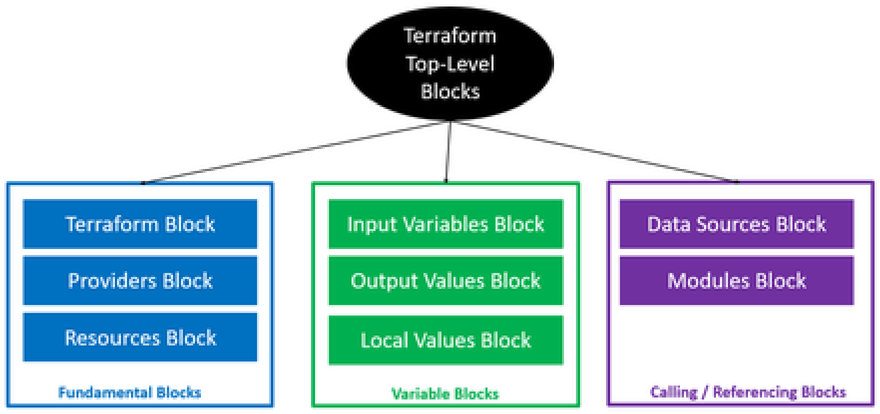
011-Terraform Top-Level Blocks

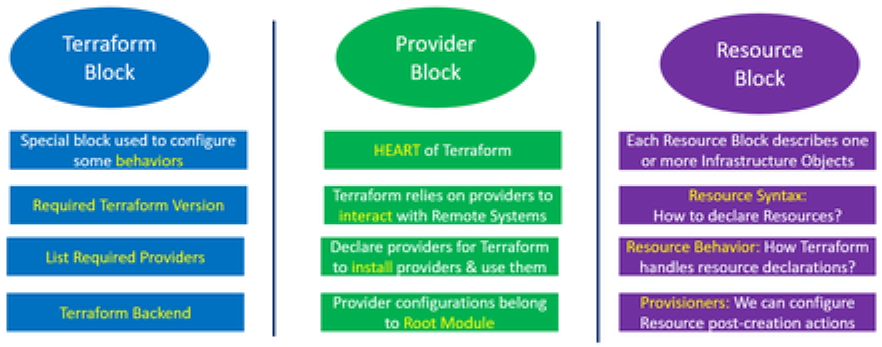
Terraform language uses a limited number of **top-level block** types, which are **blocks** that can appear outside of any other **block** in a TF configuration file.

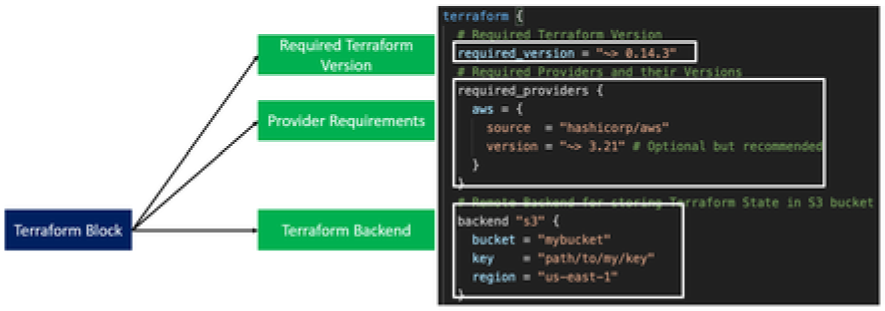
PS: If you master all these Top-Level Blocks, you master terraform at least **70-80%**

Understand Terraform Top-Level Blocks

- Discuss Terraform Top-Level blocks
  - Terraform Settings Block
  - Provider Block
  - Resource Block
  - Backend Block
  - Input Variables Block
  - Output Values Block
  - Local Values Block
  - Data Sources Block
  - Modules Block




Terraform Block
Special block used to configure some behaviors
Required Terraform Version
List Required Providers
Terraform Backend
Provider Block
HEART of Terraform
Terraform relies on providers to interact with Remote Systems
Declare providers for Terraform to install providers & use them
Provider configurations belong to Root Module
Resource Block
Each Resource Block describes one or more Infrastructure Objects
Resource Syntax: How to declare Resources?
Resource Behavior: How Terraform handles resource declarations?
Provisioners: We can configure Resource post-creation actions



```
terraform {
  # Required Terraform Version
  required_version = "~> 0.14.3"
  # Required Providers and their Versions
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.21" # Optional but recommended
    }
  }
  # Remote Backend for storing Terraform State in S3 bucket
  backend "s3" {
    bucket = "mybucket"
    key    = "path/to/my/key"
    region = "us-east-1"
  }
}
```

Terraform Settings Block

This block can be called in 3 ways. All means the same.

