2025

ANSIBLE

FROM SCRATCH TO ADVANCE

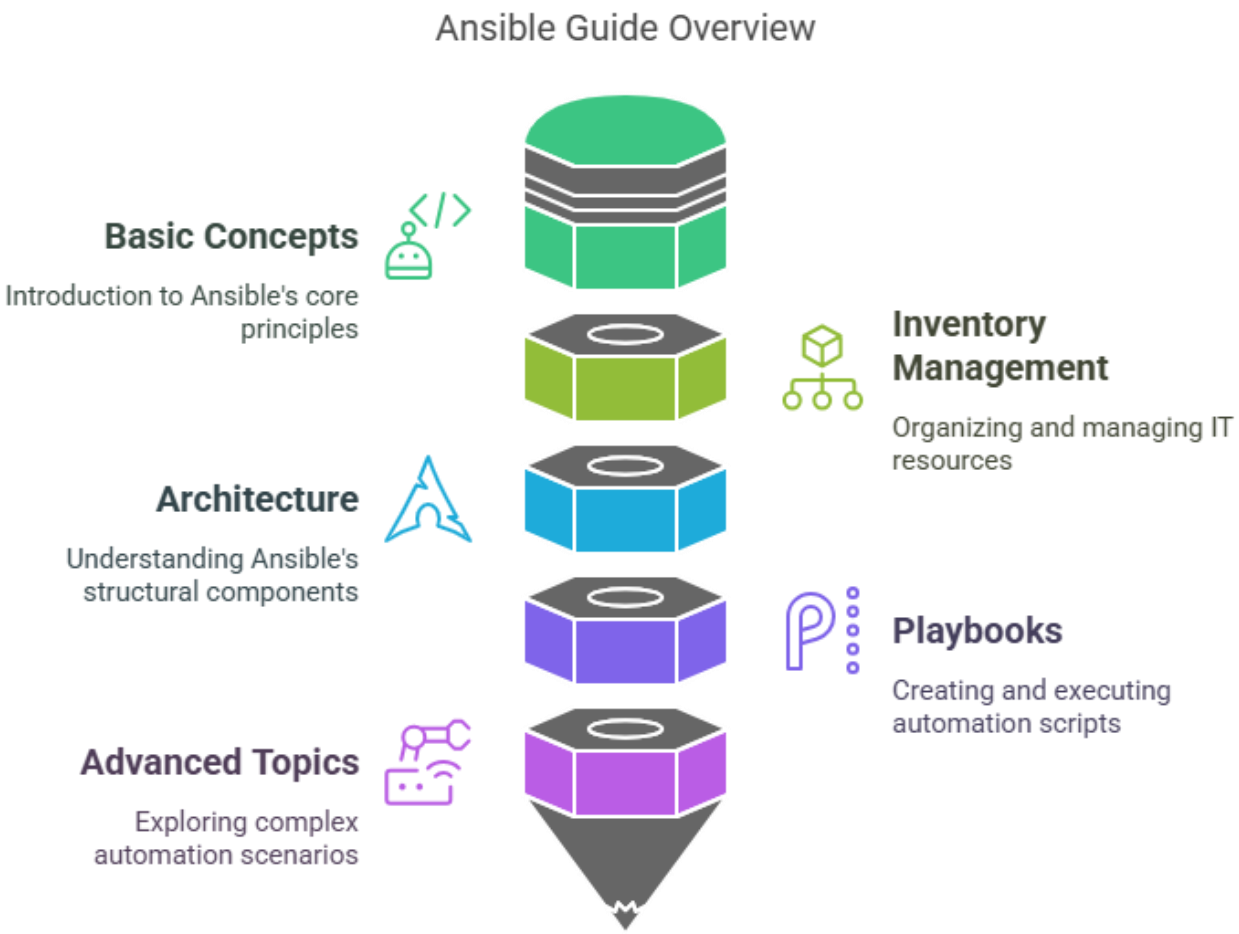
VISHAL MACHAN MAIL ID : vishalmachan15@gmail.com LINKEDIN: www.linkedin.com/in/machan-vishal

📚 Table of Contents: Ansible Deep Dive Guide

🌐 5. Network Automation: Advanced Topics

- Network Resource Modules
  - Module States
  - Using Network Modules
  - Example: Config Verification
  - Example: VLAN Management
- Ansible Network Examples
  - Prerequisites
  - Inventory Groups and Variables
  - Example 1: Collection Facts & Backups
  - Example 2: Cross-platform Modules
  - Implementation Notes
  - Troubleshooting
- Parsing Semi-Structured Text
  - CLI Parser Basics
  - Parsing CLI Output
  - Advanced Parsing Use Cases
- Data Validation with Ansible
  - Using the `validate` Plugin
  - Structuring & Validating Data
- Network Debug and Troubleshooting Guide
  - General Troubleshooting
  - Shell Issues
  - Socket/Path Errors
  - Timeout Errors
  - Proxy & Miscellaneous Issues

