Terraform Cheatsheet

Terraform Cheatsheet

Installation

```
# Install Terraform
wget -O- https://apt.releases.hashicorp.com/gpg | sudo gpg --dearmor -o /usr/share/keyrings/h
echo "deb [signed-by=/usr/share/keyrings/hashicorp-archive-keyring.gpg] https://apt.releases.
sudo apt update && sudo apt install terraform

# Verify installation
terraform --version
terraform -install-autocomplete

# Install terraform-docs (optional)
brew install terraform-docs  # macOS
go install github.com/terraform-docs/terraform-docs@latest  # Go
```

Basic Commands

```
# Initialize working directory
terraform init

# Validate configuration
terraform validate

# Format code
terraform fmt
terraform fmt -check

# Plan changes
terraform plan
terraform plan -out=tfplan

# Apply changes
terraform apply
terraform apply -auto-approve
terraform apply tfplan

# Destroy resources
terraform destroy
```

```
# State inspection
terraform show
terraform state list
terraform state show aws_instance.example

# Output values
terraform output
terraform output instance_ip

# Import existing resources
terraform import aws_instance.example i-0123456789abcdef0
```

Configuration Structure

```
# provider.tf
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
  required_version = ">= 1.0"
}

provider "aws" {
  region = "us-west-2"
}

# variables.tf
variable "instance_type" {
  description = "EC2 instance type"
  type        = string
  default     = "t3.micro"
}

variable "tags" {
  description = "Resource tags"
  type        = map(string)
  default = {
    Name        = "example"
    Environment = "dev"
  }
}

# main.tf
resource "aws_instance" "example" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = var.instance_type

  tags = merge(var.tags, {
    Name = "terraform-example"
  })
}

# outputs.tf
output "instance_ip" {
```

```
  description = "Public IP of the instance"
  value       = aws_instance.example.public_ip
}
```

Resources

```
# AWS EC2 Instance
resource "aws_instance" "web" {
  ami                    = data.aws_ami.ubuntu.id
  instance_type          = "t3.micro"
  key_name               = var.ssh_key
  vpc_security_group_ids = [aws_security_group.web.id]

  root_block_device {
    volume_type           = "gp3"
    volume_size           = 20
    delete_on_termination = true
  }

  user_data = filebase64("${path.module}/user-data.sh")
  tags      = var.tags
}

# AWS S3 Bucket
resource "aws_s3_bucket" "data" {
  bucket = "my-unique-bucket-name-${random_id.bucket_suffix.hex}"

  tags = var.tags
}

resource "aws_s3_bucket_versioning" "data" {
  bucket = aws_s3_bucket.data.id
  versioning_configuration {
    status = "Enabled"
  }
}

# AWS RDS Database
resource "aws_db_instance" "postgres" {
  identifier          = "my-postgres"
  engine              = " postgres"
  engine_version      = "15.4"
  instance_class      = "db.t3.micro"
  allocated_storage   = 20
  storage_type        = "gp2"
  db_name             = "appdb"
  username            = var.db_username
  password            = var.db_password
  db_subnet_group_name = aws_db_subnet_group.main.name
  vpc_security_group_ids = [aws_security_group.db.id]

  skip_final_snapshot = true
}
```

Data Sources

```
# AWS AMI data source
data "aws_ami" "ubuntu" {
  most_recent = true
  owners      = ["099720109477"]  # Canonical

  filter {
    name   = "name"
    values = ["ubuntu/images/hvm-ssd/ubuntu-jammy-22.04-amd64-server-*"]
  }

  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
}

# AWS availability zones
data "aws_availability_zones" "available" {
  state = "available"
}

# AWS current region
data "aws_region" "current" {}

# Use in resources
resource "aws_subnet" "example" {
  vpc_id            = aws_vpc.main.id
  cidr_block        = cidrsubnet(aws_vpc.main.cidr_block, 4, 0)
  availability_zone = data.aws_availability_zones.available.names[0]
}
```

