LEVEL 3: TERRAFORM ADVANCED (1 DAY)

Duration: 1 Day (8 Hours)
Prerequisites:

- Completion of Level 2 or equivalent Terraform expertise.
- Strong working knowledge of Git/GitHub/Bitbucket and Jenkins CI/CD.

Target Audience:
Senior DevOps Engineers, Cloud Architects, and Technical Leads responsible for defining **secure CI/CD standards**, **policy enforcement**, and **production-grade infrastructure automation**.

Project Deliverable:
CloudApp v3.0 – A **fully production-ready AWS deployment**, integrated with **Jenkins CI/CD pipelines**, **Bitbucket/GitHub branch-based workflows**, **security scanning**, **policy enforcement**, and **secrets management**.

Day 1: Automation, Security, and Production Operations

Focus:
Integrate Terraform with enterprise CI/CD, security validation, and policy enforcement mechanisms to ensure **governance, compliance, and zero-drift operations** for production environments.

Module 1: Deep CI/CD Integration (Jenkins + Bitbucket/GitHub)

Key Concepts:

- Designing **end-to-end production pipelines**:
  - Multi-branch flow: `feature → staging → production`

    - PR-based **automated validation and manual approval** gates.

- Terraform version pinning and module version locking.

- Managing **pipeline authentication**, **remote backend state locking**, and **state recovery**.

- Handling **pipeline failure diagnostics**:

    - Authentication issues (AWS credentials, backend access).

    - State lock conflicts and recovery using `terraform force-unlock`.

Outcome:
Build resilient Jenkins pipelines that can handle real-world production deployment challenges automatically and securely.

Module 2: Static Analysis & Policy-as-Code (PaC)

Key Concepts:

- **Automated validation and compliance enforcement** before deployment:

    - **tflint** for style, correctness, and best practice validation.

    - **checkov** for static security scanning (IAM, S3, encryption, etc.).

- **Policy-as-Code (PaC)** implementation overview:

    - Open Policy Agent (OPA) and Sentinel concepts.

    - Enforcing governance rules — e.g., mandatory encryption, tagging, or region restrictions.

- Integrating these checks into Jenkins pipelines and PR validation workflows.

Outcome:
Achieve automated infrastructure compliance and enforce standards at the CI/CD level.

Module 3: Secrets and Sensitive Data Management

Key Concepts:

- Understanding the risks of **hard-coded secrets** in Terraform.
- **AWS Secrets Manager / HashiCorp Vault** integration patterns:
    - Using `data "aws_secretsmanager_secret_version"` or Vault data sources.
    - Injecting secrets into Terraform at runtime via Jenkins credentials.
- **Credential rotation** and secure injection into Terraform pipelines.
- Using Jenkins credentials binding and environment masking.

Outcome:
Deploy infrastructure securely without exposing sensitive data in code, logs, or pipelines.

Module 4: Production Operations & Disaster Recovery

Key Concepts:

- Structuring **large-scale Terraform projects** for production (multi-module, multi-team).
- Using `terraform graph` to visualize and document dependencies.
- **Disaster Recovery (DR)** for Terraform state and configuration:
    - Backups of S3/DynamoDB state files.
    - Recovery and reconstruction procedures for corrupted or lost states.
- Emergency workflows:
    - Using `terraform state rm/mv` for emergency remediation.
    - `terraform refresh` and validation checks.

- Aligning with enterprise change control and STLC checkpoints.

Outcome:
Establish robust operational processes for Terraform-managed production environments.

Hands-On Labs – Day 1 (AWS + Jenkins + Bitbucket/GitHub)

Lab 11: Automated Validation Pipeline (Terraform + tflint + checkov)

Scenario:
Integrate static validation into your Jenkins pipeline to automatically scan Terraform code for compliance and style violations **before** any infrastructure deployment occurs.

Steps:

1. Add a **static analysis stage** in the Jenkins pipeline that runs:
    - `tflint` for linting/style validation.
    - `checkov` for security and compliance scanning.
2. Configure the pipeline to **fail the build or PR check** if:
    - Unencrypted S3 buckets, open security groups, or missing tags are detected.
3. Push your Terraform code to a `feature/security-scan` branch.
4. Validate that Jenkins automatically blocks the merge due to violations.
5. Fix the issues and re-run to ensure a green build before merging to `main`.

Goal:
Learn how to enforce compliance and coding standards automatically using **PaC and static analysis tools** integrated in Jenkins.

Lab 12: Production Deployment & Review (Manual Approval + Visualization)

Scenario:
Deploy CloudApp v3.0 to the production AWS environment using a controlled CI/CD process with **manual approval gates**, **terraform graph visualization**, and **post-deployment validation**.

Steps:

1. Configure Jenkins multibranch pipeline:
   - On PR merge into `staging` → automatic `terraform plan`.
   - On merge into `main` → **manual approval gate** before `terraform apply`.
2. Implement Terraform version pinning (`required_version` and provider constraints).
3. Run `terraform graph | dot -Tpng > infra-graph.png` to generate a dependency map of the AWS environment.
4. Upload the generated graph as an **artifact in Jenkins** or commit to repo under `/docs/`.
5. Simulate a state lock conflict, then recover using `terraform force-unlock`.
6. Execute the final `terraform apply` for production.

Goal:
Master **end-to-end production deployment workflows** with visibility, security, and control — the exact flow followed in real DevOps teams.

✅ **Final Outcome – CloudApp v3.0 (Production-Ready):**
By the end of Level 3, participants will:

- Build secure, automated **Terraform CI/CD pipelines** using Jenkins and Bitbucket/GitHub.
- Integrate **tflint**, **checkov**, and **policy-as-code** for compliance enforcement.

- Manage secrets securely through **Vault/Secrets Manager** and CI/CD bindings.
- Handle **production-grade Terraform state recovery**, approvals, and visualization.