🔧 Ansible: From Scratch to Deep Dive



---

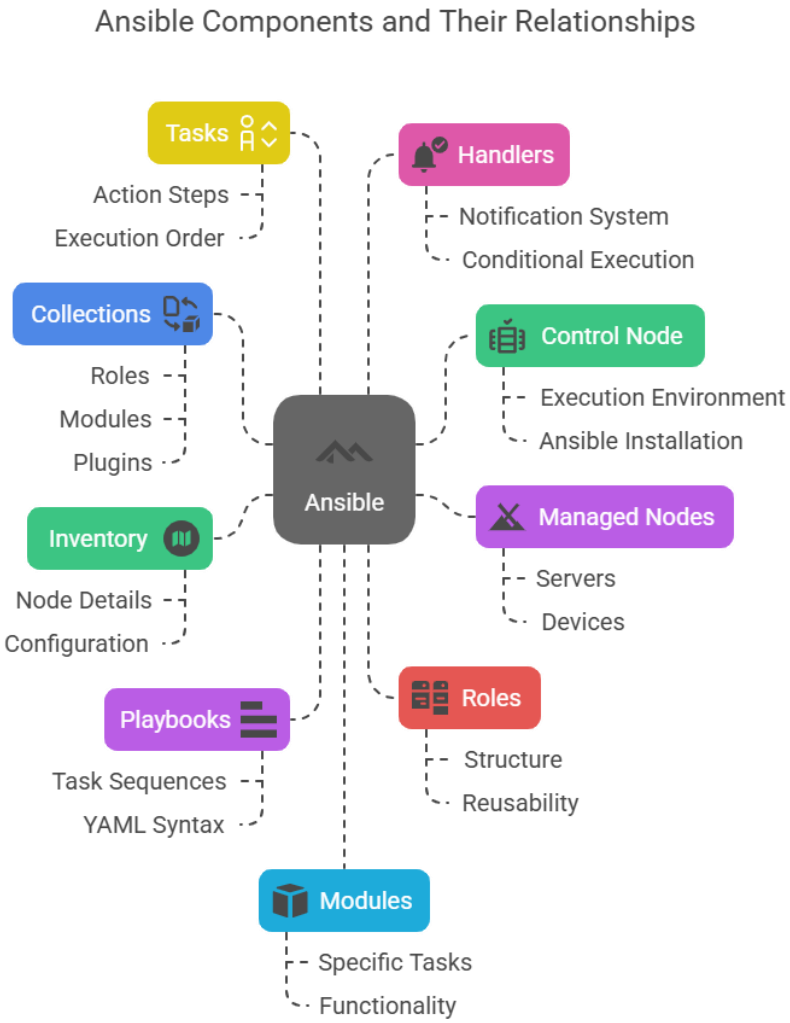
1. 🟢 Introduction to Ansible

Ansible is an open-source IT automation tool developed by Red Hat that enables:

- Configuration Management
- Application Deployment
- Orchestration
- Provisioning
- Continuous Delivery

It uses simple YAML files (called playbooks) and does not require an agent to be installed on target machines, making it lightweight and easy to manage.

Basic Concepts



Control Node

- The machine where Ansible is installed and from where commands are executed.
- This is the *central orchestrator* that pushes instructions to other systems (managed nodes) via SSH.

Managed Nodes

- The remote machines (servers, VMs, etc.) that Ansible manages.
- No need to install an agent; communication happens over SSH or Win-RM (for Windows).

Inventory

- A file (usually hosts) that defines which machines are managed.
- Can be static (INI/YAML) or dynamic (using scripts or plugins).

- Example:
- [webservers]
- web1.example.com
- web2.example.com

Playbooks

- YAML files that define *what* needs to be done.
- Each consists of one or more *plays*, each targeting a group of hosts with defined tasks.
- Example:
- - hosts: webservers
- tasks:
- - name: Install nginx
- apt:
- name: nginx
- state: present

Plays

- A play maps a group of hosts to tasks.
- A play consists of:
    - Host selection
    - Variable declarations
    - Task execution
- Purpose: Describe the policy for a specific set of hosts.

Roles

- A structured way to organize playbooks and reuse code.
- Components:
    - tasks/
    - handlers/
    - defaults/
    - vars/
    - files/

- templates/
- meta/

Example usage:

```
- hosts: webservers
  roles:
    - nginx
```

Tasks

- A single action to be executed on managed nodes.
- Executes modules with specified arguments.
- Tasks are the smallest unit in playbooks.

Example:

```
- name: Install nginx
  apt:
    name: nginx
    state: present
```

Handlers

- Tasks that are triggered only when notified.
- Used for idempotent operations like restarting services after configuration changes.

Example:

```
- name: Restart nginx
  service:
    name: nginx
    state: restarted
  listen: "restart nginx"
```

Modules

- Reusable standalone scripts used by Ansible to perform tasks.
- Examples:
  - apt, yum, copy, file, service, command, shell

- You can write custom modules in any language.

📌 **Plugins**

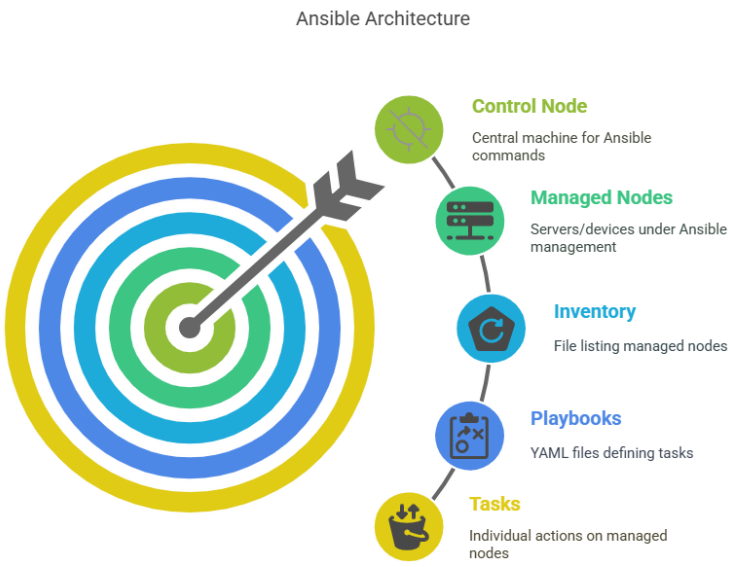
- Extend Ansible's core functionality.
- Types:
  - Callback plugins (e.g., output formatting)
  - Connection plugins (e.g., SSH, Docker)
  - Lookup plugins
  - Filter plugins

📌 **Collections**

- Packaging format introduced in Ansible 2.9+.
- Collections bundle roles, playbooks, modules, and plugins.
- Distributed via Ansible Galaxy.

