Terraform Infrastructure as Code (IaC) Coding Standards

# 1. Introduction

This document defines the coding standards and best practices for writing, organizing, and maintaining Terraform Infrastructure as Code (IaC) within the project. The goal is to ensure consistency, readability, reusability, and maintainability across all environments.

# 2. File and Directory Structure

• Use a standardized folder structure for Terraform projects:

- modules/ → reusable Terraform modules

- environments/ → environment-specific configurations (dev, test, preprod, prod)

- main.tf → main entry point

- variables.tf → input variables

- outputs.tf → output values

- providers.tf → provider configuration

- versions.tf → required provider and Terraform versions

# 3. Naming Conventions

• Use lowercase letters, numbers, and underscores for resource names.

• Use descriptive names that indicate the purpose of the resource.

• Example: aws\_vpc.main, aws\_subnet.private, aws\_security\_group.app\_sg

# 4. Variables and Outputs

• Define all variables in variables.tf with type and description.

• Use default values sparingly and only when appropriate.

• Clearly document outputs in outputs.tf with meaningful names.

# 5. Providers and Versions

• Lock provider versions in versions.tf to avoid unexpected changes.

• Specify the required Terraform version using the required\_version attribute.

# 6. State Management

• Always use remote state with state locking (e.g., AWS S3 + DynamoDB).

• Do not commit \*.tfstate or \*.tfstate.backup files to version control.

• Use workspaces or separate state files for different environments.

# 7. Code Formatting and Linting

• Use 'terraform fmt' to maintain consistent formatting.

• Use 'terraform validate' to catch errors early.

• Use 'tflint' to enforce coding standards and detect issues.

# 8. Security Best Practices

• Do not hardcode secrets or credentials in Terraform code.

• Use AWS Secrets Manager, SSM Parameter Store, or Vault for sensitive data.

• Restrict IAM roles and policies to the principle of least privilege.

# 9. Reusability and Modularity

• Write reusable Terraform modules for common infrastructure patterns.

• Use versioning for modules to track changes safely.

• Avoid duplicating code across environments; use variables and modules.

# 10. Review and Testing

• All Terraform code changes must go through code review (pull requests).

• Test changes in lower environments (dev/test) before promoting to prod.

• Use automated pipelines (e.g., Azure DevOps, GitHub Actions) for validation, plan, and apply.

# 11. Documentation

• Document all modules with README.md including inputs, outputs, usage, and examples.

• Maintain architecture diagrams where applicable.

• Keep changelogs for significant updates.

# 12. Conclusion

By adhering to these coding standards, the project ensures consistency, maintainability, and security in Terraform Infrastructure as Code deployments.