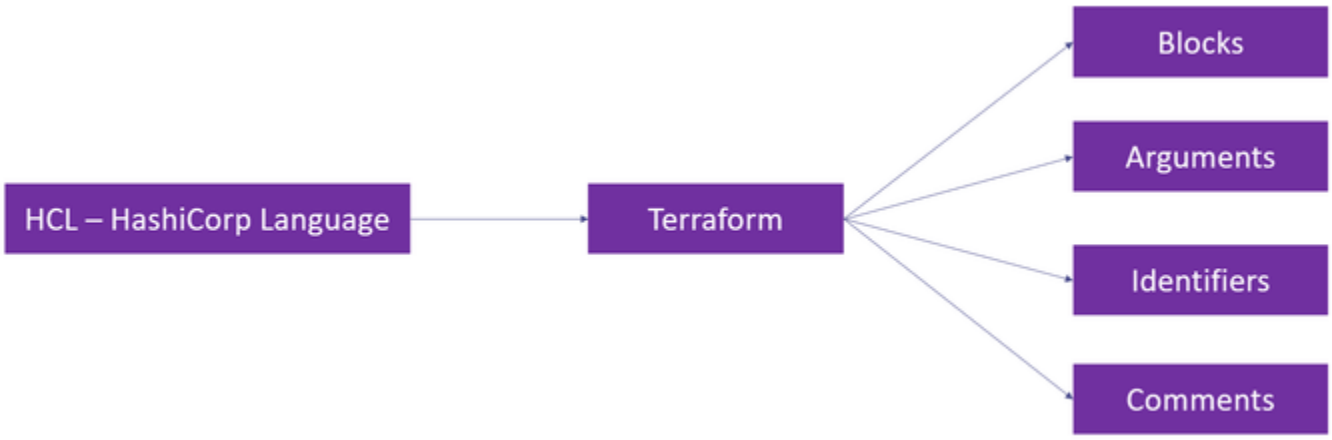
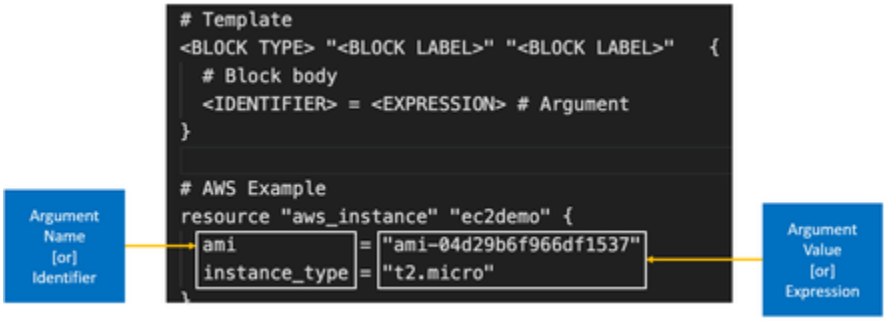
09-Terraform Configuration Syntax

The Terraform language syntax is built around two key syntax constructs: **arguments and blocks.**

Understand Terraform Language Basics

- Understand Blocks
- Understand Arguments (Key or identifiers and value or expression)
- Understand Meta-Arguments
- Understand Identifiers or arguments name
- Understand Attributes
- Understand Comments






```
# Template
<BLOCK TYPE> "<BLOCK LABEL>" "<BLOCK LABEL>"   {
  # Block body
  <IDENTIFIER> = <EXPRESSION> # Argument
}

# AWS Example
resource "aws_instance" "ec2demo" { # BLOCK
  ami           = "ami-04d29b6f966df1537" # Argument
  instance_type = var.instance_type # Argument with value as expression
(Variable value replaced from varibales.tf)
}
```

Blocks:

A *block* is a container for other content

`aws_instance` = Resource type or Block label

`ec2demo` = Reference or label. This should be unique in the entire code. It is used to refer to the resource elsewhere

`resource` = Block type

```
# Resource Block
resource "aws_instance" "ec2demo" {
---
---
---
---
---
}
```

