Infrastructure as Code

Infrastructure as Code

> **Project:** {{PROJECT_NAME}} **Version:** {{VERSION}} **Date:** {{DATE}}
> **Author:** {{AUTHOR}} **Status:** Draft | In Review | Approved **Reviewers:** {{REVIEWERS}}

Document History

Version	Date	Author	Changes
0.1	{{DATE}}	{{AUTHOR}}	Initial draft

---

1. Overview

IaC Tool: {{IAC_TOOL}} **Tool Version:** {{IAC_VERSION}} **Provider:** {{CLOUD_PROVIDER}} **Provider Version:** {{PROVIDER_VERSION}}

Rationale for tool choice:

{{IAC_RATIONALE}}

Core Principles:

- All infrastructure changes go through code (no manual console changes in staging/prod)
- IaC reviewed like application code (PR, review, merge)
- State is the single source of truth
- Modules are versioned and reusable

---

2. Repository Structure

```
{{IaC_REPO}}/
├── modules/                    # Reusable modules
│   ├── networking/             # VPC, subnets, security groups
│   ├── compute/                # EC2, ECS, Lambda
│   ├── database/               # RDS, ElastiCache
│   ├── storage/                # S3, EFS
│   └── monitoring/             # CloudWatch, alerts
├── environments/               # Environment-specific configs
│   ├── dev/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── terraform.tfvars
│   ├── staging/
│   │   ├── main.tf
│   │   ├── variables.tf
│   │   └── terraform.tfvars
│   └── production/
│       ├── main.tf
│       ├── variables.tf
│       └── terraform.tfvars
├── shared/                     # Shared resources (DNS, accounts)
├── scripts/                    # Helper scripts
│   ├── bootstrap.sh            # Initialize state backend
│   └── validate.sh             # Pre-apply validation
├── .terraform-version          # Pin tool version (tfenv)
├── .tflint.hcl                 # Linting config
└── README.md
```

2.1 Module Organization

Module	Purpose	Inputs	Outputs
`modules/networking`	VPC, subnets, routing	region, cidr_block, az_count	vpc_id, subnet_ids, sg_ids
`modules/compute`	ECS cluster, task definitions	cluster_name, instance_type	cluster_arn, task_role_arn
`modules/database`	RDS instance, parameter groups	engine, instance_class	db_endpoint, db_secret_arn
`modules/storage`	S3 buckets with policies	bucket_name, purpose	bucket_arn, bucket_name
`modules/monitoring`	CloudWatch dashboards, alarms	service_name, thresholds	alarm_arns, dashboard_url

2.2 Environment Separation

- Each environment directory is **independently deployable**
- Environments call the same modules with different variable values
- No cross-environment dependencies (except shared DNS zone)
- Production has stricter apply controls (see Section 6)

2.3 Shared Modules

Shared module registry: {{MODULE_REGISTRY}}

Module	Source	Version	Used By
`networking`	`{{REGISTRY}}/networking`	`~> 2.0`	All environments
`database`	`{{REGISTRY}}/database`	`~> 1.5`	Staging, Production
`monitoring`	`{{REGISTRY}}/monitoring`	`~> 1.2`	All environments

---

3. State Management

3.1 Remote State Backend

Backend: {{STATE_BACKEND}}

Environment	State Location	Access
Dev	`{{STATE_BUCKET}}/dev/terraform.tfstate`	DevOps team
Staging	`{{STATE_BUCKET}}/staging/terraform.tfstate`	DevOps team
Production	`{{STATE_BUCKET}}/production/terraform.tfstate`	Senior DevOps + CI only

Bootstrap (first-time setup):

```
bash scripts/bootstrap.sh {{ENVIRONMENT}}
```

3.2 State Locking

Locking Mechanism: {{LOCK_MECHANISM}} **Lock timeout:** {{LOCK_TIMEOUT}}s **Force unlock:** Only by senior DevOps after verifying no active apply

Lock table (if DynamoDB):

- Table: `{{LOCK_TABLE}}`
- Key: `LockID`
- Billing: On-demand

3.3 State File Organization

Splitting strategy: {{SPLIT_STRATEGY}}

State File	Contains	Reason for split
`base/terraform.tfstate`	Networking, IAM	Infrequently changed
`app/terraform.tfstate`	Compute, app services	Frequently changed
`data/terraform.tfstate`	Databases, caches	High risk, separate lifecycle

---

4. Module Design

4.1 Naming Conventions

Resource naming pattern: `{{PROJECT}}-{{ENVIRONMENT}}-{{COMPONENT}}-{{SUFFIX}}`

Resource	Example
VPC	`myapp-prod-vpc`
ECS Cluster	`myapp-prod-cluster`
RDS Instance	`myapp-prod-db-primary`
S3 Bucket	`myapp-prod-assets-{{ACCOUNT_ID}}`
Security Group	`myapp-prod-app-sg`
IAM Role	`myapp-prod-app-task-role`

4.2 Input / Output Variables

Required variable fields:

```
variable "environment" {
  description = "Deployment environment (dev/staging/production)"
  type        = string
```

```
  validation {
    condition     = contains(["dev", "staging", "production"], var.environment)
    error_message = "Environment must be dev, staging, or production."
  }
}
```

