Terraform for Teams — Structured Guide

From first config to reusable modules, with clear definitions, conventions, and runnable examples.

Contents

1. Key concepts & workflow

- **Declarative**: describe desired state. Terraform calculates the diff.
- **State**: source of truth for what Terraform manages; use remote state + locking.
- **Blocks**: *terraform*, *provider*, *resource*, *data*, *module*, *variable*, *locals*, *output*.
- **Workflow**: init → validate → plan → apply → destroy

```
terraform init -upgrade terraform fmt -check && terraform validate terraform plan -out=tfplan
terraform apply tfplan
```

2. Config anatomy & resource block

A production■ready minimal configuration:

```
# versions.tf terraform { required_version = ">= 1.7" required_providers { aws = { source =
"hashicorp/aws", version = "~> 5.0" } } backend "s3" { bucket = "org-terraform-state" key =
"platform/example/terraform.tfstate" region = "ap-south-1" dynamodb_table =
"org-terraform-locks" } } # providers.tf provider "aws" { region = var.region }
```

Resource block anatomy (fields you set are *arguments*; values Terraform exposes are *attributes*):

```
resource "aws_s3_bucket" "logs" { bucket = "${var.app}-logs-${var.region}" # argument
(input) force_destroy = true # argument tags = merge(var.tags, { "component" = "logs" }) #
argument } # After apply, these attributes are available: # aws_s3_bucket.logs.arn, .id,
.bucket_domain_name, ...
```

3. Attributes vs Parameters (and Arguments)

Term	Definition	Where used	Example
Argument (input)	Field you set inside a block to configure it.	resource/data/module blocks	bucket = var.name
Attribute (output)	Computed or known value exported by a resource instance.	read via references	aws_s3_bucket.logs.arn
Parameter	Colloquial for a module input; in Terraform, this is a variable.	inside module calls	module "vpc" { cidr = var.vpc_cidr }
Variable	Named input value for root or child modules.	variables.tf / *.tfvars	variable "vpc_cidr" { type = string }

Rule of thumb: you *set* arguments; you *read* attributes; you *pass* variables (parameters) across modules.

4. Types, variables, and locals

- Primitive: string, number, bool.
- Collections: list(T), set(T), map(T). Structures: object({}), tuple([..]).
- Defaults must be **literal**; no references or functions in defaults.

- Variable sources precedence: tfvars files → TF_VAR_* env → var/-var file

```
# variables.tf variable "region" { type = string } variable "tags" { type = map(string) }
variable "subnets" { type = list(string) } # locals.tf locals { name_prefix =
"${var.env}-${var.app}" dns_suffix = lower(replace(local.name_prefix, " ", "-")) }
```

```
# example.auto.tfvars env = "dev" app = "ml-platform" tags = { app = "ml-platform", owner =
"data" }
```

5. Expressions & interpolation

- Modern style: bucket = var.name (instead of "${var.name}").
- String templates still valid for complex strings: "${local.first}-${local.last}".
- Use functions: *merge*, *format*, *replace*, *contains*, *lookup*, *coalesce*.

6. Data sources & dependencies

Data sources fetch information without creating resources. Dependencies are usually inferred by references; use **depends_on** sparingly.

```
data "aws_caller_identity" "me" {} data "aws_availability_zones" "available" {} resource
"aws_iam_user" "svc" { name = "${var.app}-svc" tags = var.tags } resource
"aws_iam_user_policy_attachment" "attach" { user = aws_iam_user.svc.name # implicit
dependency policy_arn = "arn:aws:iam::aws:policy/ReadOnlyAccess" }
```

7. Modules — consuming & authoring

Consume vetted registry modules and author thin, composable internal modules.

```
# Consuming a registry module module "vpc" { source = "terraform-aws-modules/vpc/aws" version
= "~> 5.0" name = "${var.env}-core" cidr = var.vpc_cidr azs =
slice(data.aws_availability_zones.available.names, 0, 2) tags = var.tags }
```

```
# Authoring a simple internal module: modules/s3_logs # modules/s3_logs/variables.tf
variable "name" { type = string } variable "tags" { type = map(string) } #
modules/s3_logs/main.tf resource "aws_s3_bucket" "this" { bucket = "${var.name}-logs"
force_destroy = true tags = var.tags } # modules/s3_logs/outputs.tf output "arn" { value =
aws_s3_bucket.this.arn }
```

```
# Root module usage module "logs" { source = "./modules/s3_logs" name =
"${var.env}-${var.app}" tags = var.tags } output "logs_bucket_arn" { value = module.logs.arn
}
```