Example:

```
- name: Use community.mysql
  hosts: db
  collections:
    - community.mysql
```



🔴 **Deep Dive**

🔍 **Execution Model**

1. Inventory is loaded
2. Variables are parsed and merged
3. Plays are selected based on hosts
4. Tasks are executed in order using modules
5. Handlers are notified and executed
6. Results are reported

🔒 **Ansible Vault**

- Encrypt sensitive data like passwords and keys.
- Usage:

- ansible-vault encrypt secrets.yml

⚙️ Dynamic Inventory

- Used in cloud environments (AWS, Azure, GCP) where hosts change frequently.
- Ansible provides inventory scripts and plugins for dynamic environments.

🔁 Loops, Conditionals, and Filters

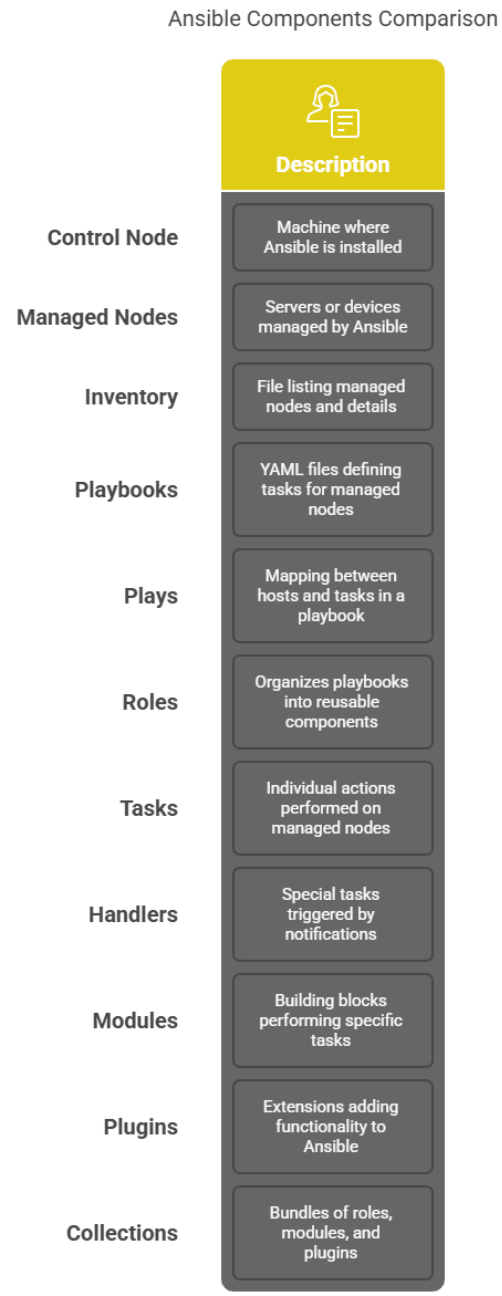
- Loop:
- - name: Install packages
- apt:
- name: "{{ item }}"
- state: present
- loop:
- - nginx
- - git
- Conditionals:
- - name: Only for Debian
- apt:
- name: nginx
- state: present
- when: ansible_os_family == "Debian"
- Filters:
- {{ myvar | lower }}

🧩 Templates (Jinja2)

Use Jinja2 templating for dynamic config files.

- - name: Template nginx config
- template:
- src: nginx.conf.j2
- dest: /etc/nginx/nginx.conf

🔄 Idempotency



- Ansible ensures that running a playbook multiple times doesn't change the system if it's already in the desired state.

📈 Ansible Tower / AWX

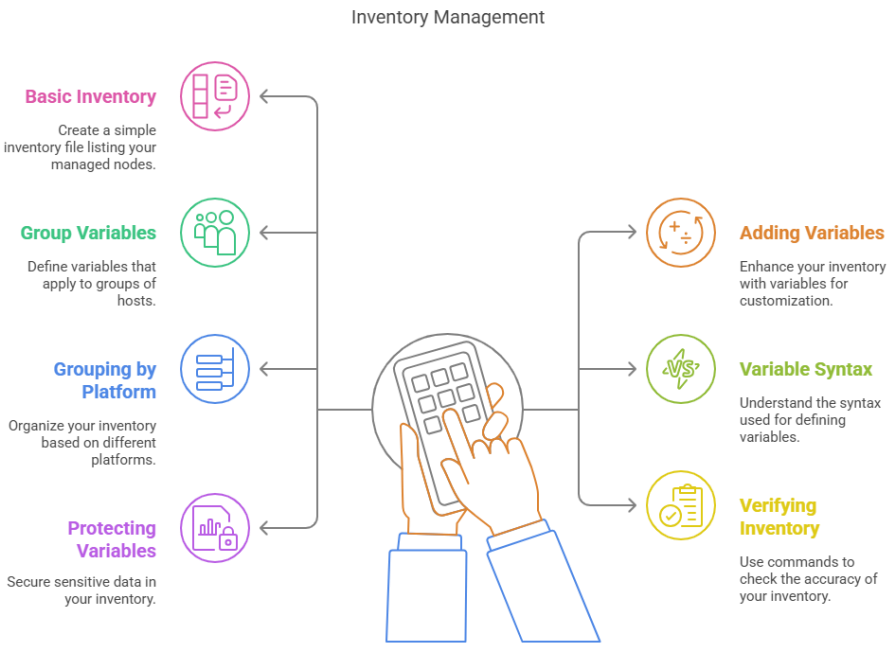
- A web-based UI and REST API for managing Ansible automation.
- Adds RBAC, inventory sync, logging, job scheduling, and workflows.

✅ Best Practices

- Use roles for code reuse and structure.
- Store secrets with Ansible Vault.
- Keep playbooks idempotent.
- Use dynamic inventory in the cloud.
- Use tags for selective task execution.
- Document playbooks and use meaningful task names.

---

2. 🗂️ Deep Dive: Building Your Ansible Inventory



📌 1. What is an Inventory in Ansible?

An **inventory** in Ansible is a file or script that defines **managed nodes (hosts)** and how to connect to them. It is the foundation for targeting machines with your playbooks.

