

Terraform for Data Engineers

Provision and manage cloud data infrastructure as code — storage, access control, warehouses, compute platforms, and orchestration.

Last reviewed 29 Sep 2026 ·

Prerequisites: [Cloud Storage](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/cloud-storage/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/cloud-storage/) · [Git for DE](https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/git-for-de/) (https://sarangambekar1997.github.io/data-engineering-handbook/00-foundations/git-for-de/)

Related: [Snowflake](https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/01-storage/snowflake-reference/) · [Databricks](https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/databricks-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/02-processing/databricks-reference/) · [Docker](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/docker-reference/) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/docker-reference/) · [Kubernetes](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/kubernetes-for-de/) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/kubernetes-for-de/) · [Testing and CI/CD](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/testing-cicd/) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/testing-cicd/) · [Glossary](https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/) (https://sarangambekar1997.github.io/data-engineering-handbook/99-reference/glossary/)

Overview

Challenge: A data platform consists of extensive infrastructure: storage buckets, identity and access policies, warehouse databases and permissions, compute workspaces and jobs, and orchestration environments. Configuring these manually in web consoles is not repeatable — settings go undocumented, environments drift apart, and recovery after a mistake is slow.

Solution: Terraform lets you *declare* the desired infrastructure in `.tf` files and calculates what to create, change, or delete to match it. The configuration lives in Git, so every change is reviewed in a pull request, and the same code builds development, staging, and production.

```
.tf files (desired state)      terraform plan
terraform apply                ↗
                               ↘
→ create / update / delete    compare with state + real
  "a bucket, a role,          infrastructure; show a
```

```
via each provider's API
  a warehouse, a grant"      diff for review
(ccloud, database, SaaS)

|

state file records what exists
```

State management: Terraform maintains a state file that maps configuration to real resource IDs. It must be protected with a remote backend, locking, and versioning — a lost or corrupted state file leaves Terraform unaware of what it manages.



On this page

Basic - [Why Infrastructure as Code](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#why-infrastructure-as-code) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#why-infrastructure-as-code) - [Terraform Concepts](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#terraform-concepts) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#terraform-concepts) - [Setup & First Resource](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#setup--first-resource) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#setup--first-resource) - [Core CLI Commands](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#core-cli-commands) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#core-cli-commands)

Intermediate - [Variables & Outputs](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#variables--outputs) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#variables--outputs) - [Modules](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#modules) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#modules) - [Remote State](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#remote-state) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#remote-state) - [AWS](#)

[Resources for DE](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#aws-resources-for-de) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#aws-resources-for-de)

Advanced - [Warehouses and Data Platforms](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#warehouses-and-data-platforms) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#warehouses-and-data-platforms) - [Snowflake with Terraform](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#snowflake-with-terraform) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#snowflake-with-terraform) - [Databricks with Terraform](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#databricks-with-terraform) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#databricks-with-terraform) - [Airflow Infra on AWS](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#airflow-infra-on-aws) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#airflow-infra-on-aws) - [Common Pitfalls](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#common-pitfalls) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#common-pitfalls)

Reference - Cheat Sheet (<https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#cheat-sheet>) - **Interview Questions** (<https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#interview-questions>) - **Further Reading** (<https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/terraform-for-de/#further-reading>)

Why Infrastructure as Code

Without IaC:

Engineer A creates an S3 bucket manually via the AWS console

Engineer B creates a similar bucket differently – different naming, no encryption

3 months later: nobody knows what exists, what settings were used, or who owns what

Deleting the wrong bucket means data loss

With IaC (Terraform):

All infrastructure is defined in .tf files in the repo

Changes go through PR review (same as code)

`terraform apply` brings any environment to the defined state

Destroyed and recreated resources are identical every time

State file tracks what exists – safe to destroy, safe to recreate

Benefits for data teams specifically: - Reproduce dev/staging/prod environments from the same code - Manage warehouse databases, roles, and permissions without manual console work - Provision compute workspaces, clusters, jobs, and secrets programmatically - IAM policies as code — auditable, reviewable, version-controlled

