15 Days of Terraform – Curated by Shaik Hari Sadia Anjum

🚀 Terraform Advanced Series – Day 1: What is IaC & Why Terraform?

Infrastructure as Code(IaC):

Infrastructure as Code (IaC) is a modern DevOps practice where infrastructure — like servers, networks, databases, and more — is provisioned and managed using code, rather than manual processes.

🧱 Before IaC: The Traditional Challenges

Before IaC came into play, infrastructure was managed manually by system administrators. This led to several common issues:

1. **Manual Configuration:**
   Servers and resources were configured manually, which introduced human errors and inconsistencies across environments.
2. **No Version Control:**
   Infrastructure changes were not tracked in version control systems like Git. Rollbacks were difficult and audit trails were missing.
3. **Documentation Dependency:**
   Most configurations were maintained in Word docs, Confluence, or wikis, which often got outdated or missed critical details.
4. **Limited Automation:**
   Teams used basic shell scripts or batch files, which lacked scalability, error handling, and flexibility.
5. **Slow Provisioning:**
   Deploying a new environment involved multiple teams, approvals, and hours of effort — slowing down release cycles.

✅ How IaC Solves This

With IaC, infrastructure is:

- **Defined as code (usually declarative)**
- **Version controlled**
- **Automated**
- **Testable and repeatable**
- **Portable across environments (dev/stage/prod)**

IaC brings speed, consistency, and reliability to modern infrastructure. It also enables full CI/CD pipelines for not just code, but also infrastructure.

Popular IaC tools include:

- **Terraform**
- **AWS CloudFormation**
- **Azure ARM templates**

- Pulumi
- Chef / Puppet / Ansible (Config Mgmt with IaC)

🔧 Why Terraform?

Terraform, developed by HashiCorp, has become the industry standard for IaC. Here's why it's the preferred choice:

🚀 1. Cloud-Agnostic and Multi-Cloud Support

Terraform works with AWS, Azure, GCP, Kubernetes, VMware, and even on-prem systems. You can define infrastructure once and reuse it across providers.

📦 2. Modular and Reusable Code

Terraform promotes modularity. You can create reusable components called modules (like templates) and version them just like application code.

💬 3. Declarative Syntax (HCL)

Terraform uses HCL (HashiCorp Configuration Language), which is clean, readable, and purpose-built for infrastructure. You just define the desired state, and Terraform takes care of the rest.

🧠 4. State Management

Terraform maintains a state file to track your infrastructure. This file helps:

- **Detect drift between what's deployed vs. what's in code**
- **Plan changes intelligently**
- **Support team collaboration**

State can be stored locally or remotely (e.g., in S3 + DynamoDB for locking).

📊 5. Plan Before Apply

Terraform's 2-step workflow (plan → apply) gives full visibility before any changes happen. You can preview additions, deletions, or modifications *before* applying.

🤝 6. Strong Ecosystem

Terraform has:

- **1000+ providers (AWS, Azure, GCP, Datadog, GitHub, etc.)**
- **1000s of open-source community modules**
- **Integrations with Jenkins, Ansible, Kubernetes, and more**

🛠️ 7. Tooling & Extensibility

Terraform supports:

- **Workspaces (for environment isolation)**

- Remote backends (for collaboration)
- Dynamic blocks & loops
- CI/CD pipelines
- Compliance tools (e.g., Sentinel)

👥 8. Huge Community & Open Source

Terraform is open-source, with strong community support, constant improvements, and extensive documentation.

Traditional Infra	With IaC
Manual setup	Automated & consistent
No versioning	Git-tracked changes
Time-consuming	Fast provisioning
Error-prone	Repeatable & tested
Siloed teams	Dev + Ops collaboration