📗

Ansible Playbook Fundamentals

Author: Ramu Chelloju

🔗 https://www.linkedin.com/in/ramuchelloju/

This document covers the foundational concepts of Ansible playbooks, focusing on syntax, execution flow, and best practices for configuration management.

1️⃣ YAML & Playbook Structure

Ansible playbooks are written in **YAML**.

Strict indentation (usually 2 spaces) is required. Tabs are not allowed.

- `---`: Indicates the start of a YAML document.
- `name:`: A human-readable label for a Play or Task. Helps track execution in the console output.
- `hosts:`: Defines the target servers for the play (e.g., `all`, `webservers`, or `localhost` for local execution.
- `connection:`: Defines how Ansible connects to the host.
  - `ssh` (default for remote systems)
  - `local` (used when targeting `localhost`)
- `become: yes`: Escalates privileges (equivalent to `sudo`). Required for installing packages, managing services, or modifying system configurations.
- `gather_facts: yes/no`: Controls whether Ansible collects system information before running tasks. Set to `no` to speed up execution if facts are not required.

2️⃣ Variables (`vars`)

Variables allow you to store data and reuse it throughout your playbook.

Definition

Declared under the `vars:` block at the Play level.

Usage

Accessed using Jinja2 templating syntax:

```
{{ variable_name }}
```

Data Types

- **String**

  ```
  course: "DevOps"
  ```

- **Number**

  ```
  duration: 140
  ```

- **Boolean**

  ```
  is_live: true
  ```

- **List**

  ```
  topics:
    - Linux
    - Ansible
  ```

- **Dictionary (Map)**

  ```
  details:
    trainer: "Ramu"
  ```

```
  months: 4
```

3️⃣ Conditionals (`when`)

Conditionals control task execution based on logical expressions.

If the condition evaluates to `false` The task is marked as `skipped`.

Boolean Checks

```
when: IS_LIVE
when: not IS_LIVE
```

Comparisons

Supports standard operators: `==`, `!=`, `>`, `<`, `>=`, `<=`

Example:

```
when: NUMBER_OF_SESSIONS > 5
```

String Matching

```
when: COURSE == "DevOps"
```

4️⃣ Loops (`loop`)

Loops allow a single task to run multiple times with different data.

Standard Loop

Ansible automatically assigns the current value to the reserved variable `item`.

```
loop:
  - git
  - curl
  - vim
```

Dictionary Loop

Used when each item has multiple attributes.

```
loop:
  - { name: "git", state: "present" }
  - { name: "httpd", state: "absent" }
```

Access properties using:

```
{{ item.name }}
{{ item.state }}
```

Loop Control

Use `loop_control` and `loop_var` to rename `item`, especially in nested loops.

5️⃣ Command vs Shell Module

Understanding the difference is critical.

ansible.builtin.command (Preferred)

- Executes commands directly.
- Does NOT support shell features like:
  - Pipes (`|`)
  - Redirects (`>`)
  - Chaining (`&&`)

- Preferred in most cases for safety.

ansible.builtin.shell

- Executes commands through `/bin/sh`.
- Supports:
  - Pipes
  - Redirects
  - Wildcards
  - Environment variables
- Use only when shell features are strictly required.

6️⃣ Registering Output (`register`)

The `register` keyword captures task output and stores it in a variable.

Example

```
- name: Check system uptime
  ansible.builtin.command: uptime
  register: uptime_result
```

Accessing Output

The registered variable is a dictionary.

Common keys:

- `uptime_result.stdout`
- `uptime_result.stderr`
- `uptime_result.rc` (return code)

Example:

```
{{ uptime_result.stdout }}
```

7️⃣ Error Handling (`ignore_errors`)

By default, Ansible stops execution if a task fails (non-zero return code).

Continue Even If Task Fails

```
ignore_errors: true
```

Common Use Case

```
- name: Check if user exists
  ansible.builtin.command: id devops
  register: user_check
  ignore_errors: true

- name: Create user if not exists
  ansible.builtin.user:
    name: devops
    state: present
  when: user_check.rc != 0
```

8️⃣ Filters (Data Transformation)

Filters transform data using the pipe (`|`) operator inside Jinja2 expressions.

Common Filters

- `default('value')` → Fallback if variable is undefined
- `split(',')` → Convert string to list

- `dict2items` / `items2dict` → Convert between dictionary and list formats
- `to_nice_json` → Format output as readable JSON
- `upper` / `lower` → Change string casing

Example:

```
{{ username | upper }}
```

9️⃣ Idempotency & State (`state`)

Idempotency is the core philosophy of Ansible.

Running the same playbook multiple times should produce the same final system state without unnecessary changes or failures.

Ansible modules internally check the current system state before making changes.

Declarative vs Imperative

- Imperative → Execute commands step by step.
- Declarative → Define the desired end state.

Ansible follows a declarative approach.

The `state` Parameter

- `state: present`

  Ensures the resource exists.

  If missing → create/install.

  If already present → no change (`ok`).

- `state: absent`

  Ensures the resource does not exist.

  If present → remove.

  If already absent → no change.

- `state: latest`

  Ensures the resource is installed and upgraded to the latest available version.

🎤 Interview Gold: Essential Q&A

Q: What is the difference between the `command` and `shell` modules?

A: The `command` module executes binaries directly and does not interpret shell operators like pipes (`|`) or redirects (`>`). The `shell` module runs commands through `/bin/sh`, allowing pipes and redirects but introducing potential shell injection risks.

Q: How do you handle a task that might fail, but you want the playbook to continue?

A: I use `ignore_errors: true` on that task. Often, I combine it with `register` to capture the return code and apply conditional logic in subsequent tasks.

Q: Explain Idempotency in Ansible.

A: Idempotency means that executing a playbook multiple times results in the same system state without redundant changes. Ansible modules compare the current state with the desired state and apply changes only if necessary.

Q: How do you run tasks only on RedHat systems?

A: I enable `gather_facts: yes` and use:

```
when: ansible_facts['os_family'] == 'RedHat'
```

Q: How do you efficiently install multiple packages?

A: I use the `ansible.builtin.package` module with `loop`:

```
- name: Install packages
  ansible.builtin.package:
    name: "{{ item }}"
    state: present
  loop:
    - git
    - curl
    - vim
```

Q: Difference between `state: present` and `state: latest`?

A:

- `present` ensures the package is installed but does not force upgrades.
- `latest` ensures installation and upgrades to the newest available version, which may introduce changes or compatibility risks.

Prepared by:

Ramu Chelloju

🔗 https://www.linkedin.com/in/ramuchelloju/