Ansible inventories can be:

- **Static**: Simple INI or YAML files
- **Dynamic**: Scripts or plugins that pull real-time host data (e.g., from AWS, Azure)

🟢 2. Basic Static Inventory (INI Format)

A simple example of an inventory in INI format:

```
[webservers]
web1.example.com
web2.example.com

[dbservers]
db1.example.com ansible_user=admin ansible_ssh_pass=secret
```

- web1.example.com and web2.example.com belong to group webservers
- ansible_user and ansible_ssh_pass are **host-level variables**

Run a playbook using this inventory:

```
ansible-playbook -i inventory.ini playbook.yml
```

✍️ 3. Add Variables to the Inventory

You can define **host-specific variables** or **group-wide variables**.

Example: Host Variables

```
[routers]
router1 ansible_host=192.168.1.1 ansible_user=admin ansible_password=admin123
```

Example: Group Variables

```
[routers:vars]
ansible_network_os=ios
ansible_connection=network_cli
```

📦 4. Group Variables within Directory Inventory

Directory-style inventories support **host_vars** and **group_vars** folders.

Structure Example:

```
inventory/
├── hosts
├── group_vars/
│   └── webservers.yml
├── host_vars/
│   └── web1.example.com.yml
```

group_vars/webservers.yml

```
ansible_user: ubuntu
ansible_python_interpreter: /usr/bin/python3
```

host_vars/web1.example.com.yml

```
nginx_port: 8080
```

📜 5. Variable Syntax and Precedence

Ansible merges variables from multiple sources. The **precedence order** (lowest to highest) includes:

1. Role defaults
2. Inventory file variables
3. Inventory group_vars
4. Inventory host_vars
5. Playbook vars
6. set_fact
7. Extra vars (--extra-vars)

You can access variables like:

```
{{ ansible_hostname }}
{{ nginx_port }}
```

🗂️ 6. Group Inventory by Platform

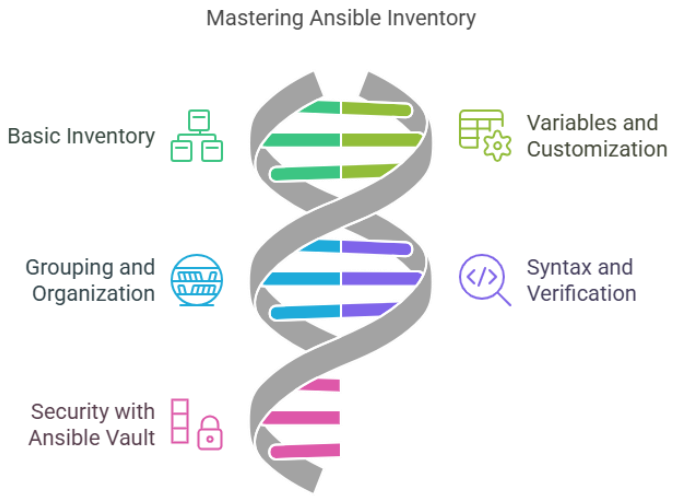
You can group devices based on their platform or OS for better targeting.

```
[cisco_ios]
switch1 ansible_host=10.0.0.1 ansible_network_os=ios
```

```
[juniper_junos]
switch2 ansible_host=10.0.0.2 ansible_network_os=junos

[network_devices:children]
cisco_ios
juniper_junos
```

This allows for playbooks like:

```
- name: Configure all network devices
  hosts: network_devices
  gather_facts: no
  tasks:
    - name: Show version
      ios_facts:
        gather_subset: hardware
```



✅ 7. Verifying the Inventory

Use these commands to verify and test your inventory setup:

```
ansible-inventory -i inventory.ini --list       # JSON dump of inventory
ansible-inventory -i inventory.ini --graph      # Group/host tree
ansible -i inventory.ini all --list-hosts       # List all hosts
```

Useful for debugging inventory parsing or dynamic inventory scripts.

🔐 8. Protecting Sensitive Variables with Ansible Vault

Ansible Vault allows encrypting any file that contains secrets (passwords, keys, tokens).

Encrypt a file:

```
ansible-vault encrypt group_vars/dbservers.yml
```

Use in Playbook:

```
ansible-playbook -i inventory.ini playbook.yml --ask-vault-pass
```

Or use a vault password file:

```
ansible-playbook -i inventory.ini playbook.yml --vault-password-file .vault_pass
```

You can encrypt only parts of a file using **inline vault**:

```
db_password: !vault |
      $ANSIBLE_VAULT;1.1;AES256
      316637386333376163613...
```

🔁 9. Dynamic Inventories (Advanced)

When working with cloud platforms, you can use **dynamic inventory scripts/plugins**.

Example: AWS EC2 plugin (inventory_aws_ec2.yml)

```
plugin: amazon.aws.aws_ec2
regions:
 - us-east-1
filters:
 tag:Environment: dev
```

Run:

```
ansible-inventory -i inventory_aws_ec2.yml --list
```

🧠 Summary Table

Component	Purpose
inventory.ini	Static host inventory
group_vars/	Group-level variables
host_vars/	Host-specific variables
ansible_network_os	Identifies OS for network modules
ansible-vault	Encrypts sensitive variables
ansible-inventory	Validates and lists inventory info
:children	Nest groups within groups

---

3. 🖥️ Ansible Architecture: Master Node, Slave Node, and Tower Deep Dive

🧠 **Ansible Deep Dive: Master Node, Slave Nodes & Tower/AWX**

🎯 **1. What is Ansible Master Node (Control Node)?**

✅ **Definition:**

The **control node** (a.k.a. master node) is the **machine where Ansible is installed** and **commands or playbooks are executed from**. It manages remote systems (managed/slave nodes) over SSH (or WinRM for Windows).