Terraform Concepts

Concept	Description
Provider	Plugin that talks to an external API (cloud providers, databases, SaaS platforms)
Resource	An infrastructure object to create (storage bucket, <u>IAM</u> role, warehouse database)

Concept	Description
Data source	Read existing infrastructure without managing it
Variable	Input parameter (like a function argument)
Output	Exported value (like a return value)
Module	Reusable group of resources (like a function)
State	File tracking what Terraform has created — never edit manually
Plan	Preview of what <code>terraform apply</code> will create/update/destroy
Backend	Where the state file lives (S3, Terraform Cloud, local)

Workflow:

Write .tf files → `terraform init` → `terraform plan` →
`terraform apply`

↑ review diff ↑

↑ confirm

Setup & First Resource

```
# Install Terraform (macOS)
brew install terraform

# Install Terraform (Linux)
wget
https://releases.hashicorp.com/terraform/1.7.0/terraform_1.
7.0_linux_amd64.zip
unzip terraform_1.7.0_linux_amd64.zip
mv terraform /usr/local/bin/

# Verify
terraform version
```

Project structure

```
infra/
├─ main.tf          # core resources
├─ variables.tf     # input variables
├─ outputs.tf       # output values
├─ providers.tf     # provider configuration
├─ terraform.tfvars # variable values (DO NOT COMMIT secrets)
└─ modules/
    ├─ s3-data-lake/
    └─ warehouse-env/
```

First resource — S3 bucket

```
# providers.tf
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 5.0"
    }
  }
  required_version = ">= 1.5.0"
}

provider "aws" {
  region = "us-east-1"
}

# main.tf
resource "aws_s3_bucket" "data_lake" {
  bucket = "mycompany-data-lake-prod"

  tags = {
    Environment = "prod"
    Team        = "data-engineering"
    ManagedBy   = "terraform"
  }
}

resource "aws_s3_bucket_versioning" "data_lake" {
  bucket = aws_s3_bucket.data_lake.id
```

```

    versioning_configuration {
      status = "Enabled"
    }
  }

  resource
  "aws_s3_bucket_server_side_encryption_configuration"
  "data_lake" {
    bucket = aws_s3_bucket.data_lake.id
    rule {
      apply_server_side_encryption_by_default {
        sse_algorithm = "AES256"
      }
    }
  }
}

resource "aws_s3_bucket_public_access_block" "data_lake" {
  bucket                = aws_s3_bucket.data_lake.id
  block_public_acls      = true
  block_public_policy    = true
  ignore_public_acls     = true
  restrict_public_buckets = true
}

```

Core CLI Commands

```

# Initialize – download providers, set up backend
terraform init

# Preview changes – never skips this step
terraform plan

# Apply changes (prompts for confirmation)
terraform apply

# Apply without prompt (use in CI only)
terraform apply -auto-approve

# Destroy all resources managed by this config (DANGEROUS)
terraform destroy

```

```
# Format all .tf files
terraform fmt

# Validate syntax and configuration
terraform validate

# Show current state
terraform show

# List all resources in state
terraform state list

# Import an existing resource into state
terraform import aws_s3_bucket.data_lake mycompany-data-lake-prod

# Target a specific resource (useful for debugging)
terraform apply -target=aws_s3_bucket.data_lake

# View outputs
terraform output
terraform output bucket_arn    # specific output
```

Variables & Outputs

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

variable "aws_region" {
  description = "AWS region for all resources"
```

```

    type          = string
    default       = "us-east-1"
}

variable "bucket_name" {
    description = "S3 bucket name"
    type        = string
}

variable "tags" {
    description = "Common tags applied to all resources"
    type        = map(string)
    default     = {}
}

variable "warehouse_account" {
    description = "Data warehouse account identifier"
    type        = string
    sensitive   = true    # redacted in logs and state display
}

```