8. Project structure & naming conventions

Recommended layout for a single environment:

```
./ ■■ versions.tf # Terraform + providers + backend ■■ providers.tf ■■ main.tf # resources
& module calls ■■ variables.tf ■■ locals.tf ■■ outputs.tf ■■ example.auto.tfvars #
sample/default inputs (non-secret) ■■ modules/ ■■ s3_logs/ ■■ main.tf ■■ variables.tf ■■
outputs.tf
```

- Variable names: snake_case. Resource names: short and stable.
- Strings in kebab-case for tags/names in AWS where allowed.
- Always pin versions for providers and modules.
- Always use remote state (S3) with DynamoDB locking.

9. End■to■end examples

Example A — S3 + IAM policy attachment:

```
# main.tf module "logs" { source = "./modules/s3_logs" name = "${var.env}-${var.app}" tags =
var.tags } data "aws_iam_policy_document" "ro" { statement { actions =
["s3:GetObject","s3:ListBucket"] resources = [module.logs.arn, "${module.logs.arn}/*"] } }
resource "aws_iam_policy" "ro" { name = "${var.app}-logs-ro"; policy =
data.aws_iam_policy_document.ro.json }
```

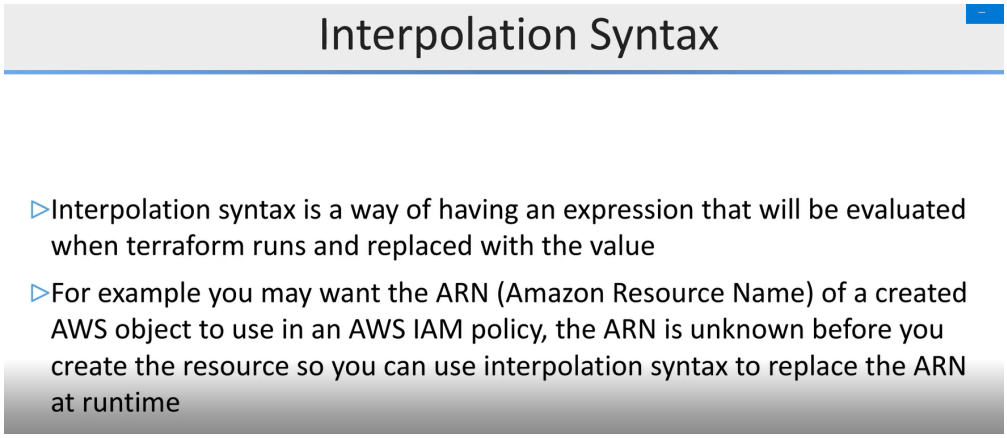
Example B — Security groups via for_each:

```
variable "sg_rules" { type = map(object({ from_port = number, to_port = number, protocol = string, cidr = list(string) })) } resource "aws_security_group" "svc" { name = "${var.app}-svc" description = "App SG" vpc_id = var.vpc_id } resource "aws_security_group_rule" "ingress" { for_each = var.sg_rules type = "ingress" security_group_id = aws_security_group.svc.id from_port = each.value.from_port to_port = each.value.to_port protocol = each.value.protocol cidr_blocks = each.value.cidr }
```