📌 **Characteristics:**

- No agents are installed on managed nodes
- Uses **SSH (Linux/Unix)** or **WinRM (Windows)** to connect
- Push-based model: instructions are sent from master → slaves
- Hosts the **inventory**, **playbooks**, and **ansible.cfg**

🧰 **Common Commands:**

```
ansible-inventory --list
ansible-playbook site.yml
ansible all -m ping
```

🖥️ **2. Managed Nodes (a.k.a. Slave Nodes)**

✅ **Definition:**

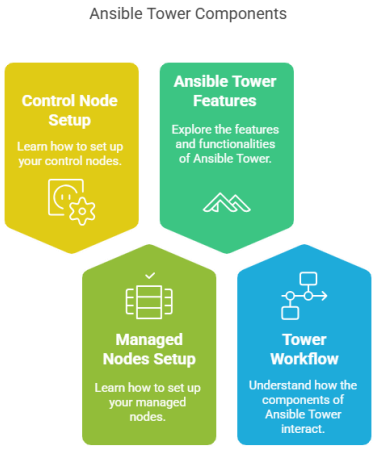
A **managed node** is any **target machine (Linux, Windows, network devices, etc.)** that Ansible controls.

💉 **Requirements:**

- **SSH enabled (Linux) or WinRM (Windows)**
- Python ≥ 2.7 or Python 3 (Linux systems)
- For network devices, may use API or CLI interfaces (platform-specific modules)

🔐 **Authentication Options:**

- SSH keys
- SSH password (not recommended)
- Vault-encrypted credentials
- Ansible Tower credentials manager



🔁 Master ↔ Managed Node Communication Flow

[Control Node (Ansible Master)]

|

| SSH / WinRM

↓

[Managed Node(s) - Servers, Routers, Switches, etc.]

- Communication is **stateless and agentless**
- Uses **YAML playbooks** to send commands or config changes
- No persistent daemon on managed nodes

🏢 3. Ansible Tower / AWX

🏰 What is Ansible Tower?

- **Ansible Tower** is a **web-based UI and REST API** for managing, visualizing, scheduling, and auditing your Ansible automation.
- **AWX** is the **open-source upstream project** of Tower.

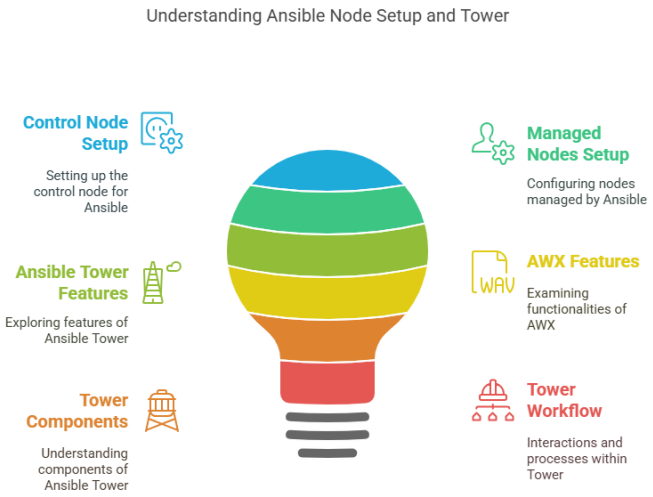
🔍 Features:

- GUI for playbook execution
- Role-based access control (RBAC)
- Credential storage (vaulted)
- Job scheduling & logging
- Workflow visualizer
- REST API & CLI

🧱 Tower Architecture Components

Component	Role
Web UI	GUI frontend
Task Engine	Executes playbooks (uses Ansible under the hood)
Inventory Sync	Pulls inventory from cloud, SCM, CMDB
PostgreSQL DB	Stores users, inventory, job history
RabbitMQ	Internal communication between components
Redis	Task queue backend

🛠️ Tower Workflow Example



[User or API]

↓

[Tower UI or REST API]

↓

[Launch Job Template]

↓

[Uses SSH/WinRM to manage nodes]

↓

[Status + Logs stored in PostgreSQL]

🧪 Key Concepts in Tower/AWX

Concept	Description
Projects	Source of playbooks (Git, SVN, etc.)
Inventory	Target hosts and groups
Credentials	Vaulted passwords, SSH keys, cloud access keys
Job Templates	Predefined jobs with playbook + inventory + credentials
Workflows	Chains of job templates

Concept	Description
Schedules	Automate playbook runs
Surveys	Collect user input for playbooks dynamically

🔐 Credential Types in Tower

- SSH key (RSA, Ed25519)
- Username/password
- AWS IAM keys
- Vault passwords
- Azure, GCP credentials
- Custom credential plugins

📌 Tower vs CLI

Feature	CLI (Core Ansible)	Tower / AWX
GUI support	❌	✅
Role-based Access (RBAC)	❌	✅
Scheduling	❌	✅
Job visualization	❌	✅
API integration	✅ (limited)	✅
Inventory sync (cloud)	Manual	Automated
Vault integration	CLI vault password	Stored + managed securely
Audit logs	❌	✅

📘 When to Use Tower/AWX

✅ Teams collaborating on automation
✅ You need secure credential handling
✅ You want audit trails and scheduling

✅ Large-scale inventory management (cloud dynamic)
✅ REST API integration with external systems

🚀 **Real-World Use Case: CI/CD with Tower**

- Git push to Ansible repo triggers webhook
- Tower syncs Git project
- Tower auto-runs playbook on staging
- Workflow validates configs
- Approval step → production deployment

🛡️ **Security in Tower**

- Role-Based Access Control (RBAC)
- Vault integration for secrets
- Job isolation
- Logging and audit compliance
- LDAP/AD and SAML integration

🎓 **Summary**

Component	Purpose
Master (Control Node)	Executes playbooks and manages inventory
Managed Node	Target systems managed over SSH/WinRM
Tower / AWX	Web interface, scheduling, RBAC, logging, API

---

4. **Ansible playbook with explanation**
    a. **Sample**
    b. **Master and Slaves**
    c. **Ansible Tower**

✅ **Part 1: Full-Fledged Ansible Playbook (Single Playbook)**

📁 **Use Case: Install and configure Apache Web Server on a group of Ubuntu servers, start and enable the service, and deploy a custom index.html.**