```

# terraform.tfvars (DO NOT commit to git – add to
.gitignore)
environment      = "prod"
bucket_name      = "mycompany-data-lake-prod"
warehouse_account = "myorg-analytics"

```

```

# Reference variables in resources
resource "aws_s3_bucket" "data_lake" {
    bucket = var.bucket_name
    tags   = merge(var.tags, { Environment = var.environment })
}

# outputs.tf
output "bucket_arn" {
    description = "ARN of the data lake S3 bucket"
    value       = aws_s3_bucket.data_lake.arn
}

output "bucket_name" {

```

```
    value = aws_s3_bucket.data_lake.bucket
}
```

Modules

Reusable groups of resources — like functions for infrastructure.

```
# modules/s3-data-lake/main.tf
variable "bucket_name" { type = string }
variable "environment" { type = string }
variable "layers" {
  type    = list(string)
  default = ["bronze", "silver", "gold"]
}

resource "aws_s3_bucket" "this" {
  bucket = var.bucket_name
  tags   = { Environment = var.environment, ManagedBy =
"terraform" }
}

# Create folder structure via lifecycle marker objects
resource "aws_s3_object" "layer_folders" {
  for_each = toset(var.layers)
  bucket   = aws_s3_bucket.this.id
  key      = "${each.value}/"
  content  = ""
}

resource
"aws_s3_bucket_server_side_encryption_configuration" "this"
{
  bucket = aws_s3_bucket.this.id
  rule {
    apply_server_side_encryption_by_default { sse_algorithm
= "AES256" }
  }
}
```

```
output "bucket_arn" { value = aws_s3_bucket.this.arn }
output "bucket_name" { value = aws_s3_bucket.this.bucket }
```

```
# main.tf – using the module
module "data_lake_prod" {
  source      = "./modules/s3-data-lake"
  bucket_name = "mycompany-data-lake-prod"
  environment = "prod"
  layers      = ["bronze", "silver", "gold", "archive"]
}

module "data_lake_dev" {
  source      = "./modules/s3-data-lake"
  bucket_name = "mycompany-data-lake-dev"
  environment = "dev"
}

# Reference module outputs
output "prod_bucket_arn" {
  value = module.data_lake_prod.bucket_arn
}
```

Remote State

State must live somewhere shared — not on a local laptop.

```
# backend.tf – store state in S3 with native S3 locking
(Terraform 1.11+)
terraform {
  backend "s3" {
    bucket      = "mycompany-terraform-state"
    key         = "data-platform/prod/terraform.tfstate"
    region      = "us-east-1"
    encrypt     = true
    use_lockfile = true      # lock via a .tflock object in
S3 – prevents concurrent applies
    # Older Terraform: dynamodb_table = "terraform-state-
lock" (now deprecated)
```

```
}  
}
```

```
# Create the state bucket BEFORE init (once, manually or  
with a bootstrap script)  
aws s3api create-bucket --bucket mycompany-terraform-state  
--region us-east-1  
aws s3api put-bucket-versioning --bucket mycompany-  
terraform-state \  
    --versioning-configuration Status=Enabled    # lets you  
recover a corrupted state file
```

AWS Resources for DE

S3 data lake with IAM

```
# Full data lake setup: bucket + IAM role for pipeline  
access  
  
resource "aws_s3_bucket" "data_lake" {  
    bucket = "mycompany-data-lake-${var.environment}"  
}  
  
# Lifecycle policy: move old data to cheaper storage  
resource "aws_s3_bucket_lifecycle_configuration"  
"data_lake" {  
    bucket = aws_s3_bucket.data_lake.id  
  
    rule {  
        id      = "bronze-lifecycle"  
        status = "Enabled"  
        filter { prefix = "bronze/" }  
  
        transition {  
            days      = 30  
            storage_class = "STANDARD_IA"  
        }  
        transition {  
            days      = 90
```

