_root_dir }}"
  notify: restart nginx
```

Role Handlers

- Handlers are defined in handlers/main.yml file within the role directory and are triggered by another task.
- Handlers are useful when a task needs to be executed only if changes have been made.
- Best practices for using role handlers:
  - Use handlers for specific actions.
  - Use unique handler names.
  - Use notify to trigger handlers only when changes have been made.

```yaml
# handlers/main.yml
- name: restart nginx
  service:
    name: nginx
    state: restarted
```

Adding templates and files to Roles

- Ansible roles can include templates and files in addition to tasks and handlers.
- Templates are used to generate dynamic configuration files based on variables, while files are used to store static files.
- Best practices for adding templates and files to roles:
  - Organize templates and files in separate directories within the role directory.
  - Use relative paths when referencing files or templates.
  - Use templates for dynamic configuration.
  - Use files for static files.

```
roles/
├── myrole/
|    ├── tasks/
|    |    ├── main.yml
|    ├── handlers/
|    |    ├── main.yml
|    ├── templates/
|    |    ├── nginx.conf.j2
|    ├── files/
|    |    ├── script.sh
```

Examples of role-based deployments

```
roles/
├── web_servers/
│   ├── tasks/
│   │   ├── main.yml
│   ├── handlers/
│   │   ├── main.yml
│   ├── templates/
│   │   ├── nginx.conf.j2
│   ├── files/
│   │   ├── index.html
├── app_servers/
│   ├── tasks/
│   │   ├── main.yml
│   ├── handlers/
│   │   ├── main.yml
│   ├── templates/
│   │   ├── app.conf.j2
│   ├── files/
│   │   ├── app.jar
├── database_servers/
│   ├── tasks/
│   │   ├── main.yml
│   ├── handlers/
│   │   ├── main.yml
│   ├── templates/
│   │   ├── my.cnf.j2
```

```
roles/
├── web_servers/
│   ├── tasks/
│   │   ├── main.yml
│   ├── handlers/
│   │   ├── main.yml
│   ├── templates/
│   │   ├── nginx.conf.j2
├── database_servers/
│   ├── tasks/
│   │   ├── main.yml
│   ├── handlers/
│   │   ├── main.yml
│   ├── templates/
│   │   ├── my.cnf.j2
├── monitoring_agents/
│   ├── tasks/
│   │   ├── main.yml
│   ├── handlers/
│   │   ├── main.yml
│   ├── files/
│   │   ├── agent.jar
```

Installing and managing roles

- With the ansible-galaxy command-line tool, users can easily manage Ansible roles by installing, updating, or removing them. This simplifies the process of working with roles and helps maintain a consistent and up-to-date role library.
- Installing roles: To install a role from the Ansible Galaxy platform or a custom source, use the following command:
  - **`ansible-galaxy install [options] role_name`**
  - You can also specify a specific version, source repository, or a requirements file containing a list of roles.
- Updating roles: To update an already installed role, you can use the --force flag when installing the role. This will overwrite the existing role with the latest version or the specified version:
  - **`ansible-galaxy install --force [options] role_name`**
- Listing installed roles: To view a list of currently installed roles on your system, use the following command:
  - **`ansible-galaxy list`**
  - This will display the role names along with their installed versions and paths.
- Removing roles: To remove an installed role, use the following command:
  - **`ansible-galaxy remove role_name`**
  - This command will delete the specified role from your local role library.

Initializing a New Role with ansible-galaxy

`ansible-galaxy init [options] role_name`

- This command creates a new directory named role_name and generates a standard role directory structure with default files and folders.

Options:

- **`-p, --roles-path:`** Specify a custom path for the new role directory. By default, the role will be created in the current working directory.
  - **`ansible-galaxy init -p /path/to/roles role_name`**
- **`--init-path:`** Deprecated in favor of the --roles-path option.
- **`--force:`** Overwrite an existing role directory with the same name, if it exists.
  - **`ansible-galaxy init --force role_name`**
- **`--offline:`** Don't query the Ansible Galaxy API when creating a new role.
- **`--role-skeleton:`** Specify a custom role skeleton directory to use as a template when initializing the new role. By default, ansible-galaxy uses the role skeleton included with Ansible.
  - **`ansible-galaxy init --role-skeleton /path/to/skeleton role_name`**
- **`-v, --verbose:`** Display more detailed information during the initialization process.

Role Naming conventions

- Use lowercase letters and underscores
  - Easy to read, understand, and compatible
  - Example: `web_server` instead of Web-Server
- Be descriptive and concise
  - Accurately describe the role's purpose
  - Example: `nginx_server` instead of install_and_configure_nginx_on_server
- Use a consistent naming scheme
  - Improves collaboration and maintainability
  - Example: `myorg_nginx_server`, `myorg_mysql_server`, `myorg_php_fpm`
- Namespace your roles
  - Avoid naming conflicts and identify the author
  - Example: `yourusername.nginx_server` or `yourorganization.nginx_server`

Role versioning

- Use Semantic Versioning (SemVer)
  - Format: `MAJOR.MINOR.PATCH`
  - Helps users understand changes between versions
- Update `meta/main.yml`
  - Specify role version in version field
- Use version control (e.g., Git)
  - Track changes, collaborate, and maintain history
- Tag releases in version control
  - Corresponds to role's version number
  - Easy access and reference to specific versions
- Specify role versions in playbooks
  - Ensure compatibility and stability
  - Use `src`, `name`, and `version` fields in `requirements.yml` or with `ansible-galaxy` command

Ansible Collections

- Ansible Collections are a way to package and distribute roles, modules, plugins, and other Ansible content as self-contained units.
- Collections can be published to the Ansible Galaxy platform or used locally within an organization.
- Combining roles with collections allows for easier sharing and distribution of roles and promotes reusability across different projects.
- Roles can be included in collections as part of a larger package of Ansible content.
- Collections can include multiple roles and can be versioned and managed independently from each other.
- Collections can be installed and managed using the **`ansible-galaxy`** command, similar to how roles are managed.
- e.g. https://galaxy.ansible.com/community/docker, https://galaxy.ansible.com/l3d/git, …

Workshop Exercise - Roles: Making your playbooks reusable

https://aap2.demoredhat.com/exercises/ansible_rhel/1.7-role/

end