
Terraform
RESOURCES

❖ RESOURCES:

- Resources are the most important element in the Terraform language.
- Each resource block describes one or more infrastructure objects, such as virtual networks, compute instances, or higher-level components such as DNS records.

EXAMPLE:

```
resource "aws_instance" "web" {
  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"
}
```

→ The resource type **("aws_instance")** and name **("Web")** together must be unique.
→ Within the block body **(between { and })** are the configuration arguments for the resource itself.
→ **ami** and **instance_type** are special arguments for the aws_instance resource type.

➢ RESOURCE TYPES:

- Each resource is associated with a single resource type, which determines the kind of infrastructure object it manages and what arguments and other attributes the resource supports.
  - PROVIDERS
  - RESOURCE ARGUMENTS
  - META-ARGUMENTS
  - CUSTOM CONDITION CHECKS
  - OPERATION TIMEOUTS

PROVIDERS:

- A provider is a plugin for Terraform that offers a collection of resource types. Each resource type is implemented by a provider.
- A provider provides resources to manage a single cloud or on-premises infrastructure platform.

RESOURCE ARGUMENTS:

- Most of the arguments within the body of a resource block are specific to the selected resource type.
- The values for resource arguments can make full use of expressions and other dynamic Terraform language features.

META-ARGUMENTS:

- It can be used with any resource type to change the behavior of resources:
  - **depends_on** : for specifying hidden dependencies
  - **count** : for creating multiple resource instances according to a count
  - **for_each** : to create multiple instances according to a map, or set of strings
  - **provider** : for selecting a non-default provider configuration
  - **lifecycle** : for lifecycle customizations
  - **provisioner** : for taking extra actions after resource creation

EXAMPLE:

```
resource "aws_instance" "app_server" {
  count          = 1
  ami            = "ami-079db87dc4c10ac91"
  instance_type  = "t2.micro"
  subnet_id      = "subnet-060027219f6e3dae6"
  security_groups = ["sg-04cecc7e117da949e"]
  key_name       = "ram"
  tags = {
    Name = "My-Server"
  }
}
```

CUSTOM CONDITION CHECKS:

- You can use precondition and postcondition blocks to specify assumptions and guarantees about how the resource operates.

```
resource "aws_instance" "example" {
  instance_type = "t2.micro"
  ami           = "ami-abc123"
  lifecycle {
    # The AMI ID must refer to an AMI that contains an os.
    # for the `x86_64` architecture.
      precondition {
      condition     = data.aws_ami.example.architecture == "x86_64"
      error_message = "The selected AMI must be for the x86_64 architecture."
      }
  }
}
```

$terraform fmt : It is used to rewrite Terraform configuration files to a canonical format and style.

$terraform validate : It runs checks that verify whether a configuration is syntactically valid and internally consistent, regardless of any provided variables or existing state.

$terraform plan : It creates an execution plan, which lets you preview the changes that Terraform plans to make to your infrastructure. By default, when Terraform creates a plan it:$terraform plan

$terraform apply : It executes the actions proposed in a Terraform plan.$terraform apply

$terraform destroy : It is a convenient way to destroy all remote objects managed by a particular Terraform configuration.

OPERATION TIMEOUTS:

- Some resource types provide a special timeouts nested block argument that allows you to customize how long certain operations are allowed to take before being considered to have failed.
- For example, aws_db_instance allows configurable timeouts for create, update, and delete operations.

```
resource "aws_db_instance" "example" {
  # ...
  timeouts {
    create = "60m"
    delete = "2h"
  }
}
```

➢ WHAT IS .ID:

- The **.id** is used internally by Terraform to track resource dependencies and manage state.
- The **.id attribute** is primarily used in two ways:

REFERENCING RESOURCES:

- You can use the .id attribute to reference resources in your Terraform configuration.
- This is particularly useful when you need to create dependencies between resources.

OUTPUTTING RESOURCE IDS:

- You can output the .id of a resource using the output keyword.
- This is useful when you need to retrieve the ID of a resource for use outside of Terraform.

EXAMPLE:

```
resource "aws_s3_bucket" "example" {
  bucket = "cloud-aws-bucket"
  tags = {
    Name        = "My bucket"
    Environment = "Developer"
  }
}

resource "aws_s3_bucket_versioning" "versioning_example" {
  bucket = aws_s3_bucket.example.id
  versioning_configuration {
    status = "Enabled"
  }
}
```

```
$terraform fmt
$terraform validate
$terraform plan
$terraform apply
$terraform destroy
```