# DevOps Foundation: System Configuration, Setup & Overview

**1. Introduction**

In any DevOps lifecycle, proper system configuration and setup form the backbone of a smooth, secure, and scalable automation process. This section introduces the foundational system-level concepts that are vital before deploying DevOps tools and pipelines. These include:

- Mapping hostnames using the /etc/hosts file for simplified connectivity.
- Understanding the difference between control nodes (where automation is initiated) and managed nodes (which receive configuration).
- Launching and accessing AWS EC2 instances securely using SSH.
- Defining an Ansible inventory to manage infrastructure at scale.
- Configuring the SSH daemon (sshd) for secure access and compliance.

With this foundational knowledge, you'll be equipped to set up reliable environments that support automation tools like Ansible, Jenkins, Terraform, and more. These setups ensure smoother configuration management, deployment, and scaling in real-world DevOps projects.

---

**2. System Configuration & Setup**

**2.1 Configuring Host File (/etc/hosts)**

- The /etc/hosts file is used to map IP addresses to hostnames, acting as a local DNS system.
- It's particularly useful in lab setups, testing environments, or in networks without DNS.
- Format:
- <IP Address>  <Fully Qualified Domain Name>  <Alias>

**Example:**

192.168.1.100  webserver.local  webserver

192.168.1.101  dbserver.local  dbserver

- Benefits:
    - Reduces reliance on external DNS
    - Faster hostname resolution in internal environments

---

**2.2 Control Node vs Managed Nodes**

- **Control Node**: The machine where automation tools like Ansible are installed and executed.
- **Managed Nodes**: The machines (servers/VMs) that are configured or managed by the control node.

**Differences:**

| Feature | Control Node | Managed Node |
|---|---|---|
| Role | Executes automation tasks | Receives tasks |
| Tool Installed | Ansible, Terraform, etc. | Requires SSH access |
| OS Dependency | Linux preferred | Any OS that supports SSH |

---

**2.3 AWS EC2 Instance Launch & SSH Remote Login**

- Launch EC2 using AWS Console:
    - Choose AMI: Amazon Linux 2 / Ubuntu / RHEL 9
    - Configure Instance: VPC, Subnet, IAM Role
    - Add storage and tags
    - Configure Security Group (allow SSH - port 22)
    - Launch using a Key Pair (.pem file)

**Remote Access via SSH:**

chmod 400 mykey.pem

ssh -i "mykey.pem" ec2-user@<EC2-Public-IP>

- Default usernames:
    - Amazon Linux: ec2-user
    - Ubuntu: ubuntu
    - RHEL: ec2-user or root

**Security Best Practices:**

- Disable root login
- Use SSH key authentication only
- Regularly rotate keys

---

**2.4 Ansible Inventory (Host List) Setup**

- Inventory file tells Ansible which machines to manage
- Static inventory: manually defined IPs/hostnames
- Dynamic inventory: scripts or cloud APIs to auto-fetch hosts

