Ansible

1. Ansible Overview & Architecture

What is Ansible?
Ansible is an open-source automation and configuration management tool used to automate IT tasks such as application deployment, system configuration, orchestration, and provisioning. It is agentless and works primarily over SSH, making it simple to set up and operate.

Ansible follows a push-based model, where a central control node pushes configurations and commands to managed nodes over SSH.

Key Characteristics of Ansible
Agentless: No software needs to be installed on managed nodes
Uses SSH: Secure and simple connectivity
YAML-based: Human-readable playbooks
Idempotent: Safe to run multiple times without unintended changes
Modular: Uses reusable modules and roles
Easy learning curve compared to other automation tools

Ansible Architecture
1. Control Node
Machine where Ansible is installed
Executes playbooks and ad-hoc commands
Requires Python and Ansible

2. Managed Nodes
Target servers managed by Ansible
Require only SSH access and Python
No Ansible agent required

3. Inventory
List of managed nodes
Can be static (INI/YAML) or dynamic (cloud-based)

4. Playbooks
YAML files defining automation logic
Consist of plays, tasks, modules, and variables

How Ansible Works (High Level Flow)
User runs an Ansible command or playbook
Ansible reads inventory and configuration
SSH connection is established to managed nodes
Modules are executed on target systems
Results are returned to the control node

Common Use Cases of Ansible

1. Configuration Management
Install and configure packages
Manage system files like /etc/hosts, sshd_config
Enforce system state across servers

Example:
Ensure NGINX is installed on all servers

2. Application Deployment
Deploy applications consistently
Restart services when configuration changes
Roll out updates across multiple servers

Example:
Deploy Java, Python, or Node.js applications

3. Server Provisioning
Prepare servers after OS installation
Install required packages, users, and configurations

Example:
Provision EC2 instances after launch

4. Orchestration
Coordinate multi-tier applications
Control the order of operations across systems

Example:
Start database → backend → frontend services

5. Patch Management
Apply OS updates across servers
Ensure compliance with security policies

6. Cloud Automation
Manage cloud resources using modules
Integrates with AWS, Azure, GCP

Example:
Create EC2 instances, security groups, load balancers

7. Continuous Deployment / DevOps

Works with CI/CD pipelines
Automates infrastructure and application delivery

Ansible vs Traditional Scripts

Feature	Shell Script	Ansible
Readability	Low	High (YAML)
Idempotency	No	Yes
Error Handling	Manual	Built-in
Scalability	Poor	Excellent
Reusability	Low	High

Why Ansible is Popular
Simple to learn and use
No agents required
Strong community support
Widely used in DevOps and Cloud environments

Summary:
Ansible is a powerful, simple, and agentless automation tool used to manage infrastructure, deploy applications, and orchestrate complex workflows. It is widely adopted in modern DevOps practices due to its ease of use and flexibility.

2. Lab Architecture Setup

Below is a simple Ansible lab architecture using **3 Ubuntu EC2 instances**.

```
Ansible Control Node (Ubuntu)
        |
        |-- SSH
        |
  ---------------------
  |                   |
Client-1 (Ubuntu)   Client-2 (Ubuntu)
```

Step 1: Launch EC2 Instances
Create 3 Ubuntu EC2 instances:
Instance 1 → Ansible Server (Control Node)
Instance 2 → Client Server 1 (Managed Node)
Instance 3 → Client Server 2 (Managed Node)
👉 Use the same key pair for all 3 instances (simplifies SSH access).

Security Group Configuration
Allow SSH (port 22) access:
From your public IP → to access the Ansible control node
From Ansible control node's private IP → to access client servers

This ensures:
Secure access from your system
Controlled SSH communication between Ansible server and clients

Set a hostname for all 3 servers

Set hostname for ansible server:
hostnamectl set-hostname ansible

```
output:
root@ip-172-31-28-167:~# hostnamectl set-hostname ansible
root@ip-172-31-28-167:~# hostname
ansible
root@ip-172-31-28-167:~#

Set a hostname for client servers:
Output:
root@ip-172-31-23-217:~# hostnamectl set-hostname linux1
root@ip-172-31-23-217:~# hostname
linux1
root@ip-172-31-23-217:~#

root@ip-172-31-23-239:~# hostnamectl set-hostname linux2
root@ip-172-31-23-239:~# hostname
linux2
root@ip-172-31-23-239:~#
```