- Terraform Block
- Terraform Settings Block
- Terraform Configuration Block

```
# Block-1: Terraform Settings Block
terraform {
  required_version = "~> 0.14"
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
```

Backend Block

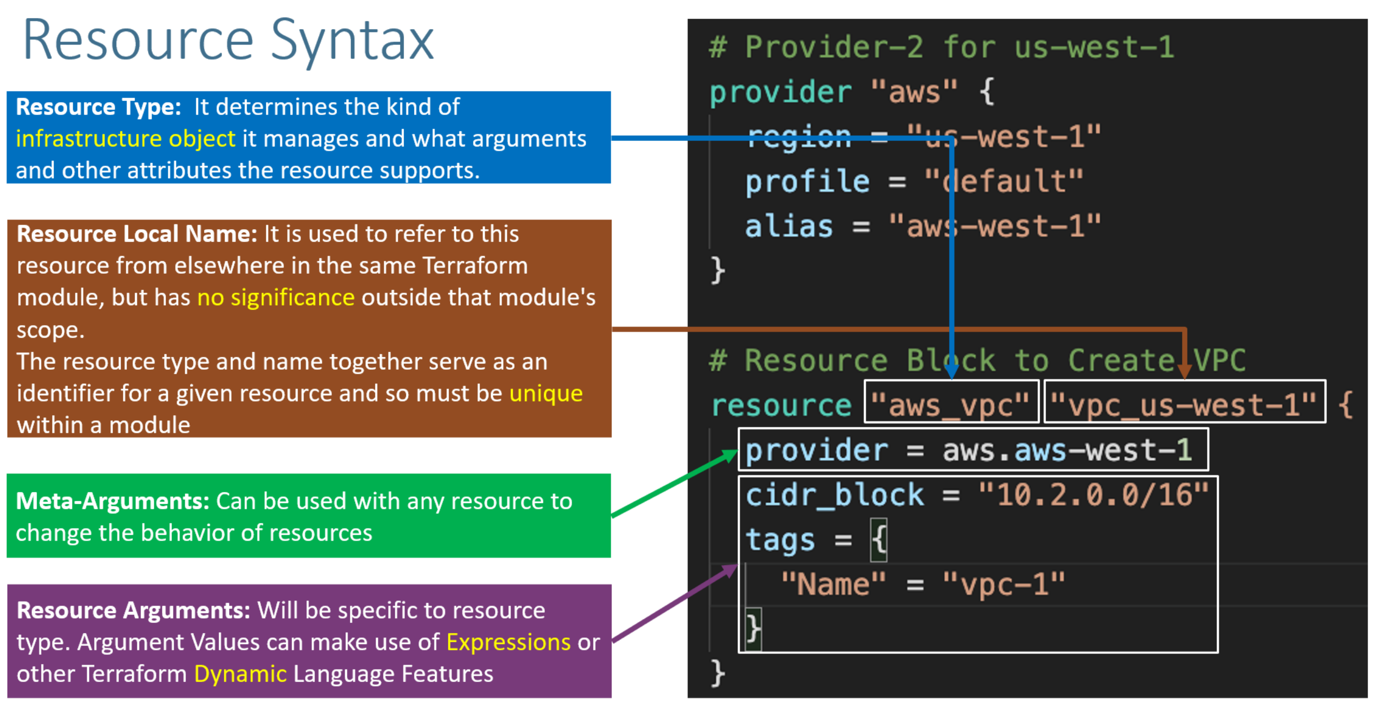
```
# Backend as S3 for Remote State Storage with State Locking
  backend "s3" {
    bucket = "terraform-statefile"
    key    = "dev2/terraform.tfstate"
    region = "us-east-1"

    # For State Locking
    dynamodb_table = "terraform-dev-state-table"
  }
}
```

Provider Block

```
provider "aws" {
  profile = "default" # AWS Credentials Profile configured on your
local desktop terminal  $HOME/.aws/credentials
  region  = "us-east-1"
}
```

Resource Block

```
resource "aws_instance" "ec2demo" {
  ami           = "ami-04d29b6f966df1537" # Amazon Linux
  instance_type = var.instance_type
}
```



Resource Syntax

Resource Type: It determines the kind of infrastructure object it manages and what arguments and other attributes the resource supports.