```

        storage_class = "GLACIER"
    }
    expiration { days = 365 }
}
}

# IAM role for the data pipeline (Airflow, Glue, etc.)
resource "aws_iam_role" "data_pipeline" {
    name = "data-pipeline-role-${var.environment}"
    assume_role_policy = jsonencode({
        Version = "2012-10-17"
        Statement = [{
            Effect      = "Allow"
            Principal   = { Service = "ec2.amazonaws.com" }
            Action       = "sts:AssumeRole"
        }]
    })
}

resource "aws_iam_policy" "data_lake_access" {
    name = "data-lake-access-${var.environment}"
    policy = jsonencode({
        Version = "2012-10-17"
        Statement = [
            {
                Effect = "Allow"
                Action = ["s3:GetObject", "s3:PutObject",
"s3:DeleteObject", "s3:ListBucket"]
                Resource = [
                    aws_s3_bucket.data_lake.arn,
                    "${aws_s3_bucket.data_lake.arn}/*"
                ]
            }
        ]
    })
}

resource "aws_iam_role_policy_attachment" "data_pipeline" {
    role          = aws_iam_role.data_pipeline.name
    policy_arn    = aws_iam_policy.data_lake_access.arn
}

```

Secrets Manager

```
# Store pipeline credentials in AWS Secrets Manager
resource "aws_secretsmanager_secret" "warehouse_creds" {
  name      = "/data-
platform/${var.environment}/warehouse"
  description = "Warehouse credentials for the data
pipeline"
}

resource "aws_secretsmanager_secret_version"
"warehouse_creds" {
  secret_id = aws_secretsmanager_secret.warehouse_creds.id
  secret_string = jsonencode({
    account = var.warehouse_account
    username = var.warehouse_username
    password = var.warehouse_password    # pass via env
var, never hardcode
  })
}

output "warehouse_secret_arn" {
  value = aws_secretsmanager_secret.warehouse_creds.arn
}
```

Warehouses and Data Platforms

Every major warehouse and data platform has a Terraform provider, and the pattern is the same: declare databases or datasets, compute, roles, and grants, and manage them through pull requests. Two short examples on the cloud providers' own services come first; Snowflake and Databricks follow in more depth.

```
# BigQuery (google provider): dataset + read access for a group
resource "google_bigquery_dataset" "analytics" {
  dataset_id = "analytics_${var.environment}"
  location   = "US"
}

resource "google_bigquery_dataset_iam_member"
```

```

"analysts_read" {
  dataset_id = google_bigquery_dataset.analytics.dataset_id
  role       = "roles/bigquery.dataViewer"
  member     = "group:data-team@example.com"
}

# Amazon Redshift Serverless (aws provider): namespace +
workgroup
resource "aws_redshiftserverless_namespace" "analytics" {
  namespace_name = "analytics-${var.environment}"
  db_name        = "analytics"
}

resource "aws_redshiftserverless_workgroup" "analytics" {
  namespace_name =
aws_redshiftserverless_namespace.analytics.namespace_name
  workgroup_name = "analytics-${var.environment}"
  base_capacity  = 8           # RPU; scales up
                             automatically under load
}

```

Snowflake with Terraform

Uses the `snowflakedb/snowflake` provider **v1+** (formerly `Snowflake-Labs/snowflake`). The 1.0 release renamed several resources — e.g. `snowflake_role` → `snowflake_account_role`, `snowflake_grant_privileges_to_role` → `snowflake_grant_privileges_to_account_role` — so older examples online often won't work.