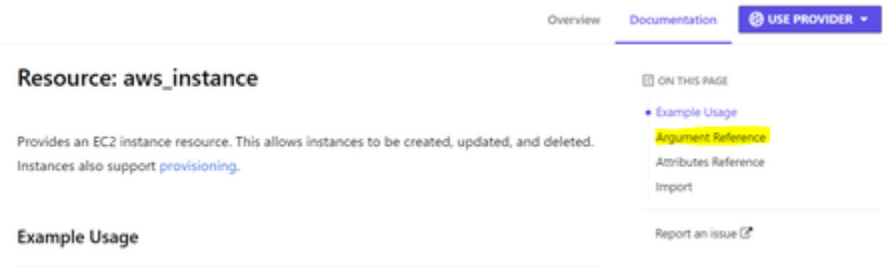
Arguments HERE

An *argument* assigns a value to a particular name

The identifier before the equals sign is the ***argument name***, and the expression after the equals sign is the argument's **value.**

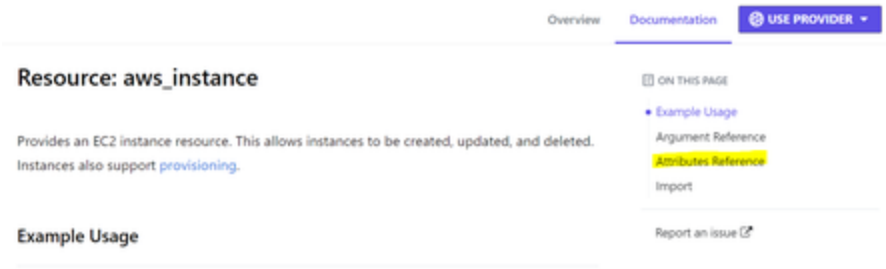
```
# Resource Block
resource "aws_instance" "ec2demo" {
  ami           = "ami-04d29b6f966df1537" # Arguments
  instance_type = "t2.micro" #Arguments
}
```

- Arguments can be `required` or `optional`. Terraform will fail to create a resource if all required arguments are not provided. When you want to create a resource, make sure you put all required arguments in the block
- **AMI:** it is a required argument to create an EC2 instance
- **Instance_type** is a required argument HERE
- Resource: AWS Instance Argument Reference



Attributes HERE

these are basically some elements that we want to display at the end when we run terraform. There are mostly used in `output.tf` file.




```
output "public_ip" {
  description = "Public IP of instance (or EIP)"
  value       = concat(aws_eip.default.*.public_ip, aws_instance.
default.*.public_ip, [""])[0]
}

output "private_ip" {
  description = "Private IP of instance"
  value       = join("", aws_instance.default.*.private_ip)
}

output "private_dns" {
  description = "Private DNS of instance"
  value       = join("", aws_instance.default.*.private_dns)
}

output "public_dns" {
  description = "Public DNS of instance (or DNS of EIP)"
  value       = local.public_dns
}

output "id" {
  description = "Disambiguated ID of the instance"
  value       = join("", aws_instance.default.*.id)
}

output "arn" {
  description = "ARN of the instance"
  value       = join("", aws_instance.default.*.arn)
}

output "name" {
  description = "Instance name"
  value       = module.this.id
}

output "ssh_key_pair" {
  description = "Name of the SSH key pair provisioned on the instance"
  value       = var.ssh_key_pair
}
```

```
output "security_group_ids" {
  description = "IDs on the AWS Security Groups associated with the
instance"
  value = compact(
    concat(
      formatlist("%s", module.security_group.id),
      var.security_groups
    )
  )
}
```

Meta-Arguments HERE

They are used to change the behavior of the resource. We have 5 types Meta-Arguments in terraform:

- depends_on Meta-Argument
- count Meta-Argument
- for_each Meta-Argument
- lifecycle Meta-Argument
- provider Meta-Argument

depends_on Meta-Argument:

This is when one resource depends on another resource

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

data "aws_caller_identity" "my_account" {}


resource "aws_s3_bucket" "bucket1" {
  bucket = "${data.aws_caller_identity.my_account.id}-bucket1"
}

resource "aws_s3_bucket" "bucket2" {
  bucket = data.aws_caller_identity.my_account.id
  depends_on = [
    aws_s3_bucket.bucket1
   ]
}
```

