Terraform

Terraform Commands

* terraform init
* terraform validate
* terraform plan
* terraform apply
* terraform destroy

Terraform Init:

The terraform init command prepares a working directory for Terraform configuration files. It's the first command to run after writing a new Terraform configuration or cloning an existing one.

In order to prepare the working directory for use with Terraform, the terraform init command performs the following steps:

* Backend Initialization
* Child Module Installation
* Plugin Installation

Terraform validate:

The Terraform validate command checks the syntax of Terraform files and ensures the correct usage of attributes and values. It also validates the configuration based on the core syntax of Terraform and by checking all the providers in the code.

Terraform plan

The terraform plan command 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: Reads the current state of any already-existing remote objects to make sure that the Terraform state is up-to-date

Terraform apply

The Terraform apply command executes the actions proposed in a Terraform plan. It's used to deploy infrastructure and is typically run after Terraform init and Terraform plan

To make Terraform apply changes without asking for confirmation, you can use the --auto-approve

Terraform destroy

The terraform destroy command removes all resources created by the Terraform configuration. It is the opposite of the terraform apply command, which creates resources.

To destroy a specific resource, you can use the command terraform destroy -target='resource.name'

Terraform Top-Level Blocks

* Terraform Settings Block
* Provider Block
* Resource Block
* Input Variables Block
* Output Values Block
* Local Values Block
* Data Sources Block
* Modules Block

Terraform Settings Block

terraform {

  # Required Terraform Version

  required\_version = "~> 0.14"

  # Required Providers and their Versions

  required\_providers {

    aws = {

      source  = "hashicorp/aws"

      version = "~> 3.21" # Optional but recommended

    }

    random = {

      source = "hashicorp/random"

      version = "3.0.1"

    }

  }

  # Remote Backend for storing Terraform State in S3 bucket

  backend "s3" {

    bucket = "mybucket"

    key    = "path/to/my/key"

    region = "us-east-1"

  }

  # Experimental Features (Not required)

  experiments = [ example ]

  # Passing Metadata to Providers (Super Advanced - Terraform documentation says not needed in many cases)

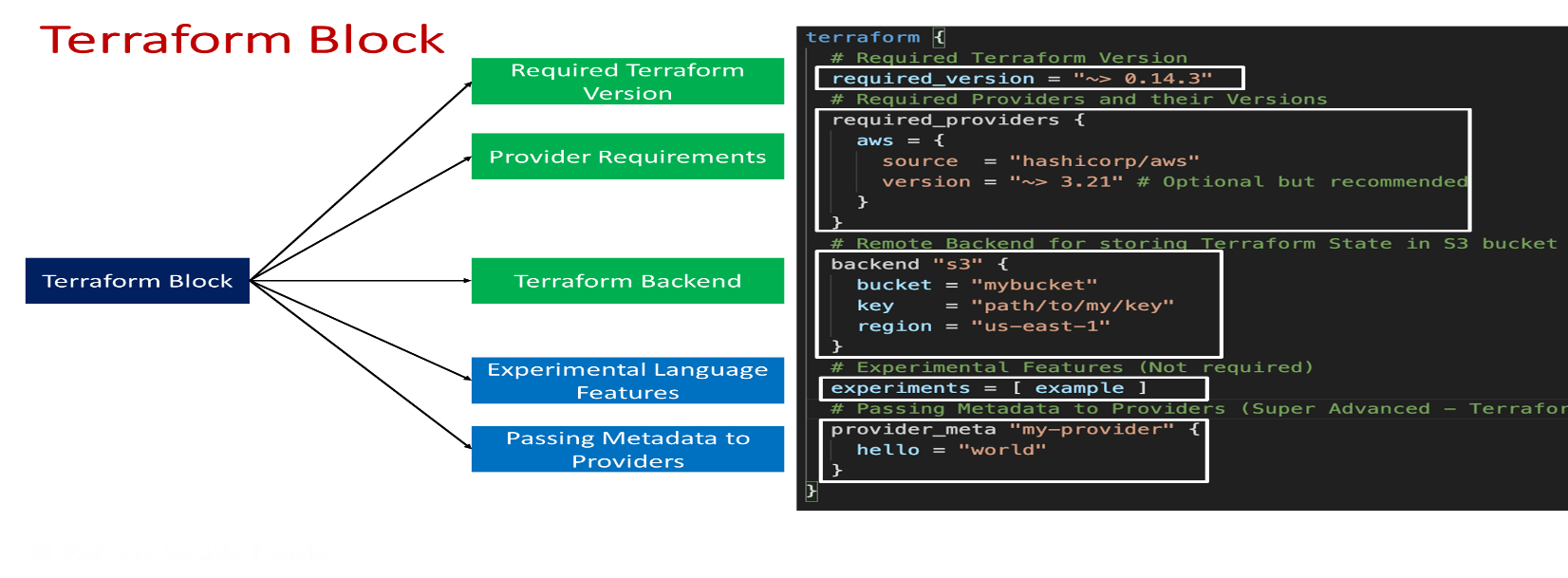
  provider\_meta "my-provider" {

    hello = "world"

  }

}

* Setting the required Terraform version.
* Configuring the backend for storing the state file.
* List Required Providers
* Defining experimental or optional features.
* Specifying variables used across multiple modules or configurations.



Provide Block

The provider block is used to configure and define the provider for a specific cloud or infrastructure program,It specifies details such as the provider name and version, authentication credentials, and other settings. By correctly configuring the provider block, you ensure that Terraform knows which provider to use and how to authenticate with it.

# Provider Block

provider "aws" {

  region     = "us-west-2"

  access\_key = "my-access-key"

  secret\_key = "my-secret-key"

}

Terraform Resources

It is used to declare and define the provider for a specific cloud or infrastructure program. Resources represent components such as virtual machines, networks, storage, databases, and other services. Each resource block specifies the resource type, name, and configuration parameters specific to that resource. By using resource blocks effectively, we can create and manage the desired infrastructure resources in a consistent and repeatable manner.

# VPC resource block

resource "aws\_vpc" "my\_vpc" {

  cidr\_block = "10.0.0.0/16"

  enable\_dns\_support = true

  enable\_dns\_hostnames = true

  tags = {

    Name = "MyVPC"

  }

}

# Subnet resource block within the VPC

resource "aws\_subnet" "my\_subnet" {

  vpc\_id                  = aws\_vpc.my\_vpc.id

  cidr\_block              = "10.0.1.0/24"

  availability\_zone       = "us-east-1a"

  tags = {

    Name = "MySubnet"

  }

}

# Security Group resource block

resource "aws\_security\_group" "my\_security\_group" {

  vpc\_id = aws\_vpc.my\_vpc.id

  egress {

    from\_port   = 0

    to\_port     = 0

    protocol    = "-1"

    cidr\_blocks = ["0.0.0.0/0"]

  }

  ingress {

    from\_port   = 22

    to\_port     = 22

    protocol    = "tcp"

    cidr\_blocks = ["0.0.0.0/0"]

  }

  tags = {

    Name = "MySecurityGroup"

Block Type

  }

Resource Type

}

Resource Local Name

# EC2 instance resource block

Block Label

resource "aws\_instance" "my\_instance" {

  ami             = "ami-0c55b159cbfafe1f0"

  instance\_type   = "t2.micro"

Argument Value

  subnet\_id       = aws\_subnet.my\_subnet.id

  key\_name        = "your-key-pair"

Argument Name

  vpc\_security\_group\_ids = [aws\_security\_group.my\_security\_group.id]

  tags = {

    Name = "MyEC2Instance"

  }

}