```

# providers.tf – add Snowflake provider
terraform {
  required_providers {
    snowflake = {
      source  = "snowflakedb/snowflake"
      version = "~> 2.0"
    }
  }
}

provider "snowflake" {

```

```

organization_name = var.snowflake_organization
account_name      = var.snowflake_account
user              = var.snowflake_user
authenticator     = "SNOWFLAKE_JWT"           # key-
pair auth for automation
private_key       = var.snowflake_private_key # pass
via env/secret store, never commit
role              = "SYSADMIN"
}

# Databases
resource "snowflake_database" "raw" {
  name      = "RAW_${upper(var.environment)}"
  comment   = "Raw ingested data – bronze layer"
}

resource "snowflake_database" "analytics" {
  name      = "ANALYTICS_${upper(var.environment)}"
  comment   = "Transformed data – silver and gold layers"
}

# Schemas
resource "snowflake_schema" "raw_orders" {
  database = snowflake_database.raw.name
  name     = "ORDERS"
}

resource "snowflake_schema" "marts" {
  database = snowflake_database.analytics.name
  name     = "MARTS"
}

# Virtual warehouses
resource "snowflake_warehouse" "transform" {
  name              = "TRANSFORM_WH_${upper(var.environment)}"
  warehouse_size    = var.environment == "prod" ? "MEDIUM" :
"XSMALL"
  auto_suspend      = 60
  auto_resume       = "true"           # string in provider v1+
("true" / "false")
  comment           = "Used by dbt transformations"
}

```

```

resource "snowflake_warehouse" "reporting" {
  name          = "REPORTING_WH_${upper(var.environment)}"
  warehouse_size = "SMALL"
  auto_suspend  = 300
  auto_resume   = "true"
}

# Roles
resource "snowflake_account_role" "analyst" {
  name      = "ANALYST_${upper(var.environment)}"
  comment   = "Read access to gold layer tables"
}

resource "snowflake_account_role" "transformer" {
  name      = "TRANSFORMER_${upper(var.environment)}"
  comment   = "Used by dbt to run transformations"
}

# Grants – analysts can read all future tables in MARTS
resource "snowflake_grant_privileges_to_account_role"
"analyst_select" {
  account_role_name = snowflake_account_role.analyst.name
  privileges        = ["SELECT"]
  on_schema_object {
    future {
      object_type_plural = "TABLES"
      in_schema          =
snowflake_schema.marts.fully_qualified_name
    }
  }
}

resource "snowflake_grant_privileges_to_account_role"
"transformer_all" {
  account_role_name =
snowflake_account_role.transformer.name
  all_privileges    = true
  on_schema {
    schema_name =
snowflake_schema.marts.fully_qualified_name
  }
}

```

```

}

# Service user for dbt (TYPE = SERVICE: key-pair auth, no
password, no MFA prompts)
resource "snowflake_service_user" "dbt" {
  name          = "DBT_SA_${upper(var.environment)}"
  login_name    = "dbt_sa_${var.environment}"
  default_role  =
snowflake_account_role.transformer.name
  default_warehouse = snowflake_warehouse.transform.name
  rsa_public_key = var.dbt_rsa_public_key
}

resource "snowflake_grant_account_role" "dbt_sa_role" {
  role_name = snowflake_account_role.transformer.name
  user_name = snowflake_service_user.dbt.name
}

```

Databricks with Terraform

```

# providers.tf
terraform {
  required_providers {
    databricks = {
      source = "databricks/databricks"
      version = "~> 1.40"
    }
  }
}

provider "databricks" {
  host = var.databricks_host # e.g. https://adb-
xxx.azure.databricks.net
  token = var.databricks_token # use env:
DATABRICKS_TOKEN
}

# Cluster
resource "databricks_cluster" "etl" {
  cluster_name = "etl-

```

```

cluster-${var.environment}"
  spark_version          = "14.3.x-scala2.12"
  node_type_id           = var.environment == "prod" ?
"Standard_DS5_v2" : "Standard_DS3_v2"
  autotermination_minutes = 30

  autoscale {
    min_workers = 2
    max_workers = 8
  }

  spark_conf = {
    "spark.databricks.delta.preview.enabled" = "true"
  }

  library {
    pypi { package = "dbt-databricks==1.7.0" }
  }
}

# Secrets scope (backed by Azure Key Vault or Databricks-
managed)
resource "databricks_secret_scope" "pipeline_secrets" {
  name = "pipeline-secrets-${var.environment}"
}

resource "databricks_secret" "snowflake_password" {
  key          = "snowflake-password"
  string_value = var.snowflake_password
  scope        =
databricks_secret_scope.pipeline_secrets.name
}