📜 **site.yml**

```
---
- name: Install and Configure Apache Web Server
  hosts: webservers
  become: true
  vars:
    apache_package: apache2
    apache_service: apache2
    custom_index: "<h1>Welcome to Ansible Web Server!</h1>"

  tasks:
    - name: Install Apache Package
      apt:
        name: "{{ apache_package }}"
        state: present
        update_cache: true

    - name: Start and Enable Apache Service
      service:
        name: "{{ apache_service }}"
        state: started
        enabled: true

    - name: Deploy custom index.html
      copy:
        content: "{{ custom_index }}"
```

```
    dest: /var/www/html/index.html
    owner: www-data
    group: www-data
    mode: '0644'

  - name: Open port 80 in UFW (if enabled)
    ufw:
      rule: allow
      port: '80'
      proto: tcp
    when: ansible_facts['distribution'] == "Ubuntu"

  - name: Ensure Apache is running
    shell: systemctl status apache2
    register: apache_status
    ignore_errors: yes

  - name: Show Apache status
    debug:
      var: apache_status.stdout
```

🔍 Step-by-Step Explanation

Line	Explanation
hosts: webservers	Applies to the group webservers defined in inventory
become: true	Executes commands as sudo/root
vars:	Sets variables for reuse
apt:	Installs packages (Debian-based systems)
service:	Manages services (start, enable, stop)

Line	Explanation
copy:	Deploys static content
ufw:	Configures firewall
shell:	Runs raw shell commands
register:	Stores command output for later use
debug:	Prints output of previous tasks

✅ **Part 2: 2 Playbooks — Master-Slave Example**

Imagine a scenario with:

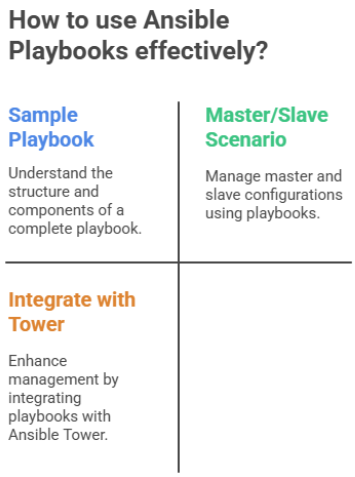
- Master (control node) managing slaves
- Slave group includes appservers and dbservers
- One playbook handles **App server setup**, the other handles **Database setup**

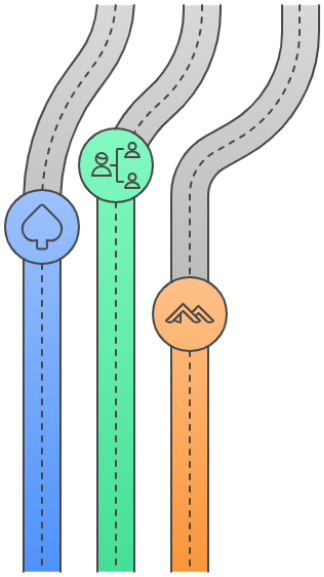
📁 **app.yml – App Server Playbook**

```
---
- name: Deploy Node.js App Server
  hosts: appservers
  become: true

  tasks:
    - name: Install Node.js
      apt:
        name: nodejs
        state: present

    - name: Install npm
      apt:
        name: npm
```

```
    state: present

  - name: Copy application code
    copy:
      src: ./app/
      dest: /opt/app/
```

📁 **db.yml – DB Server Playbook**





```
---
- name: Setup MySQL Database Server
  hosts: dbservers
  become: true

  vars:
    mysql_root_password: "StrongRootPass123"

  tasks:
    - name: Install MySQL Server
      apt:
        name: mysql-server
        state: present

    - name: Secure MySQL Installation
      mysql_secure_installation:
        login_password: ''
        new_password: "{{ mysql_root_password }}"
        validate_password_policy: LOW
        remove_anonymous_users: yes
```

```
    disallow_root_login_remotely: yes
```

📁 **Sample Inventory (inventory.ini)**

```
[appservers]
app01 ansible_host=192.168.10.101

[dbservers]
db01 ansible_host=192.168.10.102
```

📜 **Run from master:**

```
ansible-playbook -i inventory.ini app.yml
ansible-playbook -i inventory.ini db.yml
```

✅ **Part 3: Ansible Tower / AWX Example**

🧱 **Step-by-Step Components in Tower**

🔐 **1. Credential**

- Type: Machine
- SSH Private Key or Password
- Used to access managed hosts

📦 **2. Project**

- Source: Git (e.g., GitHub repo with playbook)
- Syncs the repo regularly or on-demand

📘 **3. Inventory**

- Add hosts manually or use dynamic inventory (AWS, GCP, etc.)
- Add webservers group with host IPs

⚙️ **4. Job Template**

- Ties together:
    - Project
    - Playbook (e.g., site.yml)
    - Inventory
    - Credentials
    - Verbosity
    - Extra Vars (optional)

📁 **Example Git Project Structure**

```
├── site.yml
├── roles/
│   └── apache/
│       ├── tasks/
│       │   └── main.yml
│       └── templates/
│           └── index.html.j2
```

site.yml

```
- name: Web Server Provisioning via Tower
  hosts: webservers
  become: true
  roles:
    - apache
```

roles/apache/tasks/main.yml

```
- name: Install Apache
  apt:
    name: apache2
    state: present
    update_cache: yes
```

```
- name: Deploy templated index.html
  template:
    src: index.html.j2
    dest: /var/www/html/index.html
```

roles/apache/templates/index.html.j2

```
<h1>Welcome to {{ inventory_hostname }} - provisioned by Tower!</h1>
```

🔁 Workflow Summary in AWX

Step	Description
1. Git push to project repo	Tower syncs project
2. Launch job template	Uses inventory + credentials
3. Executes playbook	Playbook provisions Apache
4. Job history + logs	Available in UI
5. RBAC	Limits access to project/resources
6. Notifications	Can integrate Slack, email, Webhook