Create ansible user, enable passwordless sudo, and enable password-based SSH login

Create an ansible user:
sudo useradd -m ansible
sudo passwd ansible
sudo usermod -aG sudo ansible
Options explained: -m to create a home directory
-G = supplementary group(s)
-a = add without deleting existing groups

Enable passwordless sudo:
sudo visudo
Add:
ansible ALL=(ALL) NOPASSWD:ALL

Enable password-based SSH login
Update the below two files.

sudo nano /etc/ssh/sshd_config
sudo nano /etc/ssh/sshd_config.d/60-cloudimg-settings.conf
set :

```
PasswordAuthentication yes
```

And keep PermitRootLogin unchanged (whatever the system default is).

- Restart SSH:

```
sudo systemctl restart sshd
```

- Now you can log in with the password you set for ansible.

⚠ Password-based SSH is enabled only for lab/demo purposes. In production environments, key-based authentication, restricted sudo access, and hardened SSH settings must be used.

Generate an SSH key and copy it to client servers for SSH key-based (passwordless) authentication:
Login to Ansible server with the ansible user.
ssh-keygen

```
Note:
By default, SSH keys are generated under ~/.ssh/id_rsa and ~/.ssh/id_rsa.pub
```

After key generation, copy to client servers.
ssh-copy-id ansible@linux1
ssh-copy-id ansible@linux2

Verify passwordless SSH authentication from the Ansible server:
ssh ansible@linux1
ssh ansible@linux2
You should be able to log in without entering a password.

---

3. Inventory Management

Check default inventory file
Run:
ls -l /etc/ansible/hosts
Case A: File does NOT exist

Create it:
sudo mkdir -p /etc/ansible
sudo nano /etc/ansible/hosts

Case B: File exists but empty
Put this exact content (replace IPs with yours):
Open it:
sudo nano /etc/ansible/hosts

[clients]
linux1 ansible_host=54.87.137.58 ansible_user=ansible
linux2 ansible_host=54.146.216.14 ansible_user=ansible

/etc/ansible/hosts is the default inventory file, so Ansible will automatically use it without specifying -i.

Verify inventory parsing:
ansible-inventory --list

```
Output:
ansible@ansible:~$ ansible-inventory --list
{
    "_meta": {
        "hostvars": {
            "linux1": {
                "ansible_host": "54.87.137.58",
                "ansible_user": "ansible"
```

```
            },
            "linux2": {
                "ansible_host": "54.146.216.14",
                "ansible_user": "ansible"
            }
        }
    },
    "all": {
        "children": [
            "clients",
            "ungrouped"
        ]
    },
    "clients": {
        "hosts": [
            "linux1",
            "linux2"
        ]
    }
}
ansible@ansible:~$
```

In real projects, it is recommended to use project-level inventory files instead of modifying /etc/ansible/hosts

Run a ping test to verify connectivity with managed nodes.

```
ansible@ansible:~$ ansible all -m ping
linux1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
linux2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
ansible@ansible:~$
```

The above command uses the default inventory file /etc/ansible/hosts.

Alternative approach (if you don't want default inventory)
You can also test using a custom inventory file:
Create a custom inventory file and update it with the following entries: inventory.ini

```
[clients]
linux1 ansible_host=54.87.137.58 ansible_user=ansible
linux2 ansible_host=54.146.216.14 ansible_user=ansible
```

Ping the client servers:
ansible@ansible:~$ ansible -i inventory.ini all -m ping

```
linux1 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
linux2 | SUCCESS => {
    "ansible_facts": {
        "discovered_interpreter_python": "/usr/bin/python3"
    },
    "changed": false,
    "ping": "pong"
}
ansible@ansible:~$
```

---

4. Ad-hoc Commands

Use the command or shell module.
Recommended (safe):
ansible -i inventory.ini all -m command -a uptime

The shell module should be used only when pipes (|), redirection (>),
or other shell features are required; otherwise, prefer the command module.