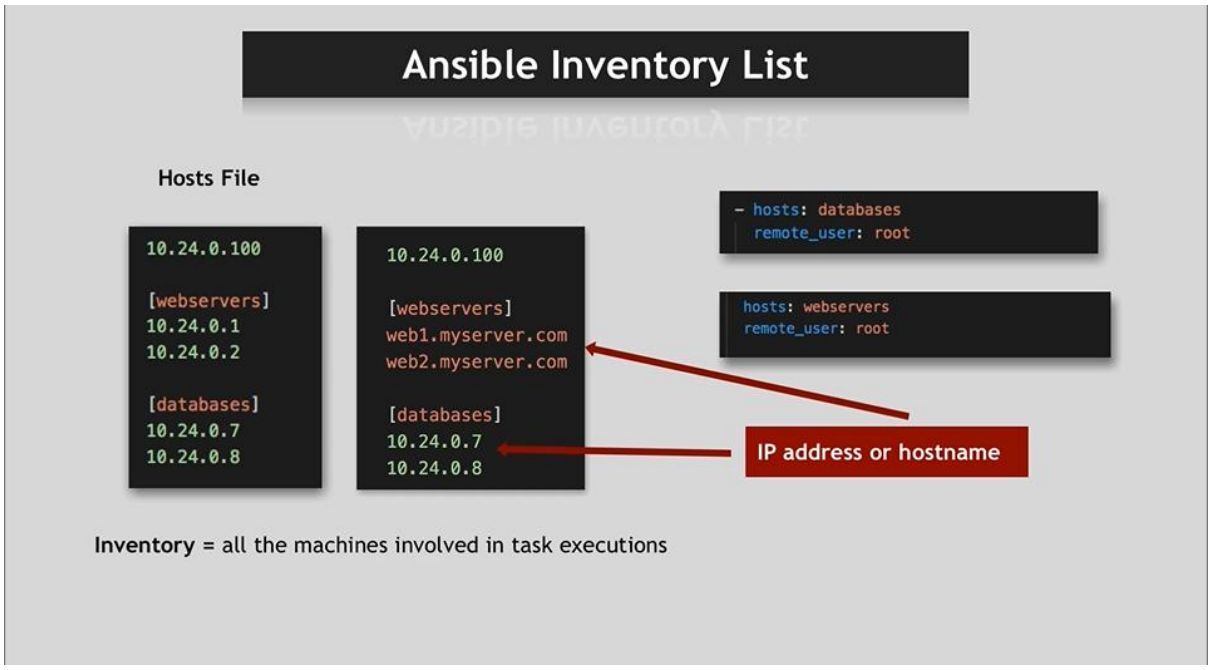
**Static Inventory Example (/etc/ansible/hosts):**

[webservers]

192.168.1.101 ansible_user=ec2-user ansible_ssh_private_key_file=~/mykey.pem

**Best Practices:**

- Use groups (e.g., [dbservers], [loadbalancers])




- Separate inventory files for environments (dev, staging, prod)

---

**2.5 sshd Configuration**

- The SSH Daemon (sshd) handles incoming SSH connections
- Configuration File: /etc/ssh/sshd_config

**Important Parameters:**

- PermitRootLogin no: Prevent direct root access
- PasswordAuthentication no: Force key-based login
- Port 22: Change if needed for security

**Commands:**

sudo nano /etc/ssh/sshd_config

sudo systemctl restart sshd

sudo systemctl status sshd

---

**3. DevOps Overview**

**3.1 DevOps Toolchain Introduction**
DevOps integrates development and operations for faster, reliable software delivery. The toolchain supports stages like planning, building, testing, release, deployment, monitoring.

**Typical DevOps Toolchain:**

- **Plan:** Jira, Trello
- **Code:** Git, GitHub, GitLab
- **Build:** Maven, Gradle
- **Test:** JUnit, Selenium
- **Release:** Jenkins, GitHub Actions
- **Deploy:** Ansible, Kubernetes, Terraform
- **Operate:** Docker, Kubernetes
- **Monitor:** Prometheus, Grafana

---

**3.2 Configuration Types: Scripted vs Intelligent**

| **Feature** | **Scripted Configurations** | **Intelligent Configurations** |
|---|---|---|
| Type | Imperative | Declarative |
| Tools Used | Bash, Shell Scripts | Ansible, Terraform |
| Scalability | Low | High |
| Reusability | Limited | High |
| Idempotency | Not Guaranteed | Guaranteed |

**Example:**

- **Scripted:**

sudo yum install httpd -y

sudo systemctl start httpd

- **Intelligent (Ansible):**

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

---

**3.3 RHEL 9 Setup on Cloud/VM**

- Download RHEL 9 ISO from Red Hat Developer Portal
- Create Virtual Machine in VirtualBox/VMware or use AWS EC2

