**Ansible Variables, Dictionaries, and Facts - Beginner Study Material**

**What are Ansible Variables?**

Variables in Ansible are used to store values that can change dynamically during execution. These values can include strings, numbers, lists, and dictionaries. Variables help make playbooks more flexible and reusable.

**Rules for Ansible Variables:**

* Variables can contain letters, numbers, or underscores (\_).
* Variables must start with a letter.
* Variables are case-sensitive.
* Variables can be defined in multiple ways:
  + Inside playbooks.
  + In separate variable files.
  + In inventory files.
  + As command-line arguments.

**Example of Variables in Ansible:**

---

vars:

username: "JohnDoe"

age: 30

country: "USA"

**What is an Ansible Dictionary?**

A dictionary in Ansible is a collection of key-value pairs. It allows multiple related variables to be stored together.

**Example of a Dictionary in Ansible:**

---

vars:

user:

name: "Alice"

age: 25

city: "New York"

**Accessing Dictionary Values:**

To access dictionary values in a playbook:

- name: Display user details

debug:

msg: "User's name is {{ user.name }} and age is {{ user.age }}"

**What are Ansible Facts?**

Ansible facts are system information that Ansible gathers automatically when executing a playbook. Facts provide details about the target host, such as:

* Operating System
* IP Address
* Memory and Disk Usage
* Hostname

Facts are useful when you need to perform tasks conditionally based on the system's state.

**Example: Using Ansible Facts**

- name: Display system information

debug:

msg: "The system has {{ ansible\_processor\_vcpus }} CPU cores and {{ ansible\_memory\_mb.real.total }} MB RAM."

**Using Ansible Facts in Conditions**

Facts can be used in conditional statements to execute tasks based on system properties.

**Example: Running a Task on Specific OS**

- name: Install Apache on Ubuntu

apt:

name: apache2

state: present

when: ansible\_os\_family == "Debian"

This ensures that Apache is installed only on Debian-based systems (like Ubuntu).

**Summary Table**

| **Feature** | **Description** |
| --- | --- |
| **Variables** | Store values dynamically. |
| **Dictionaries** | Store multiple related values as key-value pairs. |
| **Facts** | System information automatically gathered by Ansible. |

This study material provides a beginner-friendly understanding of Ansible variables, dictionaries, and facts, with examples to demonstrate their usage. Let me know if you need any modifications or additional explanations!

**# Ansible Variables - A Beginner's Guide**

**What are Ansible Variables?**

Ansible variables allow you to store and reuse values across your playbooks. They help make configurations flexible and manageable. Variables can store information like usernames, passwords, environment names, and file paths.

**Rules for Defining Variables:**

* Must start with a letter.
* Can contain letters, numbers, and underscores (\_).
* Case-sensitive.
* Should be defined in a structured manner.

**Defining Variables in Ansible**

**1. Defining Variables in a Playbook**

Variables can be defined inside a playbook under the vars section.

- hosts: localhost

vars:

environment\_name: "development"

tasks:

- name: Display the environment variable

debug:

msg: "The environment is {{ environment\_name }}"

**2. Defining Variables in a Separate File**

Variables can be stored in a separate YAML file and included in the playbook.

# vars.yml

environment\_name: "production"

Then, include it in your playbook:

- hosts: localhost

vars\_files:

- vars.yml

tasks:

- name: Display the environment variable

debug:

msg: "The environment is {{ environment\_name }}"

**3. Defining Variables in the Inventory File**

You can define variables in the inventory file under host\_vars or group\_vars.

[webservers]

server1 ansible\_host=192.168.1.10 environment\_name=staging

**Using Variables in Tasks**

You can reference variables using double curly braces ({{ variable\_name }}).

- hosts: localhost

vars:

environment\_name: "testing"

tasks:

- name: Create a file using a variable

file:

path: "{{ environment\_name }}\_config.txt"

state: touch

**Using Variables in Templates**

Ansible allows using variables in Jinja2 templates. Below is an example template file (config.j2):

Environment: {{ environment\_name }}

Application Name: {{ app\_name }}

You can deploy it using a task:

- hosts: localhost

vars:

environment\_name: "production"

app\_name: "MyApp"

tasks:

- name: Deploy configuration file

template:

src: config.j2

dest: /etc/myapp/config.txt

**Loops and Conditional Statements with Variables**

**Looping Through a List**

- hosts: localhost

vars:

users:

- alice

- bob

- charlie

tasks:

- name: Create user directories

file:

path: "/home/{{ item }}"

state: directory

loop: "{{ users }}"

**Using Variables in Conditions**

- hosts: localhost

vars:

os\_type: "Ubuntu"

tasks:

- name: Install software only on Ubuntu

apt:

name: nginx

state: present

when: os\_type == "Ubuntu"

**Why Use Variables?**

| **Feature** | **Benefit** |
| --- | --- |
| Reusability | Avoids repetition of values in playbooks |
| Maintainability | Centralized management of values |
| Flexibility | Easily switch configurations for different environments |
| Scalability | Helps manage a large number of hosts efficiently |

**Conclusion**

Using variables in Ansible makes automation more efficient and manageable. By defining variables in playbooks, inventory files, or separate files, you can create flexible and reusable automation scripts.