OR (simpler shortcut):
ansible -i inventory.ini all -a "uptime"

```
Both will work.

Output:
ansible@ansible:~$ ansible -i inventory.ini all -m command -a uptime
linux2 | CHANGED | rc=0 >>
 18:15:46 up  2:52,  8 users,  load average: 0.00, 0.00, 0.00
linux1 | CHANGED | rc=0 >>
 18:15:46 up  2:52,  9 users,  load average: 0.00, 0.00, 0.00
ansible@ansible:~$ ansible -i inventory.ini all -a uptime
linux1 | CHANGED | rc=0 >>
 18:15:57 up  2:52,  9 users,  load average: 0.00, 0.00, 0.00
linux2 | CHANGED | rc=0 >>
 18:15:57 up  2:52,  8 users,  load average: 0.00, 0.00, 0.00
ansible@ansible:~$
```

🧠 Golden rules (important)

Use case	Rule
Linux command	-a "command"
Ansible module	-m module_name
Pipes / redirects	Use shell
Default module	command
Custom inventory	Use -i

If it’s an Ansible module, use -m module_name
If it’s a Linux command, use -a "command"

5. Idempotency & Package Management

Package installation in Ansible (Ad-hoc first)

Since I am on Ubuntu, used the apt module (not command).

✅ 1️⃣ **Update package cache (best practice)**

ansible -i inventory.ini all -m apt -a "update_cache=yes" -b

apt → Ansible package module

update_cache=yes → apt update

-b → become (sudo)

✅ 2️⃣ **Install a package (example: nginx)**

ansible -i inventory.ini all -m apt -a "name=nginx state=present" -b

✅ 3️⃣ **Verify package installation**

ansible -i inventory.ini all -a "nginx -v"

or

ansible -i inventory.ini all -a "systemctl status nginx"

🔹 **Why NOT use command for package install** ❌

Not recommended practice:

ansible all -a "apt install nginx -y"

Why?

Not idempotent

No state tracking

Breaks Ansible philosophy

Run this twice:

ansible -i inventory.ini all -m apt -a "name=nginx state=present" -b

Second run:

No changes

No reinstallation

👉 This is **idempotency**.

🧠 **Golden rule (remember this)**

Always use Ansible modules (apt, yum, dnf) instead of shell/command for package management.

Install multiple packages

ansible -i inventory.ini all -m apt -a "name=vim,git,curl state=present" -b

---

6. Playbooks (Nginx & Apache)

Simple Ansible playbook to install and start Nginx on both linux1 and linux2.

nginx.yaml:

```
ansible@ansible:~$ cat nginx.yaml
---
- name: Install and start Nginx on web servers
  hosts: all
  become: yes

  tasks:
    - name: update apt cache
      apt:
        update_cache: yes
        cache_valid_time: 3600

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

    - name: Ensure Nginx is started and enabled
      service:
        name: nginx
        state: started
        enabled: yes

ansible@ansible:~$
```

Output:

```
ansible@ansible:~$ ansible-playbook -i inventory.ini nginx.yaml

PLAY [Install and start Nginx on web servers] **********************************************************

TASK [Gathering Facts] *********************************************************************************
ok: [linux1]
ok: [linux2]

TASK [update apt cache] ********************************************************************************
changed: [linux1]
changed: [linux2]

TASK [Install Nginx] ***********************************************************************************
ok: [linux2]
ok: [linux1]

TASK [Ensure Nginx is started and enabled] *************************************************************
ok: [linux2]
ok: [linux1]

PLAY RECAP *********************************************************************************************
linux1                     : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
linux2                     : ok=4    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

Check the status:

ansible -i inventory.ini all -m command -a "systemctl is-active nginx --no-pager"