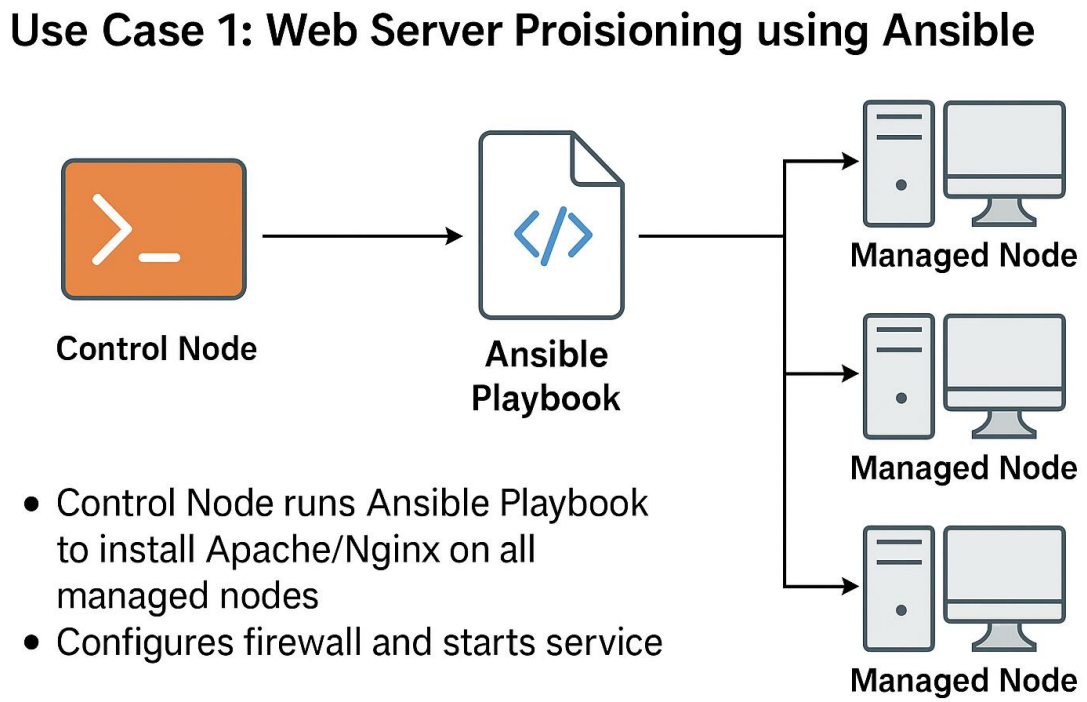
**Post-Installation Configuration:**

- Set hostname: hostnamectl set-hostname rhel9
- Update system: dnf update -y
- Enable firewall: systemctl enable --now firewalld
- Add user and set privileges
- Install common tools: vim, net-tools, git, wget