Required output fields:

```
output "database_endpoint" {
  description = "The hostname of the database endpoint"
  value       = aws_db_instance.main.endpoint
  sensitive   = false
}
```

4.3 Versioning Strategy

Module versioning: Semantic versioning (`MAJOR.MINOR.PATCH`) **Pin strategy:** `~> MAJOR.MINOR` (allow patch updates, pin minor) **Upgrade policy:** Review and test before upgrading minor/major versions **Changelog:** Every module version bump requires a CHANGELOG entry

---

5. Workflow

5.1 Standard Change Process

```
flowchart LR
    BRANCH[Create branch] --> CODE[Write/modify IaC]
    CODE --> VALIDATE[terraform validate + tflint]
    VALIDATE --> PLAN[terraform plan]
    PLAN --> PR[Open PR with plan output]
    PR --> REVIEW[Peer review]
    REVIEW --> APPROVE[Approval]
    APPROVE --> APPLY[terraform apply in CI]
    APPLY --> VERIFY[Verify resources]
```

Steps:

1. Create feature branch: `infra/{{TICKET}}-description`
2. Make changes, run `terraform validate && terraform fmt`

3. Run `terraform plan` — attach output to PR
4. Open PR for review (at least 1 reviewer required for dev/staging, 2 for production)
5. CI runs `terraform plan` automatically on PR open
6. Merge triggers `terraform apply` in CI (dev/staging)
7. Production apply requires manual trigger after PR merge

5.2 PR-Based Infrastructure Changes

PR Requirements:

- Title: `[IaC] {{ENVIRONMENT}}: description of change`
- Must include `terraform plan` output in PR description or CI artifact
- Must include justification for the change
- Must reference the related application ticket (if applicable)
- Must have passing CI validation (fmt, validate, tflint, plan)

5.3 Automated Drift Detection

Schedule: {{DRIFT_SCHEDULE}} **Tool:** {{DRIFT_TOOL}} **Alert Channel:** {{DRIFT_ALERT_CHANNEL}} **Action on drift:**

1. Investigate cause (manual change, provider issue, external system)
2. Either fix drift (apply IaC) or update IaC to reflect intentional change
3. Never leave drift unresolved for > {{DRIFT_SLA}}

---

6. Security

6.1 Least Privilege for IaC Service Account

Environment	Service Account	Permissions
Dev	`ci-iac-dev@{{PROJECT}}`	Full write within dev resources
Staging	`ci-iac-staging@{{PROJECT}}`	Full write within staging resources
Production	`ci-iac-prod@{{PROJECT}}`	Restricted write, requires MFA session

6.2 Secret Injection (Not in State)

Rule: Never pass passwords, API keys, or secrets as Terraform variables **Pattern:** Reference secrets manager in resource configuration:

```
# WRONG — secret in state
resource "aws_db_instance" "main" {
  password = var.db_password  # This will be in state in plaintext!
}

# RIGHT — secret from Secrets Manager
resource "aws_db_instance" "main" {
  manage_master_user_password = true  # AWS manages the password in Secrets Manager
}
```

6.3 Policy as Code

Tool: {{POLICY_TOOL}}

Policy	Enforcement
No public S3 buckets	Block
All resources must have environment tag	Warn
RDS must be in private subnet	Block
Security groups must not allow `0.0.0.0/0` on sensitive ports	Block
Encryption at rest required for data resources	Block

---

7. Tagging Strategy

Required tags on all resources:

Tag	Value	Purpose
`Project`	`{{PROJECT_NAME}}`	Cost attribution
`Environment`	`dev` / `staging` / `production`	Environment filter
`ManagedBy`	`terraform`	Identifies IaC-managed resources
`Team`	`{{TEAM}}`	Ownership
`CostCenter`	`{{COST_CENTER}}`	Finance attribution

Optional tags:

Tag	Value	Purpose

`Service`	`{{SERVICE_NAME}}`	Service-level grouping
`Ticket`	`{{TICKET_ID}}`	Change tracking
`ExpiresAt`	`{{DATE}}`	Ephemeral resource cleanup

---

8. Cost Management

Budget alerts:

- Dev: Alert at ${{DEV_BUDGET}} / month
- Staging: Alert at ${{STG_BUDGET}} / month
- Production: Alert at ${{PROD_BUDGET}} / month

Cost optimization built into IaC:

- Dev/staging auto-shutdown: {{AUTO_SHUTDOWN_SCHEDULE}}
- Right-sizing: Instance types reviewed quarterly
- Reserved instances / savings plans: Applied to production

---

9. Disaster Recovery for IaC State

State backup: {{STATE_BACKUP}} **Recovery procedure:**

1. Restore from most recent backup
2. Run `terraform plan` — verify no unexpected changes
3. If state is unrecoverable: `terraform import` for each managed resource (refer to resource inventory)

Prevention:

- S3 versioning enabled on state bucket
- MFA delete required for state bucket
- State bucket access logged to CloudTrail

Related Documents

- Deployment Architecture
- Environment Configuration

- CI/CD Pipeline
- Disaster Recovery Plan

Approval

Role	Name	Date	Signature
Author			
Reviewer			
Approver			

Revision #7
Created 2026-02-23 12:05:54 UTC by John
Updated 2026-05-25 07:33:51 UTC by John