```
ansible@ansible:~$ ansible -i inventory.ini all -m command -a "systemctl is-active nginx --no-pager"
linux2 | CHANGED | rc=0 >>
active
linux1 | CHANGED | rc=0 >>
active
ansible@ansible:~$
```

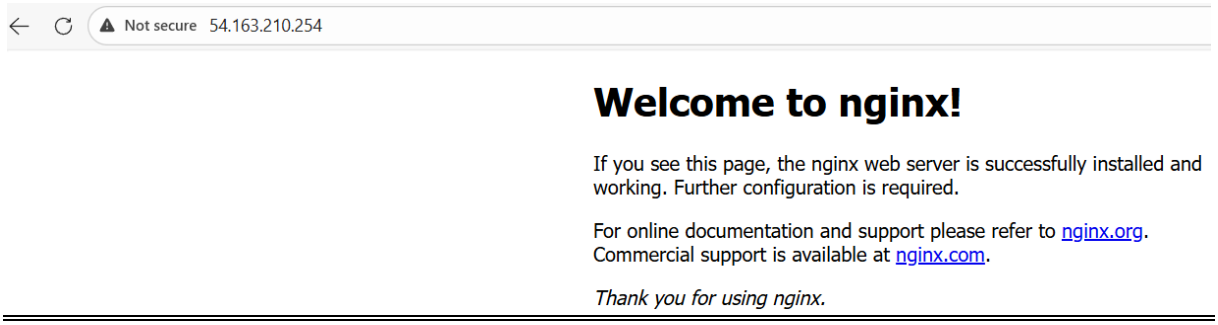
ansible -i inventory.ini all -m command -a "curl -I localhost"

Expected:

HTTP/1.1 200 OK

Verify from browser (optional)

If instances have public IP + security group allows port 80: http://<public-ip>



Deployed a Custom Nginx Web Page Using Ansible

```
ansible@ansible:~$ cat webpage.yaml
---
- name: Deploy Nginx Web Server
  hosts: all
  become: yes

  tasks:
    - name: Create a custom index.html
      copy:
        dest: /var/www/html/index.html
        content: |
          <html>
            <head>
              <title>Ansible Nginx Demo!</title>
            </head>
            <body>
              <h1>Deployed via Ansible playbook.</h1>
            </body>
          </html>
        owner: root
        group: root
        mode: '0644'
```

Output:

```
ansible@ansible:~$ ansible-playbook -i inventory.ini webpage.yaml

PLAY [Deploy Nginx Web Server] ***********************************************************************************

TASK [Gathering Facts] *******************************************************************************************
ok: [linux2]
ok: [linux1]

TASK [Create a custom index.html] ********************************************************************************
changed: [linux2]
changed: [linux1]

PLAY RECAP *******************************************************************************************************
linux1                     : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
linux2                     : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

ansible@ansible:~$
```

Webpage output:

Amazon linux servers

Update the yum, install and start the httpd service

```
[ansible@ansible ~]$ cat apache.yaml
- name: Install and start Apache on Amazon servers
  hosts: all
  become: yes

  tasks:
    - name: update yum cache
      yum:
        update_cache: yes

    - name: Install Apache
      yum:
        name: httpd
        state: present

    - name: Start and enable apache service
      service:
        name: httpd
        state: started
        enabled: yes
```

Playbook output:

```
[ansible@ansible ~]$ ansible-playbook apache.yaml

PLAY [Install and start Apache on Amazon servers] ******************************************************************

TASK [Gathering Facts] *********************************************************************************************
ok: [linux1]
ok: [linux2]

TASK [update yum cache] ********************************************************************************************
ok: [linux2]
ok: [linux1]

TASK [Install Apache] **********************************************************************************************
changed: [linux2]
changed: [linux1]

TASK [Start and enable apache service] *****************************************************************************
changed: [linux1]
changed: [linux2]

PLAY RECAP *********************************************************************************************************
linux1                     : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
linux2                     : ok=4    changed=2    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0

[ansible@ansible ~]$
```

To check the status:

ansible -i inventory.ini all -m command -a "systemctl status httpd" -b

Create a user and update a password

```
[ansible@ansible ~]$ cat user.yaml
---
- name: Create a user
  hosts: all
  become: yes

  vars_prompt:
    - name: username
      prompt: "Enter the username to create"
      private: no

    - name: user_password
      prompt: "Enter the password for the user"
      private: yes

  tasks:
    - name: Create user with password
      user:
        name: "{{ username }}"
        state: present
        create_home: yes
        shell: /bin/bash
        password: "{{ user_password | password_hash('sha512') }}"
```

Output:

```
PLAY RECAP *********************************************************************************************************
linux1                     : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
linux2                     : ok=2    changed=1    unreachable=0    failed=0    skipped=0    rescued=0    ignored=0
```

```
[ansible@ansible ~]$ ansible -i inventory.ini all -m command -a "id -a test" -b
linux2 | CHANGED | rc=0 >>
uid=1002(test) gid=1002(test) groups=1002(test)
linux1 | CHANGED | rc=0 >>
uid=1002(test) gid=1002(test) groups=1002(test)
[ansible@ansible ~]$ ansible -i inventory.ini all -m command -a "grep -i test /etc/passwd" -b
linux1 | CHANGED | rc=0 >>
test:x:1002:1002::/home/test:/bin/bash
linux2 | CHANGED | rc=0 >>
test:x:1002:1002::/home/test:/bin/bash
[ansible@ansible ~]$
```

7. Environment Segregation

- **Environment-based grouping: Organizing hosts into groups that represent different environments (development, testing, staging/preprod, production).**
- **Inventory grouping: Using Ansible's inventory file to logically separate servers by purpose or lifecycle stage.**
- **Best practice in Infrastructure as Code (IaC): Keeps automation safe and controlled, so you can test in Dev before applying to Prod.**

Inventory.ini file

```
[ansible@ansible ~]$ cat inventory.ini
[all]

[dev]
linux1 ansible_host=172.31.20.247 ansible_python_interpreter=/usr/bin/python3.9
linux2 ansible_host=172.31.31.229 ansible_python_interpreter=/usr/bin/python3.9

[preprod]
debian ansible_host=172.31.18.123 ansible_python_interpreter=/usr/bin/python3

[prod]
suse ansible_host=172.31.16.253 ansible_python_interpreter=/usr/bin/python3
```

Ping output for different environments:

```
[ansible@ansible ~]$ ansible -i inventory.ini dev -m ping
linux2 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
linux1 | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
[ansible@ansible ~]$ ansible -i inventory.ini preprod -m ping
debian | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
[ansible@ansible ~]$ ansible -i inventory.ini prod -m ping
suse | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

8. Ansible Facts – Concept and Real Use Cases

1️⃣ What are Ansible Facts?

Ansible Facts are system-generated information automatically collected from managed nodes when a playbook runs with gather_facts: yes.

Facts describe the current state of a host, such as:

Hostname

OS and version

IP addresses

CPU, memory, disk

Network interfaces

Architecture

Virtualization details

📌 **Facts are collected using the setup module.**

2️⃣ Why Facts Exist (Core Purpose)

Facts allow Ansible to make dynamic, intelligent decisions instead of hard-coding values.

Without facts → static automation

With facts → adaptive automation

3️⃣ Common Examples of Ansible Facts

Fact	Description
ansible_hostname	Short hostname
ansible_fqdn	Fully qualified domain name
ansible_os_family	RedHat / Debian
ansible_distribution	Amazon / Ubuntu / CentOS
ansible_distribution_version	OS version
ansible_default_ipv4.address	Primary IP
ansible_processor_vcpus	CPU count
ansible_memtotal_mb	Total memory
ansible_mounts	Mounted filesystems
ansible_virtualization_type	KVM / Xen / EC2

4️⃣ How Facts Are Collected

Default behavior

```
- hosts: all
  gather_facts: yes
```

➡️ Ansible runs:

setup

Manual collection (ad-hoc)

```
ansible all -m setup
```

Filter specific facts

```
ansible all -m setup -a "filter=ansible_distribution*"
```

5️⃣ Real Use Cases of Facts (Very Important)

🔹 Use Case 1: OS-based Task Execution

Problem: Same playbook, different OS.

```
- name: Install web server
  package:
    name: httpd
    state: present
  when: ansible_os_family == "RedHat"
- name: Install web server
  apt:
    name: apache2
    state: present
  when: ansible_os_family == "Debian"
```

📌 **Why facts matter:**

One playbook → works everywhere.

