Day - 9
31 May 2025 15:12

Completed

1. Devops - Fundamental.
2. CI/CD Pipeline.
3. cloud engineering using GCP.
4. Docker.
5. Terraform.

Pending:-

1. Kubernetes (K8)
2. observability (Splunk, Dynatrace)
3. Best Practices & Governance.

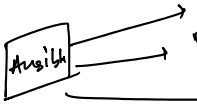
1. CI/CD Best Practices.
2. Introduction to Ansible for Configuration Management -

1. Ansible:-

1. IT Automation engine used for Configuration Management -
2. RedHat → IBM.
3. Core features:-
   1. Written in Python.
   2. Uses YAML for Config (Playbooks)
   3. SSH to communicate with target machines.
   4. Push-based model

Why?

1. Agentless:- Doesnot require agent or daemon to be installed in target windows
2. Idempotent:- Running playbook → same result everytime.
3. Simple yaml Syntax:- yaml → Automate the tasks → playbook



Readable & writeable by human.

Declarative style → What to Achieve & Not how to do it.

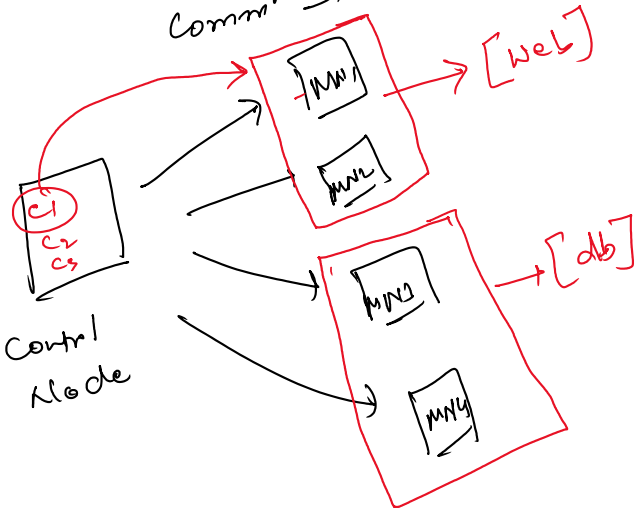
Provisioning, Config. mang., App. Deployment, Orchestration, CD, IaC

chef, puppet, Salt Stack
↓ ↓ ↓
Ruby Ruby Yaml + Python.
(Agent) (Agent) (Agent / Agentless)

Ruby (Agent) (Pull) (Agent) (Pull) (Agent) (Pull/Push)

Ansible → Control Node → Ansible [Installation & execution. running playbook, manage inventory & Pushing tasks.

→ Managed Node (Host)
↳ Target System (VM, Containers, Server)
No Agent installation required on managed node.
Comm: → SSH, WinRM



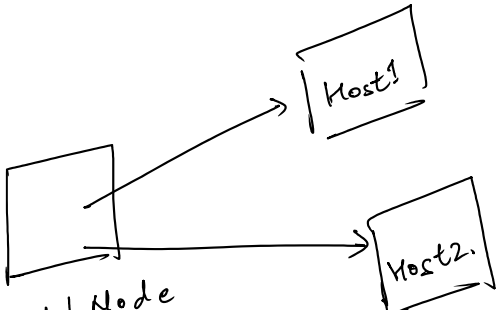
hosts.ini

[web] ← Category.
MN1, MN2 → Host

[db] ← Category.
MN3, MN4

Modules: Unit of work. used in Ansible.

1. Core Module. → Ansible team
2. Community Module. → via Galaxy.
3. Custom module. → User-defined Python Scripts.
4. Action Plugin → Specialized Module. wrapper for Playbook.




```
- name: Play Name
  hosts: group_name_or_ip
  become: true        # Run with sudo
  vars:
    var1: value1
  tasks:
    - name: Task 1 description
      module_name:
        param1: value1
        param2: value2

    - name: Task 2 description
      module_name:
        param: value
```

Hosts.ini

↳ list of host

↳ group on the basis of IP Address.

[Host1]
ip1
[Host2]
ip2

→ ip1
→ ip2

Category of Host

Playbook Name.

Run task with sudo

```
- name: Install and Start NGINX on Web Servers
  hosts: web
  become: true

  tasks:
    - name: Update APT cache
      apt:
        update_cache: yes
    - name: Install NGINX
      apt:
        name: nginx
        state: present
    - name: Ensure NGINX is running
      service:
        name: nginx
        state: started
        enabled: yes
```

Task Description.

Param

Value.

Task Description

Param.

Value.

Task Description.

Module Name.

Value.

Param.

→ install_nginx.yml

→ ansible-playbook -i hosts.ini install_nginx.yml

Command.