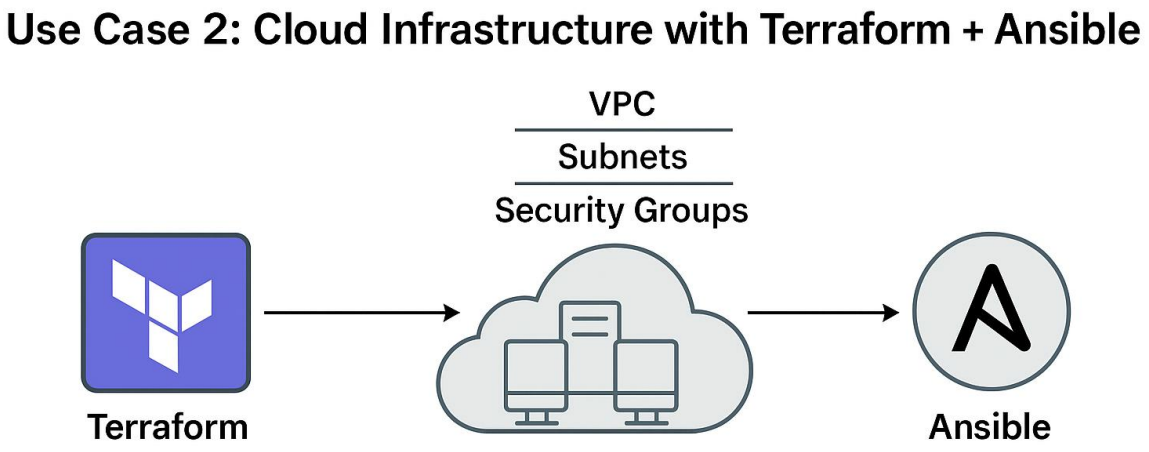
---

**4. Real-World Use Cases**

**Use Case 1: Web Server Provisioning using Ansible**

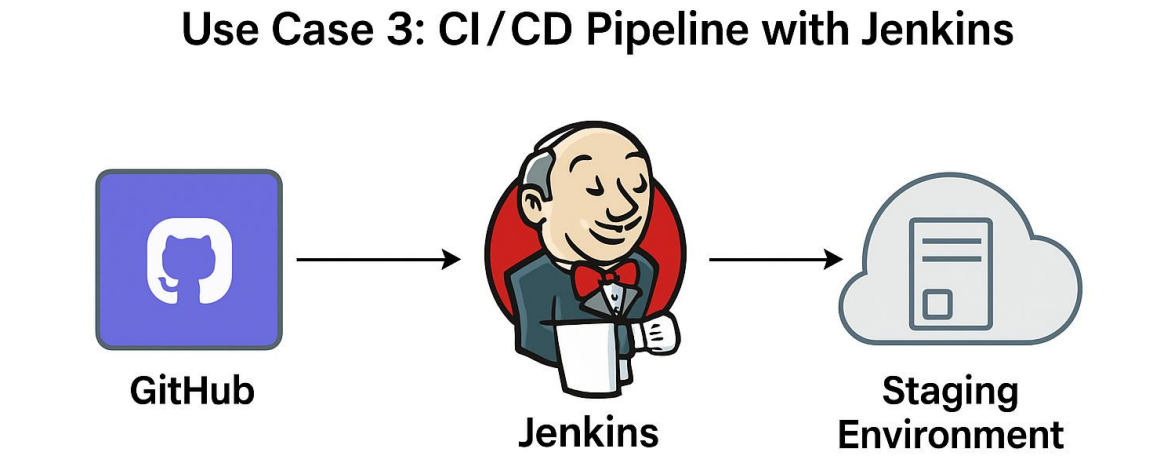


**Use Case 2: Cloud Infrastructure with Terraform + Ansible**




- Terraform: Provisions EC2, VPC, Subnets, Security Groups
- Ansible: Installs apps, configures services on those EC2s
- Scalable, repeatable, automated infrastructure setup

**Use Case 3: CI/CD Pipeline with Jenkins**




- Developer pushes code to GitHub
- Jenkins auto-builds, tests, and deploys to staging environment using Ansible

**5. Summary**

- Covered system-level configurations necessary for DevOps workflows
- Understood difference between control and managed nodes
- Explored SSH setup, Ansible inventory, and configuration types
- Reviewed RHEL setup and real-world use cases to apply these tools

---

**6. Q&A**

---

✅ **Q1: What is an Ansible Control Node and how does it communicate with Managed Nodes?**

**Answer:**

- The **Control Node** is the machine where Ansible is installed and from which playbooks are run.
- It communicates with **Managed Nodes** (remote servers) via **SSH** protocol using key-based authentication.
- No agent is needed on the managed nodes.
- Ansible uses the **inventory file** to know the IP addresses or hostnames of the managed nodes.

---

✅ **Q2: How does Ansible ensure idempotency in configuration management?**

**Answer:**

- **Idempotency** means running the same playbook multiple times will not change the system state if it's already in the desired state.
- Ansible modules (like yum, apt, service, file, etc.) are **declarative**—they check if the task is already done and only act if a change is needed.
- For example:

```
- name: Ensure Apache is installed
  yum:
    name: httpd
    state: present
```

This ensures Apache is installed; if already present, nothing changes.