Variables

```
# Define variable
variable "instance_count" {
  description = "Number of instances to create"
  type        = number
  default     = 1

  validation {
    condition     = var.instance_count > 0 && var.instance_count <= 10
    error_message = "Instance count must be between 1 and 10."
  }
}

variable "availability_zones" {
  type = list(string)
}

variable "instance_types" {
  type = map(string)
}

variable "app_config" {
  type = object({
    name     = string
    port     = number
```

```
    replicas = number
  })
}

# Use variable in code
resource "aws_instance" "example" {
  count         = var.instance_count
  instance_type = var.instance_types["web"]

  tags = {
    Name = "${var.app_config.name}-${count.index}"
  }
}
```

Terraform State

```
# Lock state file
terraform providers lock -platform=linux_amd64

# Move resource to new address
terraform state mv 'aws_instance.example' 'aws_instance.example_new'

# Remove resource from state (keep cloud resource)
terraform state rm aws_instance.example

# Import resource
terraform import aws_instance.example i-0123456789abcdef0

# State backup
terraform state pull > backup.tfstate

# State drift detection
terraform plan -refresh-only

# Workspaces
terraform workspace new dev
terraform workspace list
terraform workspace select prod

# Remote state (S3)
terraform {
  backend "s3" {
    bucket         = "my-terraform-state"
    key            = "prod/terraform.tfstate"
    region         = "us-west-2"
    encrypt        = true
    dynamodb_table = "terraform-locks"
  }
}
```

Provisioners

```
resource "aws_instance" "web" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"
```

```
  # Remote-exec provisioner
  provisioner "remote-exec" {
    inline = [
      "sudo apt-get update",
      "sudo apt-get install -y nginx"
    ]

    connection {
      type        = "ssh"
      user        = "ubuntu"
      private_key = file(var.ssh_key)
      host        = self.public_ip
    }
  }

  # File provisioner
  provisioner "file" {
    source      = "scripts/deploy.sh"
    destination = "/tmp/deploy.sh"

    connection {
      type        = "ssh"
      user        = "ubuntu"
      private_key = file(var.ssh_key)
      host        = self.public_ip
    }
  }

  # Local-exec provisioner
  provisioner "local-exec" {
    command = "echo ${self.public_ip} >> hosts.txt"
    when    = create
  }
}
```

Modules

```
# Use module
module "vpc" {
  source = "terraform-aws-modules/vpc/aws"
  version = "5.1.2"

  name = "my-vpc"
  cidr = "10.0.0.0/16"

  azs             = ["us-west-2a", "us-west-2b"]
  private_subnets = ["10.0.1.0/24", "10.0.2.0/24"]
  public_subnets  = ["10.0.101.0/24", "10.0.102.0/24"]

  enable_nat_gateway   = true
  single_nat_gateway   = true
  enable_dns_hostnames = true
}

# Module outputs
output "vpc_id" {
  description = "ID of VPC"
  value       = module.vpc.vpc_id
```

```
}

# Local module
module "webserver" {
  source = "./modules/webserver"

  instance_type = var.instance_type
  subnet_id     = module.vpc.public_subnets[0]
}
```

Loops & Conditionals

```
# count index
resource "aws_instance" "app" {
  count         = 3
  ami           = data.aws_ami.ubuntu.id
  instance_type = "t3.micro"

  tags = {
    Name = "app-${count.index}"
  }
}

# for_each (map)
resource "aws_instance" "servers" {
  for_each = {
    "web" = "t3.micro"
    "app" = "t3.small"
    "db"  = "t3.medium"
  }

  ami           = data.aws.ami.ubuntu.id
  instance_type = each.value

  tags = {
    Name = "server-${each.key}"
  }
}

# dynamic blocks
resource "aws_security_group" "web" {
  name        = "web-sg"
  description = "Allow web traffic"

  dynamic "ingress" {
    for_each = var.allowed_ports
    content {
      from_port   = ingress.value
      to_port     = ingress.value
      protocol    = "tcp"
      cidr_blocks = ["0.0.0.0/0"]
    }
  }
}
```

Functions

```
# string functions
resource "aws_s3_bucket" "bucket" {
  bucket = "${var.project_name}-${lower(var.environment)}-${random_id.unique.hex}"
}

# numeric functions
fixed_size = var.mongo_size > 20 ? var.mongo_size : 20

# collection functions
subnet_ids = [for subnet in aws_subnet.public : subnet.id]
instance_ips = values(aws_instance.app[*].public_ip)

# file functions
user_data = templatefile("${path.module}/user-data.tpl", {
  message = "Hello World"
})
```

