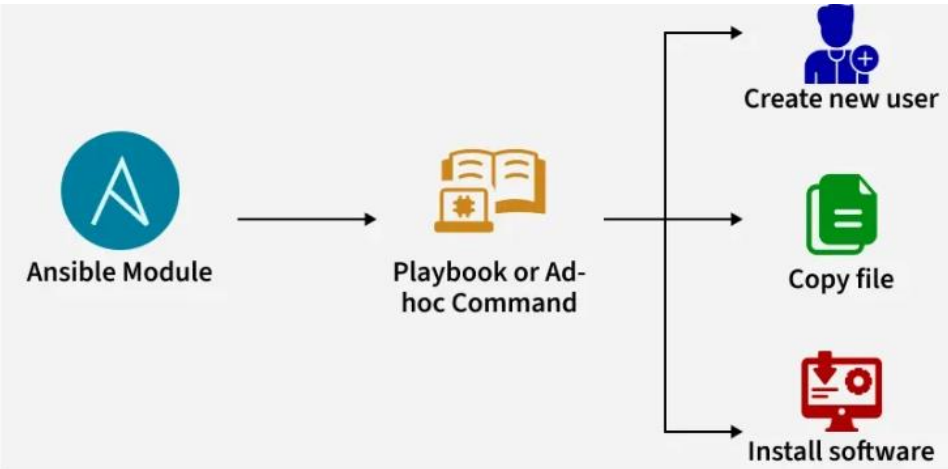
4.5 ANSIBLE MODULES

Ansible Modules are used to perform various tasks on remote hosts. These are basically small, standalone scripts that ansible uses to communicate with the remote hosts in order to execute specific actions.

An Ansible Module is a small program or script that comes pre-packaged with Ansible and is designed to perform a specific task in automation. These modules act as the building blocks of Ansible automation, allowing users to manage configurations, deploy applications, handle services, or manage cloud resources without writing long scripts.



For example:

- If you want to create a new user: Ansible uses the user module.
- If you want to copy a file: Ansible uses the copy module.
- If you want to install software: Ansible uses the apt or yum module.

Whenever you run a playbook or an ad-hoc command, Ansible automatically calls the necessary module to carry out the requested operation. This makes automation simple, reliable, and less error-prone.

Modules as the Core of Automation

Modules are the core units of work in Ansible. Each module is specialized: some handle system-level operations (like creating users or installing packages), while others handle application-level tasks (like managing databases or web servers).

- They eliminate the need for manual scripting.
- Provide reusable, standardized methods for system automation.
- Ensure cross-platform compatibility, meaning the same module can work on Linux, Windows, or cloud environments.

Idempotency of Ansible Modules

Idempotency means that if you run the same module multiple times, it will only make changes when needed.

For example:

- If a user already exists, the user module won't create it again.
- If a service is already running, the service module won't restart it unnecessarily.
- If a file already has the correct content, the copy or template module won't overwrite it.

This prevents duplication, ensures system stability, and gives you predictable results every time you run Ansible.

Types of Ansible Modules

Ansible has hundreds of modules built-in, and they are grouped based on what they do

System Modules in Ansible

System modules are the main part of Ansible automation. They help you manage the core operating system tasks like users, groups, services, and software packages. It allowing you to manage users, services, and software packages across multiple systems with ease.

- **user module**: Create, delete, or modify users on remote servers.
- **service module**: Start, stop, enable, or restart services like Apache, Nginx, or MySQL.
- **yum / apt modules**: Install or remove software packages in Red Hat (yum) or Debian/Ubuntu (apt) based systems.

File & Storage Modules in Ansible

File and storage modules allow you to manipulate files, directories, and permissions across multiple servers at once.

- **copy module**: Push configuration files (like nginx.conf, application.yml) from your local machine to multiple remote servers in one go.
- **file module**: Manage file and directory permissions, ownership, or even create symbolic links.
- **fetch module**: Pull files from remote servers to your control node.

Cloud Modules in Ansible

Cloud modules are designed for **managing cloud infrastructure** as part of Infrastructure as Code (IaC).

- **ec2 module (AWS)**: Create, start, stop, or terminate EC2 instances. Ideal for scaling applications dynamically in the cloud.
- **gcp_compute_instance module (Google Cloud)**: Manage virtual machine instances, storage, and networking in GCP.

Database Modules in Ansible

Database modules simplify **database administration tasks** by automating user management, schema creation, and backup operations.

- **mysql_user module**: Create, update, or delete MySQL users with specific privileges. Commonly used in setting up **secure web applications**.
- **postgresql_db module**: Create or drop PostgreSQL databases and manage extensions. Helpful for **DevOps CI/CD pipelines** where test databases need to be provisioned automatically.