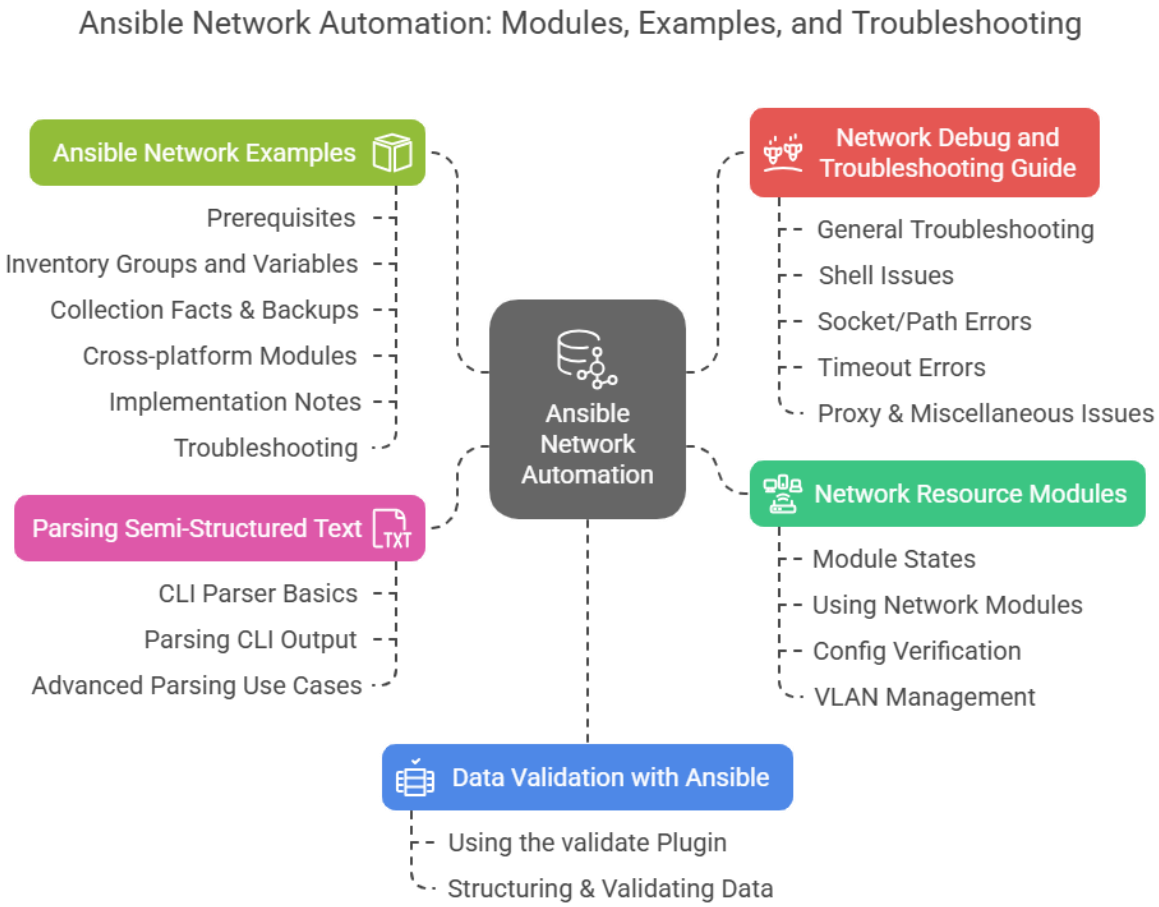
📊 Logs Example in Tower

```
TASK [apache : Install Apache] ***
ok: [web01]

TASK [apache : Deploy templated index.html] ***
changed: [web01]
```

✅ Summary Table

Part	Type	Tools	Description
1	Single Playbook	CLI	Full Apache setup with firewall
2	Master/Slave	CLI	Multiple playbooks by role
3	Tower Integration	Tower/AWX	Git-backed CI/CD-style provisioning with GUI, logging, and RBAC

5. 🌐 Ansible Network Automation: Advanced Topics



👷 Who Should Use This Guide?

This guide is designed for:

- Network Engineers and Architects
- DevNet and NetDevOps professionals
- Those managing **routers, switches, firewalls**, etc.
- Engineers comfortable with **CLI-based network configs** and basic Ansible usage

🔧 Advanced Topics Overview

1. Network Resource Modules

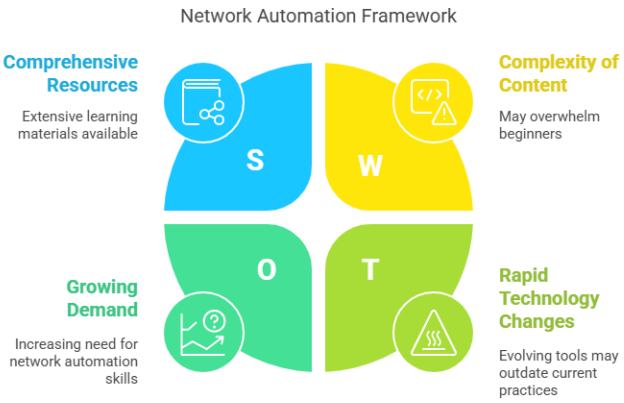
These are **platform-aware modules** that manage specific network resources like VLANs, interfaces, and BGP neighbors.
Unlike raw CLI or command modules, **resource modules** offer idempotency, better structure, and easier error handling.

2. Network Resource Module States

State	Behavior
merged	Add or update values without removing others
replaced	Replace target config fully
overridden	Make current config match desired exactly
deleted	Remove specified config only
rendered	Generate and view config only (dry run)

3. Using Network Resource Modules (Example: VLAN)

```
- name: Manage VLANs
  hosts: cisco_devices
  gather_facts: no
  tasks:
   - name: Ensure VLANs 10-20 are configured
     cisco.ios.ios_vlans:
       config:
         - vlan_id: 10
           name: Users
         - vlan_id: 20
           name: Servers
```



```
    state: merged
```

4. Verify Network Device Configuration Hasn't Changed

```
- name: Verify VLANs unchanged
 hosts: switches
 tasks:
  - name: Backup current VLAN config
    cisco.ios.ios_config:
     backup: yes
  - name: Re-apply known-good config
    cisco.ios.ios_vlans:
     config: "{{ known_vlans }}"
     state: rendered
```

Compare using diff to ensure no drift.