**Next Steps:**

* Try creating a playbook that uses variables.
* Explore group\_vars and host\_vars for managing variables per host or group.
* Use ansible-playbook command with -e flag to pass variables dynamically.

**Ansible Facts and Variables: A Beginner's Guide**

**Introduction**

Ansible is an automation tool that helps manage servers, applications, and configurations. One of its key features is the ability to use **variables** and **facts** to make automation dynamic and efficient.

This guide will explain:

* **Ansible Variables**
* **Ansible Facts**
* **How to use them in Playbooks**
* **Examples and Commands**

**1. Ansible Variables**

**What are Variables?**

Variables in Ansible are placeholders used to store data. This helps avoid hardcoding values and allows dynamic control over configurations.

**Rules for Ansible Variables:**

* Can contain **letters, numbers, and underscores (\_)**.
* Must start with a **letter**.
* Cannot have spaces.

**Defining Variables**

Variables can be defined in multiple ways:

1. **Inside Playbooks**
2. **Inside Inventory Files**
3. **Using Command-line Arguments**
4. **Inside Separate Variable Files**

**Example: Defining Variables in a Playbook**

- hosts: webservers

vars:

env\_name: "development"

tasks:

- name: Display Environment Name

debug:

msg: "The environment is {{ env\_name }}"

**Using Variables in Tasks**

You can use variables inside tasks by enclosing them in {{ }}.

- name: Create a file using a variable

file:

path: "/tmp/{{ env\_name }}\_config.txt"

state: touch

**2. Ansible Facts**

**What are Ansible Facts?**

Ansible Facts are pieces of information automatically gathered about target machines (e.g., OS type, IP address, hostname, etc.).

These facts help in making playbooks dynamic and adaptable.

**How to Gather Facts?**

* Facts are collected automatically when running a playbook.
* You can use the setup module to see all available facts.

**Example: Gathering System Facts**

ansible all -m setup

This command will display a lot of system-related information such as IP addresses, OS details, CPU info, etc.

**Using Specific Facts**

You can filter specific facts using the filter argument.

ansible all -m setup -a 'filter=ansible\_distribution'

This will return the OS name (e.g., Ubuntu or CentOS).

**Example: Using Facts in Playbooks**

- hosts: all

tasks:

- name: Display OS Type

debug:

msg: "This server is running {{ ansible\_distribution }}"

**Disabling Facts Gathering**

If you don’t want Ansible to collect facts automatically, set gather\_facts: no.

- hosts: all

gather\_facts: no

tasks:

- name: Print Message

debug:

msg: "Facts gathering is disabled!"

**3. Comparison Table: Variables vs. Facts**

| **Feature** | **Ansible Variables** | **Ansible Facts** |
| --- | --- | --- |
| **Definition** | User-defined values | System-generated values |
| **Stored In** | Playbooks, inventory, external files | Gathered automatically by Ansible |
| **Usage** | Custom configurations | System details like OS, IP, etc. |
| **Defined By** | User | Ansible’s setup module |
| **Example** | env\_name: "production" | ansible\_distribution: "Ubuntu" |

**4. Using Variables and Facts Together**

**Example Playbook Combining Variables and Facts**

- hosts: all

vars:

config\_dir: "/etc/configs"

tasks:

- name: Create config file using facts and variables

file:

path: "{{ config\_dir }}/{{ ansible\_distribution }}\_config.txt"

state: touch

In this example:

* config\_dir is a user-defined **variable**.
* ansible\_distribution is an **Ansible fact**.
* The result is a file named /etc/configs/Ubuntu\_config.txt (or another OS name) created dynamically.

**Conclusion**

Ansible Variables and Facts make playbooks powerful and flexible:

* **Variables** allow you to customize playbooks.
* **Facts** provide real-time system information.
* They can be used together for efficient automation.

By mastering variables and facts, you can build highly adaptable and scalable Ansible playbooks. 🚀