```
resource "aws_s3_bucket" "example" {
  acl    = "private"
}

resource "aws_instance" "example_c" {
  ami           = data.aws_ami.amazon_linux.id
  instance_type = "t2.micro"

  depends_on = [aws_s3_bucket.example]
}
```

count Meta-Argument

It is used to create multiple resources at once

```
resource "aws_instance" "server" {
  count = 4 # create four similar EC2 instances

  ami           = "ami-a1b2c3d4"
  instance_type = "t2.micro"

  tags = {
    Name = "Server ${count.index}"
  }
```

for_each Meta-Argument

It is used to create multiple resources at once using a for loop

```
rovider "aws" {
  region = "us-east-1"
}

variable "users" {
  type    = list(string)
  default = ["bob", "paul", "jhon"]
}

resource "aws_iam_user" "users" {
  for_each = toset(var.users)
  name     = each.value
  //name     = each.key
}
```

lifecycle Meta-Argument HERE

`lifecycle` is a nested block that can appear within a resource block. The `lifecycle` block and its contents are **meta-arguments**, available for all `resource` blocks regardless of type.

The following arguments can be used within a `lifecycle` block:

- `create_before_destroy` **(bool)** - By default, Terraform will instead destroy the existing object and then create a new replacement object. When `create_before_destroy = true`, Terraform create a new resource first before destroying the existing one if there were any modification

```
resource "azurerm_resource_group" "example" {
  # ...

  lifecycle {
    create_before_destroy = true
  }
}
```

- `prevent_destroy` **(bool)** - This meta-argument, when set to `true`, will cause Terraform to reject with an error any plan that would destroy the infrastructure object associated with the resource, as long as the argument remains present in the configuration. This will enable **deletion protection** on EC2, DB, and even S3 for instance

```
resource "azurerm_resource_group" "example" {
  # ...

  lifecycle {
    prevent_destroy = true
  }
}
```

- `ignore_changes` **(list of attribute names)** - By default, Terraform detects any difference in the current settings of a real infrastructure object and plans to update the remote object to match the configuration. Terraform should ignore when planning updates to the associated remote object.

```
resource "aws_instance" "example" {
  # ...

  lifecycle {
    ignore_changes = [
      aws_dynamodb_table.table.read_capacity,
      aws_dynamodb_table.table.write_capacity
      snapshot_identifier,
      tags,
    ]
  }
}
```

```
module "rds" {
  source = "terraform-aws-modules/rds/aws"
  ...
  engine_version = "5.7.33"

  lifecycle {
    ignore_changes = [
      engine_version
    ]
  }
  ...
 }
```

provider Meta-Argument

The `provider` meta-argument specifies which provider configuration to use for a resource, overriding Terraform's default provider

Multiple Provider Configurations (This allows us to deploy in multiple regions)

```
# The default provider configuration; resources that begin with `aws_`
will use
# it as the default, and it can be referenced as `aws`.
provider "aws" {
  region = "us-east-1"
}

# Additional provider configuration for west coast region; resources can
# reference this as `aws.west`.
provider "aws" {
  alias  = "west"
  region = "us-west-2"
}
```

```
# default configuration
provider "google" {
  region = "us-central1"
}

# alternate configuration, whose alias is "europe"
provider "google" {
  alias  = "europe"
  region = "europe-west1"
}

resource "google_compute_instance" "example" {
  # This "provider" meta-argument selects the google provider
  # configuration whose alias is "europe", rather than thedefault
configuration.
  provider = google.europe

  # ...
}
```

Comments

The Terraform language supports three different syntaxes for comments:

- # begins a single-line comment, ending at the end of the line.
- // also begins a single-line comment, as an alternative to #.
- /* and */ are the start and end delimiters for a comment that might span over multiple lines.

The # single-line comment style is the default comment style and should be used in most cases. Automatic configuration formatting tools may automatically transform // comments into # comments since the double-slash style is not idiomatic.