5. Acquiring and Updating VLANs

Use ansible_facts to dynamically retrieve and process existing configuration.

```
- name: Get existing VLANs
 cisco.ios.ios_facts:


- name: Update VLANs based on conditions
 cisco.ios.ios_vlans:
  config: "{{ new_vlan_list }}"
  state: merged
```

📚 **Ansible Network Examples**

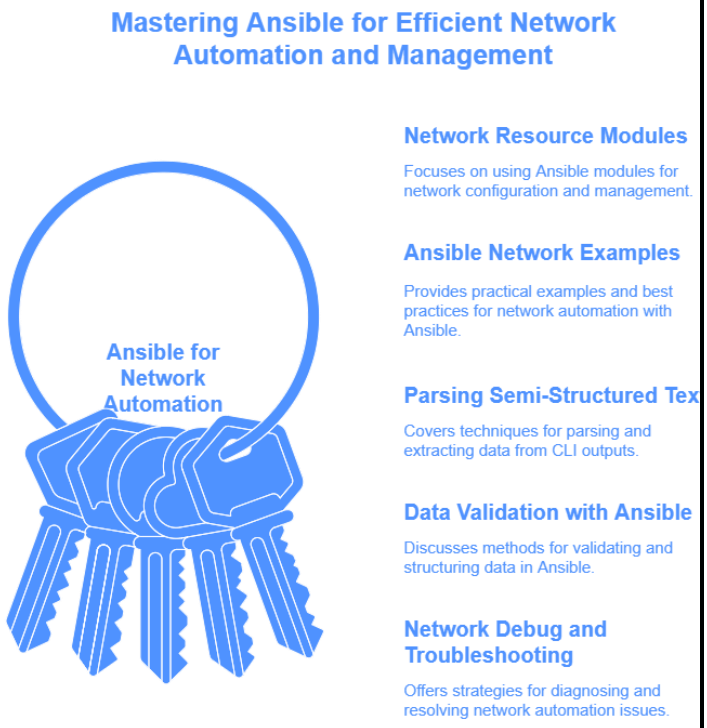
✅ **Prerequisites**

- Ensure proper ansible_connection=network_cli
- Correct ansible_network_os (e.g. ios, eos, nxos, junos, etc.)

- Enable SSH or Telnet access

📁 Groups and Variables in Inventory

```
[cisco_devices]
switch1 ansible_host=10.0.0.1

[cisco_devices:vars]
ansible_user=admin
ansible_password=secret
ansible_network_os=ios
ansible_connection=network_cli
```



💉 Example 1: Collecting Facts + Backups

```
- name: Collect and back up configs
  hosts: all
  gather_facts: no
  tasks:
    - name: Collect facts
      ios_facts:
    - name: Save running config
      ios_config:
        backup: yes
```

🔁 Example 2: Platform-Independent VLAN Management

```
- name: Manage VLANs across platforms
  hosts: all
  tasks:
    - name: Configure VLANs (platform-independent)
```

```
  network_resource:
    resource: vlans
    config:
      - vlan_id: 30
        name: Voice
    state: merged
```

network_resource is a new unified resource abstraction from ansible.netcommon.

📌 Implementation Notes

- Use gather_facts: no for network devices unless specific facts are required.
- Always validate configs in a **dry run** first using state: rendered.
- Platform modules are often split:
  - Cisco: cisco.ios, cisco.nxos
  - Juniper: junipernetworks.junos
  - Arista: arista.eos

🩹 Troubleshooting Network Automation

✅ Common Error Categories

Error Category	Examples
Socket path	"Permission denied" or "Unable to open shell"
Timeouts	"Connection timeout" or "SSH handshake failure"
Playbook logic	Variable undefined, bad loop syntax
Proxy issues	Network unreachable
CLI parsing	Output parsing issues with semi-structured text

🧪 Parsing Semi-Structured CLI Text

🔍 Using TextFSM, TTP, and Ansible Plugins

```
- name: Parse CLI output
  ios_command:
    commands:
      - show ip interface brief
  register: output

- name: Parse using TextFSM
  set_fact:
    interfaces: "{{ output.stdout[0] | parse_cli_textfsm('templates/show_ip_int_brief.textfsm') }}"
```

CLI Parser Plugins:

- parse_cli
- parse_cli_textfsm
- parse_cli_ttp

✔ **Validate Data Against Criteria**

Use the validate plugin to verify configs/data structures before applying them.

🧩 **Steps:**

1. **Structure data** — YAML, JSON, gathered facts
2. **Define criteria** — what is valid vs invalid
3. **Run validations** — fail the play if not met

```
- name: Validate VLAN IDs
  validate:
    data: "{{ vlans }}"
    criteria:
      - expression: vlan_id < 100
        message: "VLAN ID must be under 100"
```

🧠 **Troubleshooting Cheat Sheet**

Symptom	Cause	Fix
Unable to open shell	Bad ansible_connection or privilege escalation	Check inventory connection variables
Timeout	SSH unreachable	Validate IP, user/pass, ACLs
CLI parsing fails	Unsupported CLI output	Customize TextFSM or TTP template
Vault errors	Wrong vault password	Use correct --vault-password-file
Proxy issues	Restricted outbound traffic	Set proper env vars or bypass

🚀 **Ready for Production?**

Before scaling network automation:

- Validate configurations with state: rendered
- Test in lab environment
- Use **diff and backup** options
- Encrypt sensitive data with **Ansible Vault**
- Use **idempotent** modules (like ios_interfaces, ios_vlans)

THANK YOU

LINKEDIN: WWW.LINKEDIN.COM/IN/MACHAN-VISHAL

MAIL VISHALMACHAN15@GMAIL.COM

CALL (217)5889385