Comprehensive Terraform Training Program Outline

Target Audience:
Developers, Infrastructure & DevOps Engineers moving from manual provisioning to **production-grade Infrastructure as Code (IaC)** using **Terraform**.

Program Focus:
✅ Modular, reusable IaC
✅ Secure local & remote state management
✅ GitOps-driven CI/CD using **Jenkins + GitHub + Bitbucket**
✅ **Policy as Code** using **OPA** and **AWS Policies** for governance

---

LEVEL 1: Terraform Beginner (3 Days)

Project Deliverable:
CloudApp v1.0 — a secure, modular, multi-environment AWS infrastructure deployed via automated CI/CD pipelines and governed through policies.

🟩 Day 1 – Terraform Fundamentals with Local State and Core Workflow

Focus:
Understanding IaC principles, HCL, Terraform's workflow, and building first infra with **local state** (remote state covered on Day 2).

Modules

Modue	Topic	Key Concepts
M1	**Introduction to IaC**	What is IaC, its importance, and Terraform's place in the ecosystem. Declarative vs. imperative approaches. Terraform architecture overview.
M2	**Terraform Basics and HCL**	HCL syntax (blocks, arguments, expressions), Providers, Resources, and Data Sources. Navigating Terraform Registry.
M3	**Core Workflow and Lifecycle**	Understanding Terraform workflow: `init`, `fmt`, `validate`, `plan`, `apply`, `destroy`. Execution plan and resource lifecycle explained.
M4	**Local State Management**	Understanding local state files. Importance of state tracking, state locking, and potential issues in team collaboration (preparing for remote backend).

Labs – Day 1

Lab	Title	Detailed Description	Goal
Lab 1	**First Local Deployment**	Write a Terraform config to deploy an EC2 instance with local backend. Use `init`, `plan`, `apply` commands.	Build first IaC and understand local execution.
Lab 2	**Understanding State Behavior**	Modify infra manually on AWS console and run `terraform plan` to detect drift.	Learn drift detection and Terraform reconciliation.
Lab 3	**Terraform Workflow Drill**	Add multiple resources (e.g., SG + EC2 + IAM role), execute end-to-end workflow and validate outputs.	Practice Terraform's core workflow and dependency resolution.

🟨 Day 2 – Variables, Remote State & CI/CD Pipeline with Branching Strategy

Focus:
Implementing parameterization, remote state (S3 + DynamoDB), and full **GitOps-style CI/CD pipeline** using **Jenkins + GitHub/Bitbucket** with branching and PR-based automation.

Modules

Module	Topic	Key Concepts
M1	**Variables and Outputs**	Variable declaration, types, validation, and sensitive flags. Output values and data sharing between modules. Using `.tfvars` files.
M2	**Remote State Management**	Configuring AWS S3 as backend and DynamoDB for state locking. Migrating from local to remote state. Team collaboration and state isolation per environment.
M3	**Source Code Management (GitHub/Bitbucket)**	Git fundamentals — branch creation (`feature`, `develop`, `main`), pull requests (PR), merge requests (MR), and tagging. Managing Terraform versions using Git.
M4	**CI/CD Pipeline Fundamentals (Jenkins + GitOps)**	Jenkinsfile for IaC pipelines: automate `init`, `validate`, `plan`. PR-based triggers. Branch-based behavior (plan on PR, apply on main). Role-based approvals. Secure credential injection via Jenkins secrets.

Branching and CI/CD Workflow Overview

Branch	Purpose	Automation Behavior
feature/	Developer feature branches	Auto-run Terraform **validate + plan** on each commit (no apply).
develop	Integration branch	Auto-run **plan** and manual approval for **apply** (staging environment).
main	Production branch	Manual merge only after approvals. Jenkins performs **plan + apply** for production environment.

hotfix/	Emergency fix branch	Direct plan/apply workflow with minimal approval (for break-fix scenarios).

Labs – Day 2

Lab	Title	Detailed Description	Goal
Lab 4	**Migrate Local to Remote State**	Move Day 1's local state to S3 backend. Configure DynamoDB for locking and test with two users.	Secure team-ready remote state management.
Lab 5	**Parameterize Infrastructure**	Convert hardcoded values to variables (`ami`, `instance_type`, `region`, etc.). Use `terraform.tfvars` for environment overrides.	Achieve configurable and reusable IaC.
Lab 6	**Setup SCM with Branching Workflow**	Push project to Bitbucket/GitHub. Create feature branch → commit → open Pull Request → merge into develop branch.	Learn real-world Git branching and PR process.
Lab 7	**Integrate Terraform with Jenkins Pipeline**	Configure Jenkins pipeline for IaC. Automate `init`, `validate`, `plan` on PR. Post plan results as PR comments.	Build automated CI/CD for Terraform plans.
Lab 8	**Controlled Apply and Merge Workflow**	Add manual approval stage in Jenkins. Configure "plan on PR" (develop) and "apply on merge" (main).	Enforce controlled deployments with approval gates.

🟦 Day 3 – Modularity, Multi-Environment, and Governance (OPA + AWS Policies)

Focus:
Reusable modular structure, managing multiple environments using workspaces, and introducing enterprise governance using **OPA** and **AWS native policies**.

Modules

Module	Topic	Key Concepts

M1	**Modules and Reusability**	Creating local modules (networking, compute, IAM). Using registry modules. Module versioning and best practices.
M2	**Multi-Environment Deployment**	Using workspaces for `dev`, `stage`, `prod`. Workspace-specific variable sets and backend segregation.
M3	**Advanced HCL and Dynamic Resources**	Using `locals`, `count`, and `for_each`. Generating dynamic resources and leveraging data sources.
M4	**Governance and Policy as Code (OPA + AWS Policies)**	Understanding Policy as Code. Writing OPA (Rego) policies to restrict non-compliant configurations. Enforcing policies in Jenkins pipelines. Applying AWS IAM, Config, and SCPs for security governance.

Labs – Day 3

Lab	Title	Detailed Description	Goal
Lab 9	**Refactor to Modules**	Break the Terraform configuration into reusable modules: VPC, EC2, and Security Groups. Reference them in main configuration.	Build modular, maintainable infrastructure.
Lab 10	**Multi-Environment Deployment using Workspaces**	Create workspaces for dev, staging, and production. Deploy to each environment using backend and variable segregation.	Manage multi-env deployments via workspaces.
Lab 11	**OPA Policy Enforcement**	Write and apply OPA Rego policies (e.g., no public S3 buckets, mandatory tags). Integrate OPA check into Jenkins pipeline.	Enforce compliance via Policy as Code.
Lab 12	**AWS Resource Security Governance**	Apply IAM and AWS Config policies to ensure encryption and tagging standards. Integrate compliance validation in pipeline.	Combine Terraform + AWS policies for end-to-end governance.

✅ Final Project – CloudApp v1.0 (Hands-On Assessment)

Scenario:
Deploy a **multi-environment AWS infrastructure** using Terraform with:

- Modular code structure
- Remote state backend (S3 + DynamoDB)
- Full CI/CD pipeline (Jenkins + GitHub/Bitbucket)
- Branching workflow (`feature → develop → main`)
- OPA and AWS Policies for compliance

Expected Deliverables:

- Reusable Terraform module repository
- Jenkins pipeline for PR validation and controlled apply
- OPA + AWS compliance enforcement logs
- Deployment documentation