↓

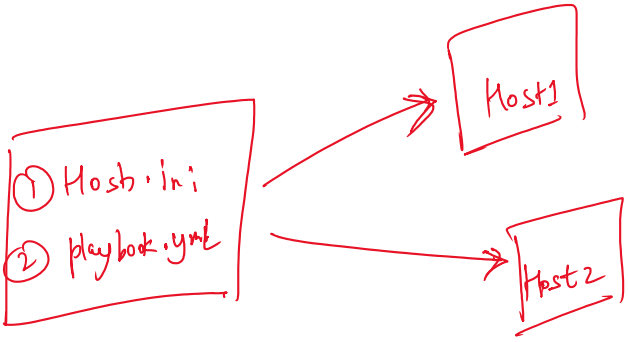
Host details

↓

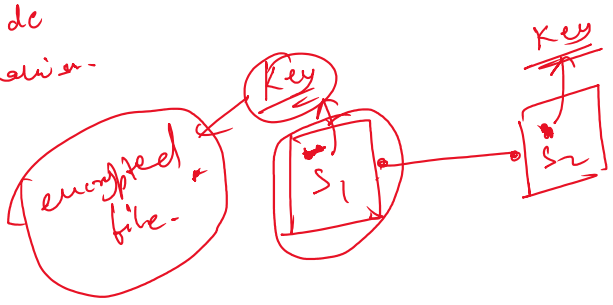
Playbook Name

apt, yum → install package
copy, template → Manage file
Service → Manage Service
user, group → Manage user

file → Set permission ownership
shell, command → shell / command
basic
debug → Print info during execution.



Control Node
1 Push Mechanism.



* Pipeline Governance & Approval in CI CD:—

Pipeline governance → Controls, policies & review mechanism.

1. Complaint with org. rules.
2. Secured & reviewed.
3. Audit & traceable
4. Approved by stakeholders

Why?

1) Security - prevent Unauthorized or Unsafe code reaching production.

(1) Security - prevent from reaching production

(2) Compliance:- Satisfy regulatory req. like HIPAA, PCI

(3) Auditability:- Maintain logs

(4) Risk Management:- Reduce failure | Breaches

(5) Team Accountability:- Enforce ownership & transparency for code changes.

Component:-

(1) RBAC -

(2) Manual Approval | Approval gates -

(3) Policy as Code -

(4) change management Integrations

(5) Audit logs & Signatures like Sigstore, Cosign.

Github Action Gitlab CI/CD Azure Devops	ArgoCD Spinnaker.

Azure DevOps
Jenkins

Q: Organization → GitOps (Pull req) → Infra config.

Question: An organization employs GitOps with Kustomize overlays to manage its infrastructure configurations across development, staging, and production environments. A critical security update needs to be propagated immediately to all environments, but the GitOps pipeline mandates pull request reviews for every environment branch. Which GitOps strategy best accelerates this update while maintaining auditability and preventing configuration drift?

Options:

Most effective

A. Commit the update directly to the production branch first, then backport the change to development and staging branches afterward → instability → Config. drift → very Risky.

B. Use git cherry-pick to selectively apply the update commit across all environment branches promptly (dev, stage or prod.) → efficient.

C. Create a hotfix branch for the update, merge it immediately into production, and then sequentially merge it into other environment branches → Slow,

D. Deploy the update to staging first, perform thorough testing, and then promote the tested artifact to development and production environments → Too slow,

Question Breakdown:-

1. GitOps with Kustomize overlay
2. dev, stage, prod — Separate git branch.
3. Critical Security update → immediately to all the environment.
4. Pull req. (required)

An organization employs Infrastructure as Code (IaC) to manage a complex cloud environment. During a recent audit, discrepancies were found between the desired state defined in the IaC configuration and the actual infrastructure state, indicating drift. To effectively address this issue with minimal manual intervention while ensuring configuration consistency, the team needs to implement a reconciliation strategy that automatically detects and corrects these deviations. What is the most appropriate strategy to resolve this drift?

→ efficient.

Options:

appropriate strategy to resolve this drift?

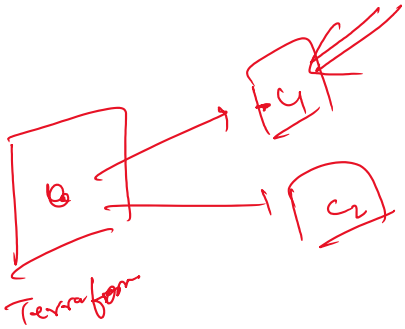
Options:

A. Implement scheduled reconciliation that runs periodically to detect and automatically correct configuration deviations, maintaining alignment efficiently

B. Use imperative scripts triggered manually to identify and correct configuration discrepancies as necessary, relying on human initiation

C. Monitor configuration drift manually and apply all fixes through formal change requests without automation

D. Employ continuous reconciliation that perpetually monitors the infrastructure and instantly corrects deviations to ensure immediate compliance



An organization is setting up a cloud infrastructure using Terraform involving multiple services such as databases and networks. The team aims to improve code reusability for better organization and maintainability. Which approach best enhances code reuse and reduces code duplication, enabling easier control and faster updates?

Options:

A. Use a single, monolithic Terraform configuration file that manages all infrastructure components in one place

B. Duplicate similar code blocks across various Terraform configuration files, which risks inconsistencies during future updates

C. Implement Terraform modules to encapsulate and reuse common infrastructure components effectively across configurations

D. Apply a complex naming convention to resources instead of adopting proper modularization techniques