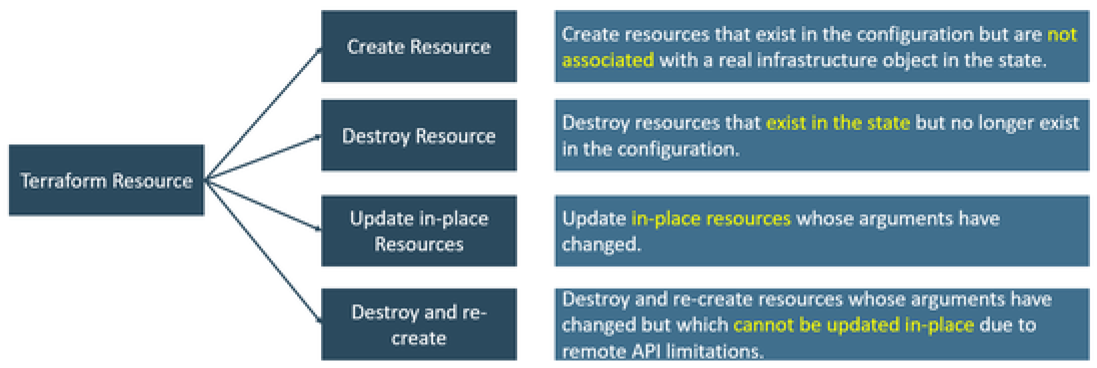
Resource Local Name: It is used to refer to this resource from elsewhere in the same Terraform module, but has no significance outside that module's scope.
The resource type and name together serve as an identifier for a given resource and so must be unique within a module

Meta-Arguments: Can be used with any resource to change the behavior of resources

Resource Arguments: Will be specific to resource type. Argument Values can make use of Expressions or other Terraform Dynamic Language Features

```
# Provider-2 for us-west-1
provider "aws" {
  region = "us-west-1"
  profile = "default"
  alias = "aws-west-1"
}

# Resource Block to Create VPC
resource "aws_vpc" "vpc_us-west-1" {
  provider = aws.aws-west-1
  cidr_block = "10.2.0.0/16"
  tags = {
    "Name" = "vpc-1"
  }
}
```

Resource Behavior



Create resources that exist in the configuration but are not associated with a real infrastructure object in the state.

Destroy resources that exist in the state but no longer exist in the configuration.

Update in-place resources whose arguments have changed.

Destroy and re-create resources whose arguments have changed but which cannot be updated in-place due to remote API limitations.

Input Variables Block

```
variable "instance_type" {
  default = "t2.micro"
  description = "EC2 Instance Type"
  type = string
}
```

Output Values Block

```
output "ec2_instance_publicip" {
  description = "EC2 Instance Public IP"
  value = aws_instance.my-ec2-vm.public_ip
}
```

Local Values Block

```
locals {
  common_tags = {
    Owner   = "DevOps Team"
    service = "backend"
  }
}
```

Data sources Block

```
# Get latest AMI ID for Amazon Linux2 OS
data "aws_ami" "amzlinux" {
  most_recent      = true
  owners           = ["amazon"]

  filter {
    name   = "name"
    values = ["amzn2-ami-hvm-*"]
  }

  filter {
    name   = "root-device-type"
    values = ["ebs"]
  }

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

  filter {
    name   = "architecture"
    values = ["x86_64"]
  }

}
```

Modules Block

```
# AWS EC2 Instance Module

module "ec2_cluster" {
  source                 = "terraform-aws-modules/ec2-instance/aws"
  version                = "~> 2.0"

  name                   = "my-modules-demo"
  instance_count         = 2

  ami                    = data.aws_ami.amzlinux.id
  instance_type          = "t2.micro"
  key_name               = "terraform-key"
  monitoring             = true
  vpc_security_group_ids = ["sg-08b25c5a5bf489ffa"]  # Get Default VPC
Security Group ID and replace
  subnet_id              = "subnet-4ee95470" # Get one public subnet id
from default vpc and replace
  user_data               = file("apache-install.sh")

  tags = {
    Terraform   = "true"
    Environment = "dev"
  }
}
```