Templates

```
# templatefile function
user_data = templatefile("${path.module}/cloud-init.tpl", {
  hostname     = var.hostname
  ssh_key_name = var.ssh_key
  region       = data.aws_region.current.name
})

# cloud-init.tpl
#!/bin/bash
hostnamectl set-hostname ${hostname}
apt-get update
apt-get install -y nginx
systemctl enable nginx
systemctl start nginx
```

Terraform 1.5+ Features

```
# Import blocks (Terraform 1.5+) — declarative resource imports
# No more manual terraform import commands
import {
  to = aws_instance.existing_web
  id = "i-0123456789abcdef0"
}

import {
  to = aws_security_group.existing_sg
  id = "sg-0abc123def456"
}

# Moved blocks — rename or move resources across modules
moved {
  from = aws_instance.old_name
  to   = aws_instance.new_name
}

moved {
```

```
  from = module.old_vpc.aws_subnet.public[0]
  to   = aws_subnet.public
}

# Check blocks (Terraform 1.5+) — pre and post conditions
resource "aws_instance" "web" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = var.instance_type

  # Precondition: ensure instance type is not too large
  lifecycle {
    precondition {
      condition     = var.instance_type != "p4d.24xlarge"
      error_message = "Instance type p4d.24xlarge is not allowed."
    }
  }
}

# Postcondition on output
output "web_public_ip" {
  value = aws_instance.web.public_ip
  postcondition {
    condition     = self != ""
    error_message = "Public IP must not be empty."
  }
}
```

State Management Deep Dive

```
# State locking mechanisms
# S3 + DynamoDB (most common for AWS)
# Azure Blob Storage with lease locking
# GCS with built-in locking
# HashiCorp Consul
# Terraform Cloud / Enterprise (native locking)

# Taint a resource (force recreate on next apply)
terraform taint aws_instance.web

# Untaint a resource
terraform untaint aws_instance.web

# Force unlock a stuck state (use with caution)
terraform force-unlock <lock-id>

# Show current state in JSON
terraform show -json > current_state.json

# Show state as a readable plan
terraform show

# Pull raw state
terraform state pull > backup.tfstate

# Push state (dangerous — use only for state migration)
terraform state push updated.tfstate
```

```
# Sensitive values in state
variable "db_password" {
  type      = string
  sensitive = true  # value hidden in plan output and state
}

# Mark output as sensitive
output "db_connection_string" {
  value     = "postgres://user:${var.db_password}@host:5432/db"
  sensitive = true
}
```

Workspaces

```
# Workspace commands
terraform workspace new dev
terraform workspace new staging
terraform workspace new prod
terraform workspace list        # list all workspaces
terraform workspace select prod  # switch workspace

# Use workspace name in configuration
terraform workspace show
```

```
# Workspace-aware configuration
resource "aws_instance" "app" {
  ami           = data.aws_ami.ubuntu.id
  instance_type = terraform.workspace == "prod" ? "m5.large" : "t3.micro"

  tags = {
    Name        = "app-${terraform.workspace}"
    Environment = terraform.workspace
  }
}

# When to use workspaces
# GOOD: Same infrastructure, different parameters (dev/stage/prod)
# BAD: Completely different infrastructure per environment
# Alternative: directory-based isolation (separate Terraform roots per environment)
```