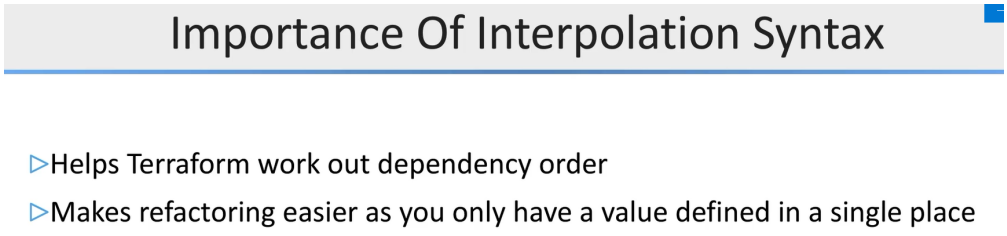
10. Common pitfalls & checklist

- ■ Expressions in variable defaults (not allowed).
- ■ Manual edits to cloud resources outside Terraform.
- ■ Unpinned provider/module versions.
- ■ CI: fmt, validate, plan; PRs include plan artifact.
- ■ Consistent tags & names; secrets via external stores.

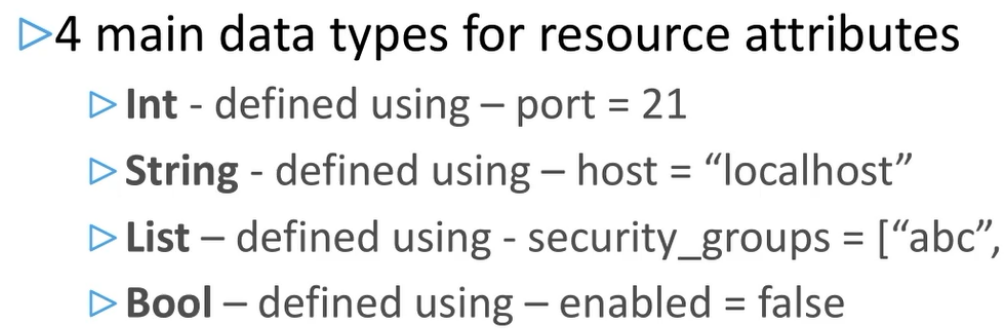
Appendix — Reference images from your slides



Resource block anatomy & arguments vs attributes.



Interpolation syntax: evaluated at runtime to get values like ARNs.



Basic types visible in resource attributes.

Terraform Locals

- Terraform Locals allow you to assign a name to an expression, you can think of them like a variable
- You can use a local value multiple times within a module
- Local values can be combined to make more local values

- Locals allow you to define a value in one place and then you can use the local in your project. This means if you change the value of the local you only have to update it in a single place

- Locals can be combined to make more locals

```
locals {
  first = "kev"
  last  = "holditch"
  name  = "${local.first}-${local.last}"
}
```

Terraform Locals — reusable expressions.

- Variables serve as parameters to a Terraform module
- When used at the top level they enable you to pass parameters into your Terraform project
  - Command line
  - File
  - Environment variables

- 3 Properties can be defined on a variable:
  - Type: (optional) if this is set it defines the type of the variable, if no type is set then the type of the default value will be used if neither are set the type will be assumed to be string. Allowed values “string”, “list” and “map”
  - Default: (optional) if this is set then the variable will take this value if you do not pass one in. If no default value is set and a value is not passed in then Terraform will raise an error
  - Description: (optional) purely to give the user of the Terraform project some information as to what this variable is used for. Terraform ignores this field

- Default value of a variable must be a literal value for example 1, “foo” or [“a”, ”b”]. Interpolation syntax is not allowed

  - Examples of variable declarations:

```
variable "key" {
  type = "string"
}
variable "images" {
  type = "map"

  default = {
    us-east-1 = "image-1234"
    us-west-2 = "image-4567"
  }
}
variable "zones" {
  default = ["us-east-1a", "us-east-1b"]
}
```

- Map variables allow you to define values for Terraform to use in different cases for example you could define a map to specify which instance size to use depending on your environment type:

```
variable "instance_size_map" {
  type = "map"
  default = {
    dev     = "t2.micro",
    test    = "t2.medium",
    prod    = "m4.large"
  }
}
```

  - To lookup a map value use the lookup function by:
    - "${lookup(<map_name>, <map_key>)}”

```
variable "instance_size_map" {
  type = "map"
  default = {
    dev     = "t2.micro",
    test    = "t2.medium",
    prod    = "m4.large"
  }
}
variable "instance_size" {}
```

Variables — declarations, defaults, and maps.