# Job
resource "databricks_job" "daily_pipeline" {
  name = "daily-orders-pipeline-${var.environment}"

  task {
    task_key = "load_bronze"
    existing_cluster_id = databricks_cluster.etl.id
    notebook_task {
      notebook_path = "/Repos/data-

```

```

team/pipelines/bronze/load_orders"
    }
}

task {
    task_key = "transform_silver"
    depends_on { task_key = "load_bronze" }
    existing_cluster_id = databricks_cluster.etl.id
    notebook_task {
        notebook_path = "/Repos/data-
team/pipelines/silver/transform_orders"
    }
}

schedule {
    quartz_cron_expression = "0 0 2 * * ?"    # 2am daily
    timezone_id            = "UTC"
    pause_status           = var.environment == "prod" ?
"UNPAUSED" : "PAUSED"
}
}

# Unity Catalog
resource "databricks_catalog" "main" {
    name      = "main"
    comment   = "Main data catalog"
}

resource "databricks_schema" "bronze" {
    catalog_name = databricks_catalog.main.name
    name         = "bronze"
    comment      = "Raw ingested data"
}

resource "databricks_grants" "bronze_schema" {
    schema =
"${databricks_catalog.main.name}.${databricks_schema.bronze
.name}"
    grant {
        principal = "data-team@example.com"
        privileges = ["USE_SCHEMA", "CREATE_TABLE", "SELECT"]
    }
}

```

```
}  
}
```

Airflow Infra on AWS

```
# MWA (Managed Workflows for Apache Airflow) on AWS  
  
resource "aws_s3_bucket" "airflow" {  
  bucket = "mycompany-airflow-${var.environment}"  
}  
  
resource "aws_s3_object" "dags_folder" {  
  bucket = aws_s3_bucket.airflow.id  
  key    = "dags/"  
  content = ""  
}  
  
resource "aws_s3_object" "requirements" {  
  bucket = aws_s3_bucket.airflow.id  
  key    = "requirements.txt"  
  source = "${path.module}/requirements.txt"  
  etag   = filemd5("${path.module}/requirements.txt")  
}  
  
resource "aws_mwaa_environment" "airflow" {  
  name = "mycompany-airflow-${var.environment}"  
  airflow_version = "2.8.1"  
  environment_class = var.environment == "prod" ?  
    "mw1.medium" : "mw1.small"  
  execution_role_arn = aws_iam_role.mwaa.arn  
  
  source_bucket_arn = aws_s3_bucket.airflow.arn  
  dag_s3_path       = "dags/"  
  requirements_s3_path = "requirements.txt"  
  
  airflow_configuration_options = {  
    "core.default_timezone" = "UTC"  
    "scheduler.dag_dir_list_interval" = "30"  
    "webserver.dag_default_view" = "graph"  
  }  
}
```

```

}

network_configuration {
  security_group_ids = [aws_security_group.mwaa.id]
  subnet_ids        = var.private_subnet_ids
}

logging_configuration {
  dag_processing_logs {
    enabled  = true
    log_level = "WARNING"
  }
  scheduler_logs {
    enabled  = true
    log_level = "WARNING"
  }
  task_logs {
    enabled  = true
    log_level = "INFO"
  }
  webserver_logs {
    enabled  = true
    log_level = "WARNING"
  }
  worker_logs {
    enabled  = true
    log_level = "WARNING"
  }
}
}

```

Common Pitfalls

1. Committing terraform.tfvars or .tfstate to git 
 Problem: exposes credentials and infrastructure details
 Fix: Add to .gitignore; use remote backend for state;
 use env vars or Secrets Manager for sensitive values

2. No remote state backend

Problem: two engineers applying at the same time = corrupted state

Fix: Always configure a remote backend with locking from day one

(S3 with `use_lockfile = true`, Terraform Cloud, or GCS/Azure backends)

3. Using terraform destroy in production

Problem: destroys everything, including live data

Fix: Use targeted destroy (`-target`), or set `prevent_destroy = true` on critical resources

4. Not running terraform plan before apply

Problem: surprises – resources you didn't expect to be destroyed

Fix: Always review plan output; in CI, require plan before merge

5. Hardcoding account IDs and region strings

Problem: not reusable across environments

Fix: Use data sources and variables

6. Single monolithic main.tf with 1000 lines

Problem: hard to navigate, long plans, blast radius too large

Fix: Split by service (`storage.tf`, `iam.tf`, `warehouse.tf`, `compute.tf`)
or by lifecycle (long-lived vs frequently-changed resources)

7. Not using lifecycle { prevent_destroy = true } on data resources

Problem: a typo in `bucket_name` causes Terraform to destroy and recreate

Fix:

```
resource "aws_s3_bucket" "data_lake" {  
  bucket = "mycompany-data-lake-prod"  
  lifecycle { prevent_destroy = true }  
}
```

8. Importing existing resources manually instead of into state

Problem: Terraform thinks the resource doesn't exist → tries to create → conflict

Fix: An import block (Terraform 1.5+), reviewed in plan like any change:

```
import {
  to = aws_s3_bucket.data_lake
  id = "mycompany-data-lake-prod"
}
```

or the CLI: `terraform import aws_s3_bucket.data_lake mycompany-data-lake-prod`

.gitignore for Terraform

```
# Terraform state – never commit
*.tfstate
*.tfstate.*
.terraform/
# .terraform.lock.hcl – DO commit this: it pins exact
provider versions and checksums
terraform.tfvars      # contains secrets – use
.tfvars.example instead
*.tfvars.backup
crash.log
override.tf
override.tf.json
```

Cheat Sheet

Task	Command
Download providers, configure the backend	<code>terraform init</code> (<code>-upgrade</code> to update providers within constraints)
Format / validate	<code>terraform fmt -recursive</code> · <code>terraform validate</code>
Preview changes	<code>terraform plan -out=tfplan</code>

Task	Command
Apply exactly what was reviewed	<code>terraform apply tfplan</code>
Per-environment values	<code>terraform plan -var-file=envs/prod.tfvars</code>
List / inspect state	<code>terraform state list</code> · <code>terraform state show <addr></code>
Rename or move a resource without recreating it	<code>a moved { from = ... to = ... }</code> block (or <code>terraform state mv</code>)
Bring an existing resource under management	<code>an import { to = ..., id = ... }</code> block (1.5+) · <code>terraform plan -generate-config-out=gen.tf</code>
Stop managing something without deleting it	<code>a removed { from = ... }</code> block (1.7+) or <code>terraform state rm</code>
Force a resource to be replaced	<code>terraform apply -replace=<addr></code>
Detect drift	<code>terraform plan -refresh-only</code>
Show outputs	<code>terraform output -json</code>

Language essentials

```

locals { name = "${var.project}-${var.environment}" }
# computed values

resource "aws_s3_bucket" "layer" {
# many from a map/set
  for_each = toset(["bronze", "silver", "gold"])
  bucket   = "${local.name}-${each.key}"
}

resource "aws_s3_bucket" "logs" {
  count = var.environment == "prod" ? 1 : 0
# conditional resource
  bucket = "${local.name}-logs"
}