🔹 **Use Case 2: Environment Validation (Dev vs Prod)**

```
- name: Display environment details
  debug:
    msg:
      - "Env: {{ env }}"
      - "Host: {{ ansible_hostname }}"
      - "IP: {{ ansible_default_ipv4.address }}"
```

📌 Used for:
Deployment validation
Change records
Audit evidence

🔹 **Use Case 3: Conditional Configuration Based on Memory**

```
- name: Tune JVM heap
  template:
    src: jvm.options.j2
    dest: /etc/jvm.options
  when: ansible_memtotal_mb > 8192
```

📌 Real-world example:
Dev → 4GB
Prod → 16GB
➡️ Same role, different tuning.

🔹 **Use Case 4: Network-Aware Configuration**

```
- name: Bind application to primary IP
  lineinfile:
    path: /etc/app/config.conf
    line: "bind_ip={{ ansible_default_ipv4.address }}"
```

📌 Prevents:
Hardcoded IPs
Wrong interface binding

🔹 **Use Case 5: Cloud vs On-Prem Detection**

```
- name: EC2-specific task
  debug:
    msg: "Running on AWS EC2"
  when: ansible_virtualization_type == "xen"
```

📌 Used for:
Cloud-only agents
Monitoring plugins
Cloud metadata scripts

🔹 **Use Case 6: Disk & Mount Validation (Prechecks)**

```
- name: Fail if /var is less than 10GB
  fail:
    msg: "/var filesystem too small"
  when: >
    ansible_mounts
    | selectattr('mount', 'equalto', '/var')
    | map(attribute='size_available')
    | first < 10737418240
```

📌 Used in:
Pre-deployment checks
Production change approvals

6️⃣ **Facts for Documentation & Audits** 📄
Example: Auto-generate documentation

```
- name: Capture system details
  copy:
    dest: /tmp/system_facts.txt
    content: |
      Hostname: {{ ansible_hostname }}
      OS: {{ ansible_distribution }} {{ ansible_distribution_version }}
      IP: {{ ansible_default_ipv4.address }}
      CPU: {{ ansible_processor_vcpus }}
      Memory: {{ ansible_memtotal_mb }} MB
```

📌 Used for:

CMDB updates

Handover docs

Compliance audits

7️⃣ **Performance Consideration**

Facts collection takes time, especially on large inventories.

Disable when not needed:

```
gather_facts: no
```

Collect only required facts:

```
- setup:
    filter:
      - ansible_distribution
      - ansible_memtotal_mb
```

Install web server based on OS family

```
[ansible@ansible ~]$ cat apache_os.yaml
---
- name: Install web server based on OS family
  hosts: all
  become: yes

  tasks:
    - name: Install web server on RedHat servers
      yum:
        name: httpd
        state: present
      when: ansible_os_family == "RedHat"

    - name: Install web server on Debian servers
      apt:
        name: apache2
        state: present
      when: ansible_os_family == "Debian"

    - name: Install web server on SUSE servers
      zypper:
        name: apache2
        state: present
      when: ansible_os_family == "Suse"
```

Output:

```
[ansible@ansible ~]$ ansible-playbook apache_os.yaml

PLAY [Install web server based on OS family] ***********************************

TASK [Gathering Facts] *********************************************************
ok: [debian]
ok: [linux1]
ok: [linux2]
ok: [suse]

TASK [Install web server on RedHat servers] ************************************
skipping: [debian]
skipping: [suse]
ok: [linux1]
ok: [linux2]

TASK [Install web server on Debian servers] ************************************
skipping: [linux1]
skipping: [linux2]
skipping: [suse]
changed: [debian]

TASK [Install web server on SUSE servers] **************************************
skipping: [linux1]
skipping: [linux2]
skipping: [debian]
changed: [suse]

PLAY RECAP *********************************************************************
debian                     : ok=2    changed=1    unreachable=0    failed=0    skipped=2
linux1                     : ok=2    changed=0    unreachable=0    failed=0    skipped=2
linux2                     : ok=2    changed=0    unreachable=0    failed=0    skipped=2
suse                       : ok=2    changed=1    unreachable=0    failed=0    skipped=2
```

---

9. Status Codes (changed vs ok)

changed vs ok (Deep Meaning)

Status	Meaning
changed	Ansible **modified** the system
ok	System already matched desired state
skipped	Condition was false
failed	Task failed
unreachable	Host not reachable