CI/CD Pipelines

```
# GitHub Actions workflow (.github/workflows/terraform.yml)
name: Terraform
on:
  push:
    branches: [main]
  pull_request:
    branches: [main]

jobs:
  terraform:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v4
```

```
      - uses: hashicorp/setup-terraform@v3
        with:
          terraform_version: "1.7.x"

      - name: Terraform Init
        run: terraform init

      - name: Terraform Validate
        run: terraform validate

      - name: Terraform Plan (on PR)
        if: github.event_name == 'pull_request'
        run: terraform plan -no-color -input=false
        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}

      - name: Terraform Apply (on main)
        if: github.ref == 'refs/heads/main' && github.event_name == 'push'
        run: terraform apply -auto-approve -input=false
        env:
          AWS_ACCESS_KEY_ID: ${{ secrets.AWS_ACCESS_KEY_ID }}
          AWS_SECRET_ACCESS_KEY: ${{ secrets.AWS_SECRET_ACCESS_KEY }}
```

```
# GitLab CI example (.gitlab-ci.yml)
stages:
  - validate
  - plan
  - apply

variables:
  TF_ROOT: "${CI_PROJECT_DIR}/terraform"

fmt:
  stage: validate
  image: hashicorp/terraform:1.7
  script:
    - cd $TF_ROOT && terraform fmt -check -recursive

validate:
  stage: validate
  image: hashicorp/terraform:1.7
  script:
    - cd $TF_ROOT && terraform init -backend=false && terraform validate

plan:
  stage: plan
  image: hashicorp/terraform:1.7
  script:
    - cd $TF_ROOT && terraform plan -out=tfplan
  artifacts:
    paths:
      - $TF_ROOT/tfplan

apply:
  stage: apply
  image: hashicorp/terraform:1.7
  when: manual
```

```
  script:
    - cd $TF_ROOT && terraform apply -auto-approve tfplan
```

```
# Security scanning
# tfsec — static analysis for Terraform
tfscan ./terraform
tfscan --soft-fail ./terraform

# checkov — policy-as-code scanner
checkov -d ./terraform --check CKV_AWS_1,CKV_AWS_2

# terraform validate + format check
terraform fmt -check -recursive
terraform validate
```

Module Composition

```
# Module source types
module "vpc" {
  source  = "terraform-aws-modules/vpc/aws"
  version = "5.5.0"  # pin module version
}

module "database" {
  source = "./modules/database"  # local module
}

module "shared" {
  source = "git::https://github.com/org/terraform-modules.git//networking?ref=v2.1.0"
}

# Module with for_each
module "s3_buckets" {
  source   = "./modules/s3-bucket"
  for_each = toset(["logs", "backups", "artifacts"])

  bucket_name = "${var.project_name}-${each.value}"
  environment = var.environment
}
```

```
# Module best practices
# modules/s3-bucket/main.tf
variable "bucket_name" {
  type        = string
  description = "Name of the S3 bucket"
}

variable "environment" {
  type        = string
  description = "Deployment environment"
}

resource "aws_s3_bucket" "this" {
  bucket = "${var.bucket_name}-${var.environment}"

  tags = {
```

```
    Name        = var.bucket_name
    Environment = var.environment
    ManagedBy   = "terraform"
  }
}

output "bucket_arn" {
  value       = aws_s3_bucket.this.arn
  description = "ARN of the created S3 bucket"
}

output "bucket_id" {
  value = aws_s3_bucket.this.id
}
```

Security Best Practices

```
# Mark secrets as sensitive
variable "api_key" {
  type      = string
  sensitive = true
}

# Use data sources for secrets (Vault, AWS Secrets Manager)
data "aws_secretsmanager_secret_version" "db_creds" {
  secret_id = "prod/database/credentials"
}

# Least-privilege IAM
resource "aws_iam_policy" "app_readonly" {
  name        = "app-readonly-policy"
  description = "Read-only access for application"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "s3:GetObject",
          "s3:ListBucket"
        ]
        Resource = [
          aws_s3_bucket.data.arn,
          "${aws_s3_bucket.data.arn}/*"
        ]
      }
    ]
  })
}

# Variable validation
variable "vpc_cidr" {
  type = string
  validation {
    condition     = can(cidrnetmask(var.vpc_cidr))
    error_message = "Must be a valid CIDR block."
  }
```

```
}

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

HCL Expressions & Types

```
# For expressions
variable "usernames" {
  type    = list(string)
  default = ["alice", "bob", "charlie"]
}

locals {
  upper_names = [for name in var.usernames : upper(name)]
  # ["ALICE", "BOB", "CHARLIE"]

  name_lengths = { for name in var.usernames : name => length(name) }
  # { "alice": 5, "bob": 3, "charlie": 7 }

  # For with filter
  long_names = [for name in var.usernames : name if length(name) > 3]
  # ["alice", "charlie"]
}

# Splat expressions
resource "aws_instance" "app" {
  count = 3
  ami   = data.aws_ami.ubuntu.id
}

# Classic splat (only for lists)
output "all_ami_ids" {
  value = aws_instance.app[*].ami
}

# Attribute splat (works with objects)
output "all_ids" {
  value = aws_instance.app[*].id
}

# Conditional expressions
resource "aws_ebs_volume" "data" {
  availability_zone = aws_instance.web.availability_zone
  size              = var.environment == "prod" ? 100 : 20
  type              = var.environment == "prod" ? "io2" : "gp3"
}

