Ansible Variables: Notes

# Ansible Variables: An Overview

Ansible variables allow you to define dynamic values in your playbooks. They help customize the playbooks for different environments, hosts, or conditions.

## Types of Variables

1. Simple Variables:

* - Basic key-value pairs.

my\_variable: "Hello, World!"

2. Complex Variables:

* - Variables containing lists, dictionaries, or other data structures.

my\_list:  
 - item1  
 - item2  
my\_dict:  
 key1: value1  
 key2: value2

3. Facts:

* - Automatically gathered information about the remote system.
* - Use `ansible\_facts` or `ansible\_hostname`, `ansible\_os\_family`, etc.
* - Can be accessed as `{{ ansible\_facts['os\_family'] }}` or `{{ ansible\_hostname }}`.

4. Magic Variables:

* - Special variables Ansible provides for context.
* - Examples: `hostvars`, `groups`, `inventory\_hostname`, `playbook\_dir`, etc.

# Variable Precedence Order

Ansible variables have a specific precedence order, which determines which variable is used when there are conflicts. The following list shows the order from lowest to highest precedence:

1. Command-line options (`-e`): Variables passed directly on the command line.

ansible-playbook playbook.yml -e "my\_variable=value"

2. Role defaults: Defined in the `defaults/main.yml` file of a role.

3. Inventory variables: Defined in inventory files or directories (e.g., `inventory/hosts`). Host-specific variables. Group-specific variables.

4. Playbook group\_vars: Defined in the `group\_vars/` directory within the playbook structure. Apply to all hosts in the group.

5. Playbook host\_vars: Defined in the `host\_vars/` directory within the playbook structure. Apply to a specific host.

6. Playbook vars\_prompt: Variables prompted for at runtime using `vars\_prompt`.

7. Playbook vars\_files: Variables loaded from external files via `vars\_files` in playbooks.

8. Playbook vars: Variables defined directly within a playbook using the `vars` keyword.

9. Task vars (only for the specific task): Variables defined for a specific task using the `vars` keyword.

10. Block vars: Variables defined within a block of tasks.

11. Role vars: Defined in the `vars/main.yml` file of a role.

12. Include vars: Variables loaded from a file included in the playbook or task using `include\_vars`.

13. Set\_facts / registered variables: Variables set using `set\_fact` or registered from task outputs.

14. Extra vars (highest precedence): Variables passed on the command line using the `--extra-vars` or `-e` option.

# Default Variables

1. Role Defaults:

* - Variables defined in `defaults/main.yml` within a role.
* - These have the lowest precedence and are often used to provide fallback values.

2. Fallback Values:

* - You can set a default value for a variable using the Jinja2 `default` filter.

my\_variable: "{{ my\_variable | default('Default Value') }}"

# Best Practices with Ansible Variables

1. Use Descriptive Variable Names: Make variable names descriptive and unique to avoid conflicts.

2. Organize Variables: Use `group\_vars` and `host\_vars` directories to organize variables according to groups and hosts.

3. Leverage Role Defaults: Use role defaults to provide baseline values that can be overridden if necessary.

4. Avoid Hardcoding Values: Where possible, avoid hardcoding values; instead, use variables to make playbooks flexible and reusable.

5. Use `vars\_files`: Separate your variable definitions into `vars\_files` for better organization and readability.