data "aws_caller_identity" "me" {}
# read, don't manage

```

```
# Inside any resource holding data: lifecycle {  
  prevent_destroy = true }
```

Project layout: reusable `modules/` · one root configuration per environment (or workspace) · a separate state file per environment and per blast radius (network, data platform, IAM)

Safe CI flow: `fmt -check` → `validate` → `plan` posted on the PR → review → `apply` of the saved plan on merge, using short-lived cloud credentials (OIDC), never long-lived keys

Interview Questions

Q: What is Terraform state and why is it important? A: State is Terraform's record of which real resources correspond to which blocks in your code, plus their last-known attributes. Terraform uses it to calculate plans (what to create, change, or destroy) and to track dependencies. It must be stored remotely (S3, GCS, Terraform Cloud) with locking so two people can't apply at the same time, versioned so it can be recovered, and treated as sensitive, because it can contain secrets in plain text.

Q: What happens during `terraform plan` and `apply`? A: `plan` refreshes state against the real infrastructure, compares that with your configuration, and produces an execution plan: resources to create (+), update in place (~), or destroy and recreate (-/+). `apply` executes the plan through each provider's API in dependency order, updating state as it goes. Saving the plan (`-out`) and applying that exact file guarantees you apply what was reviewed.

Q: How do you manage multiple environments with Terraform? A: Common approaches: a separate root configuration and state per environment that call shared modules with different variables (the most explicit and most common); Terraform workspaces (one configuration, several states — convenient but easy to apply to the wrong one); or wrappers like Terragrunt. Whichever you choose, keep state separate per environment so a mistake in dev can't touch prod.

Q: What is drift and how do you handle it? A: Drift is when real infrastructure no longer matches the code — usually because someone changed something in the console. `terraform plan` (or `plan -refresh-only`) reveals it. You either bring the code in line with the change, if it was intentional, or let the next apply revert it. Prevent it with restricted console permissions and scheduled plans that alert on unexpected differences.

Q: How do you handle secrets in Terraform? A: Don't put them in `.tf` or committed `.tfvars` files. Pass them through environment variables (`TF_VAR_...`) from CI secrets, or read them at apply time from a secrets manager with a data source. Mark variables

`sensitive = true` to hide them in output. Remember that values still end up in state, so lock down and encrypt the state backend — or better, have Terraform create the secret *container* and let another process set the value.

Q: What does a data engineer typically manage with Terraform? A: Storage (buckets with encryption, versioning, and lifecycle rules), IAM roles and policies for pipelines, warehouse objects (databases, schemas, compute, roles, grants, service users), compute platforms (workspaces, clusters, jobs, catalogs), Kafka topics, and orchestration environments like MWAA or Composer. Anything that should be identical across environments and reviewable in a PR is a good candidate.

Further Reading

- [Terraform documentation](https://developer.hashicorp.com/terraform/docs) (https://developer.hashicorp.com/terraform/docs) and [tutorials](https://developer.hashicorp.com/terraform/tutorials) (https://developer.hashicorp.com/terraform/tutorials)
 - [Terraform Registry](https://registry.terraform.io/) (https://registry.terraform.io/) — provider docs for [AWS](https://registry.terraform.io/providers/hashicorp/aws/latest/docs) (https://registry.terraform.io/providers/hashicorp/aws/latest/docs), [Snowflake](https://registry.terraform.io/providers/snowflake/snowflake/latest/docs) (https://registry.terraform.io/providers/snowflake/snowflake/latest/docs), and [Databricks](https://registry.terraform.io/providers/databricks/databricks/latest/docs) (https://registry.terraform.io/providers/databricks/databricks/latest/docs)
 - [OpenTofu](https://opentofu.org/) (https://opentofu.org/) — the open-source fork of Terraform, largely compatible
 - [tflint](https://github.com/terraform-linters/tflint) (https://github.com/terraform-linters/tflint) and [Checkov](https://www.checkov.io/) (https://www.checkov.io/) — linting and security scanning for Terraform
 - *Terraform: Up & Running, 3rd Edition* — Yevgeniy Brikman (O'Reilly)
-

Previous: [Kubernetes](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/kubernetes-for-de/) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/kubernetes-for-de/) • **Next:** [Testing and CI/CD](https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/testing-cicd/) (https://sarangambekar1997.github.io/data-engineering-handbook/06-infrastructure/testing-cicd/) • **Back to:** [Index](https://sarangambekar1997.github.io/data-engineering-handbook/) (https://sarangambekar1997.github.io/data-engineering-handbook/)

 September 29, 2026