# Null coalescing
output "instance_type" {
  value = var.instance_type != null ? var.instance_type : "t3.micro"
}
```

```
# try() — returns first non-error result
output "subnet_id" {
  value = try(aws_subnet.public[0].id, aws_subnet.private[0].id, "default-subnet")
}

# Collection functions
locals {
  flat   = flatten([[1, 2], [3, 4]])  # [1, 2, 3, 4]
  unique = distinct(["a", "b", "a"])  # ["a", "b"]
  compacted = compact([null, "a", null, "b", null])  # ["a", "b"]
  chunked   = chunklist(["a", "b", "c", "d"], 2)  # [["a","b"], ["c","d"]]
  merged    = merge({a = 1}, {b = 2})  # {a = 1, b = 2}
}
```

Data Source Patterns

```
# IAM policy document (avoid inline JSON)
data "aws_iam_policy_document" "s3_readonly" {
  statement {
    sid    = "AllowS3Read"
    effect = "Allow"
    actions = [
      "s3:GetObject",
      "s3:ListBucket"
    ]
    resources = [
      aws_s3_bucket.data.arn,
      "${aws_s3_bucket.data.arn}/*"
    ]
  }
}

resource "aws_iam_policy" "s3_readonly" {
  name   = "s3-readonly-policy"
  policy = data.aws_iam_policy_document.s3_readonly.json
}

# Current caller identity
data "aws_caller_identity" "current" {}

output "account_id" {
  value = data.aws_caller_identity.current.account_id
}

# Template file rendering
data "template_file" "user_data" {
  template = file("${path.module}/user-data.tpl")
  vars = {
    hostname = var.hostname
    port     = var.app_port
  }
}

# External data source (run shell commands)
data "external" "ami_filter" {
  program = ["bash", "${path.module}/scripts/find-ami.sh"]
  query = {
    region = var.region
```

```
  }
}
```

Import & Drift Detection

```
# Legacy import (one at a time)
terraform import aws_instance.web i-0123456789abcdef0

# Import blocks (Terraform 1.5+)
# Define in configuration, then run: terraform plan
import {
  to = aws_s3_bucket.existing
  id = "my-existing-bucket"
}

# Import with provider configuration
import {
  to = aws_instance.web
  id = "i-0123456789abcdef0"
  provider = aws.us_west_2
}

# Drift detection
terraform plan -refresh-only  # detect drift without proposing changes

# Addressing drift
terraform apply              # converge state back to configuration
terraform state rm resource   # if resource was deleted outside Terraform
```

Networking Patterns

```
# VPC with public and private subnets
resource "aws_vpc" "main" {
  cidr_block           = "10.0.0.0/16"
  enable_dns_hostnames = true
  enable_dns_support   = true

  tags = { Name = "${var.project}-vpc" }
}

resource "aws_subnet" "public" {
  count                   = length(var.azs)
  vpc_id                  = aws_vpc.main.id
  cidr_block              = cidrsubnet(aws_vpc.main.cidr_block, 4, count.index)
  availability_zone       = var.azs[count.index]
  map_public_ip_on_launch = true

  tags = { Name = "${var.project}-public-${count.index}" }
}

resource "aws_subnet" "private" {
  count             = length(var.azs)
  vpc_id            = aws_vpc.main.id
  cidr_block        = cidrsubnet(aws_vpc.main.cidr_block, 4, count.index + length(var.azs))
  availability_zone = var.azs[count.index]
```

```
  tags = { Name = "${var.project}-private-${count.index}" }
}

# Security group with dynamic blocks
variable "ingress_rules" {
  type = list(object({
    port     = number
    protocol = string
    cidr     = list(string)
  }))
  default = [
    { port = 80,  protocol = "tcp", cidr = ["0.0.0.0/0"] },
    { port = 443, protocol = "tcp", cidr = ["0.0.0.0/0"] }
  ]
}

resource "aws_security_group" "web" {
  name   = "${var.project}-web-sg"
  vpc_id = aws_vpc.main.id

  dynamic "ingress" {
    for_each = var.ingress_rules
    content {
      from_port   = ingress.value.port
      to_port     = ingress.value.port
      protocol    = ingress.value.protocol
      cidr_blocks = ingress.value.cidr
    }
  }

  egress {
    from_port   = 0
    to_port     = 0
    protocol    = "-1"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