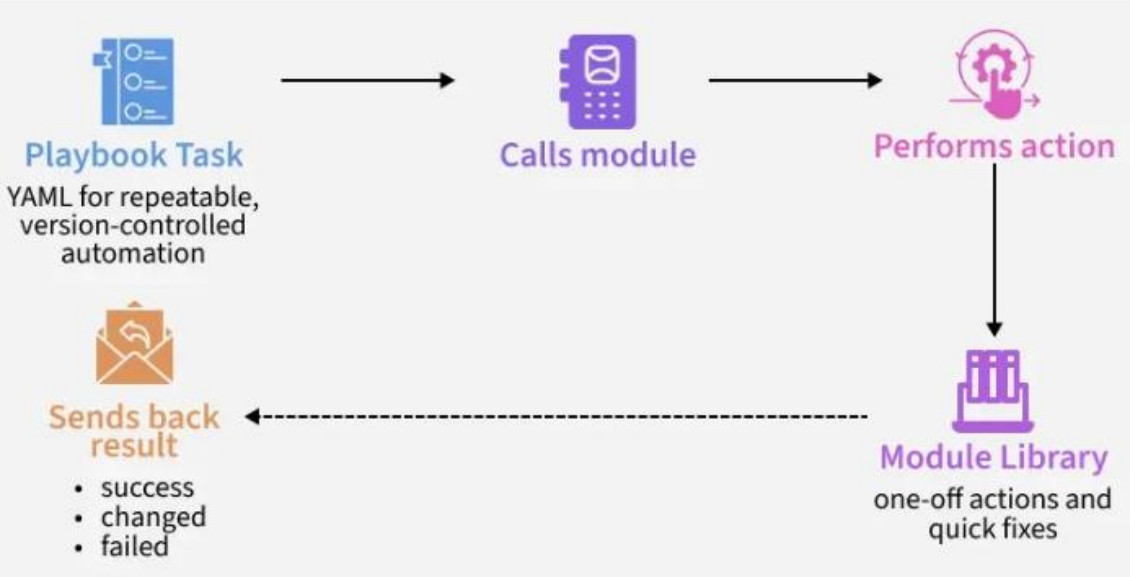
Networking Modules in Ansible

Networking modules automate tasks on **routers, switches, and firewalls**, making them crucial for **network engineers and security teams**.

- **ios_config module**: Manage configurations on Cisco IOS devices, such as VLANs, routing protocols, and ACLs.
- **firewalld module**: Configure firewall rules on Linux servers, allowing or blocking specific ports/services. This improves **infrastructure security and compliance**.

How Ansible Modules Work

Ansible run is a conversation between you, the task, and the target system you declare what you want, Ansible runs the right module, applies only the needed changes, and then reports back with a clear result. Here are the steps how it works



There are two entry points playbook task or use an ad-hoc command. Playbook task use for YAML for repeatable, version-controlled automation and Ad-hoc command (ansible all -m package -a "name=git state=present") for one-off actions and quick fixes.

2. After that Ansible calls the respective module in the background. Ansible resolves the module name (e.g., package, copy, service, user, apt, yum) from its module library, including any Collections you've installed.

3. Then the module performs the action (install package, copy file, etc.).

Modules are designed to be idempotent. They first probe current state (e.g., "Is git installed?", "Does /etc/app.conf match the desired content?").

- If the target state already matches, they do nothing: changed: false.
- If not, they make the minimal change needed: changed: true.

4. The module sends back the result to Ansible (success, changed, or failed). Each module returns a JSON payload that includes:

- changed (boolean): whether something actually changed.
- failed (boolean + message): whether the task failed and why.

4.6 ANSIBLE INVENTORY FILES

Ansible inventory files serve as a central registry for the hosts and groups of hosts that Ansible manages. These files define the targets for Ansible operations, enabling automation tasks to be executed on specific servers or collections of servers.

Key characteristics of Ansible inventory files:

Purpose: To list and organize managed nodes (servers, network devices, etc.) that Ansible will interact with.

Formats: Commonly written in INI or YAML format. Ansible includes built-in support for these, and other formats can be used via inventory plugins.

Structure:

Hosts: Individual managed nodes are listed by their hostname or IP address.

Groups: Hosts can be organized into logical groups based on shared characteristics (e.g., webservers, database_servers, production, development). Groups are defined using square brackets in INI format or as top-level keys in YAML.

Variables: Host-specific or group-specific variables can be defined within the inventory file to customize behavior for different targets.

Location: By default, Ansible looks for an inventory file at /etc/ansible/hosts. However, a custom inventory file or directory can be specified using the -i flag on the command line or by configuring the ansible.cfg file.

Dynamic Inventory: Ansible supports dynamic inventory, where host information is retrieved from external sources like cloud providers or other systems using inventory plugins. This allows for flexible and up-to-date inventories in dynamic environments.

Usage: When running Ansible playbooks or ad-hoc commands, the inventory file is used to determine which hosts or groups of hosts the operations will be performed on.

Example of a YAML-formatted inventory file:

```
all:
  hosts:
    web1.example.com:
    web2.example.com:
  children:
    webservers:
      hosts:
        web1.example.com:
        web2.example.com:
    database_servers:
      hosts:
        db1.example.com:
        db2.example.com:
  vars:
    ansible_user: ubuntu
    ansible_ssh_private_key_file: /path/to/your/ssh/key.pem
```