# ALB with target group
resource "aws_lb" "web" {
  name            = "${var.project}-alb"
  internal        = false
  security_groups = [aws_security_group.web.id]
  subnets         = aws_subnet.public[*].id
}

resource "aws_lb_target_group" "web" {
  name     = "${var.project}-tg"
  port     = 80
  protocol = "HTTP"
  vpc_id   = aws_vpc.main.id
}

resource "aws_lb_listener" "http" {
  load_balancer_arn = aws_lb.web.arn
  port              = 80
  protocol          = "HTTP"

  default_action {
```

```
    type             = "forward"
    target_group_arn = aws_lb_target_group.web.arn
  }
}

# Route53 DNS record
resource "aws_route53_record" "app" {
  zone_id = aws_route53_zone.main.zone_id
  name    = "app.${var.domain_name}"
  type    = "A"

  alias {
    name                   = aws_lb.web.dns_name
    zone_id                = aws_lb.web.zone_id
    evaluate_target_health = true
  }
}
```

Testing

```
# Terraform test (1.6+)
# Create test files in tests/ directory

# Run all tests
terraform test

# Run specific test file
terraform test -filter="s3_bucket"

# Verbose output
terraform test -verbose
```

```
# tests/s3_bucket.tftest.hcl
run "bucket_exists" {
  command = plan

  assert {
    condition = aws_s3_bucket.data.bucket != ""
    error_message = "S3 bucket name must not be empty."
  }
}

run "bucket_has_versioning" {
  command = plan

  assert {
    condition = aws_s3_bucket_versioning.data.versioning_configuration[0].status == "Enabled"
    error_message = "S3 bucket versioning must be enabled."
  }
}

# Variables for tests
variables {
  project_name = "test-project"
  environment  = "test"
}
```

Error Handling & Debugging

```
# Log levels for debugging
TF_LOG=TRACE terraform plan    # most verbose
TF_LOG=DEBUG terraform apply   # debug level
TF_LOG=INFO terraform plan     # info level
TF_LOG=WARN terraform apply    # warnings only
TF_LOG_PATH=terraform.log TF_LOG=DEBUG terraform apply  # log to file

# Target specific resources (apply only a subset)
terraform plan -target=aws_instance.web
terraform apply -target=aws_instance.web -target=aws_security_group.web

# Common errors and fixes
# Error: "Module not found" → terraform init (or check source path)
# Error: "Invalid for_each argument" → ensure the map/set has known keys
# Error: "Cycle: ..." → remove circular dependencies between resources
# Error: "Error acquiring the state lock" → terraform force-unlock <lock-id>
# Error: "Provider not installed" → terraform init -upgrade

# Plan diff to file
terraform plan -out=tfplan
terraform show -no-color tfplan > plan_diff.txt
```

Best Practices

1. **Use modules** for reusable infrastructure components
2. **Version pinning** — Pin provider and module versions with `~>` or `=`
3. **State management** — Use remote state (S3 + DynamoDB) with locking
4. **Idempotence** — Ensure resources can be applied multiple times safely
5. **Separation of concerns** — Split Terraform code into logical files (main.tf, variables.tf, outputs.tf)
6. **Validation** — Use variable validation to catch errors early
7. **Documentation** — Add descriptions to variables and outputs
8. **DRY principle** — Use locals and loops to avoid repetition
9. **Testing** — Use `terraform validate`, `terraform plan`, and `terraform test` before apply
10. **Security** — Never store secrets in state; use `sensitive = true` and external secret managers
11. **CI/CD** — Automate plan on PRs, apply on merge to main
12. **Import blocks** — Use declarative imports (Terraform 1.5+) instead of CLI `terraform import`
13. **Run `terraform fmt`** before every commit
14. **Pin Terraform version** with `required_version` in the terraform block

---

Infrastructure as Code Made Easy