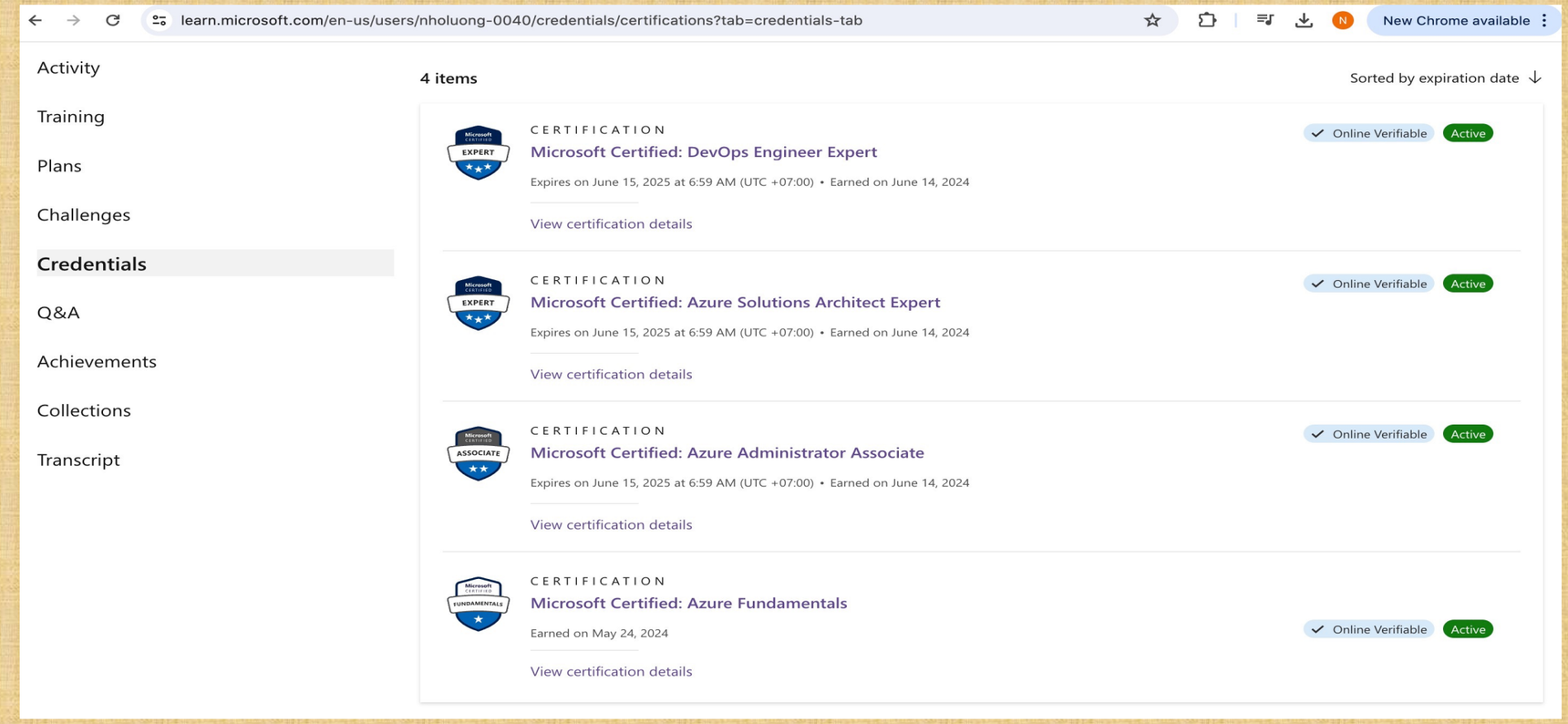
Terraform Associate on Azure Cloud

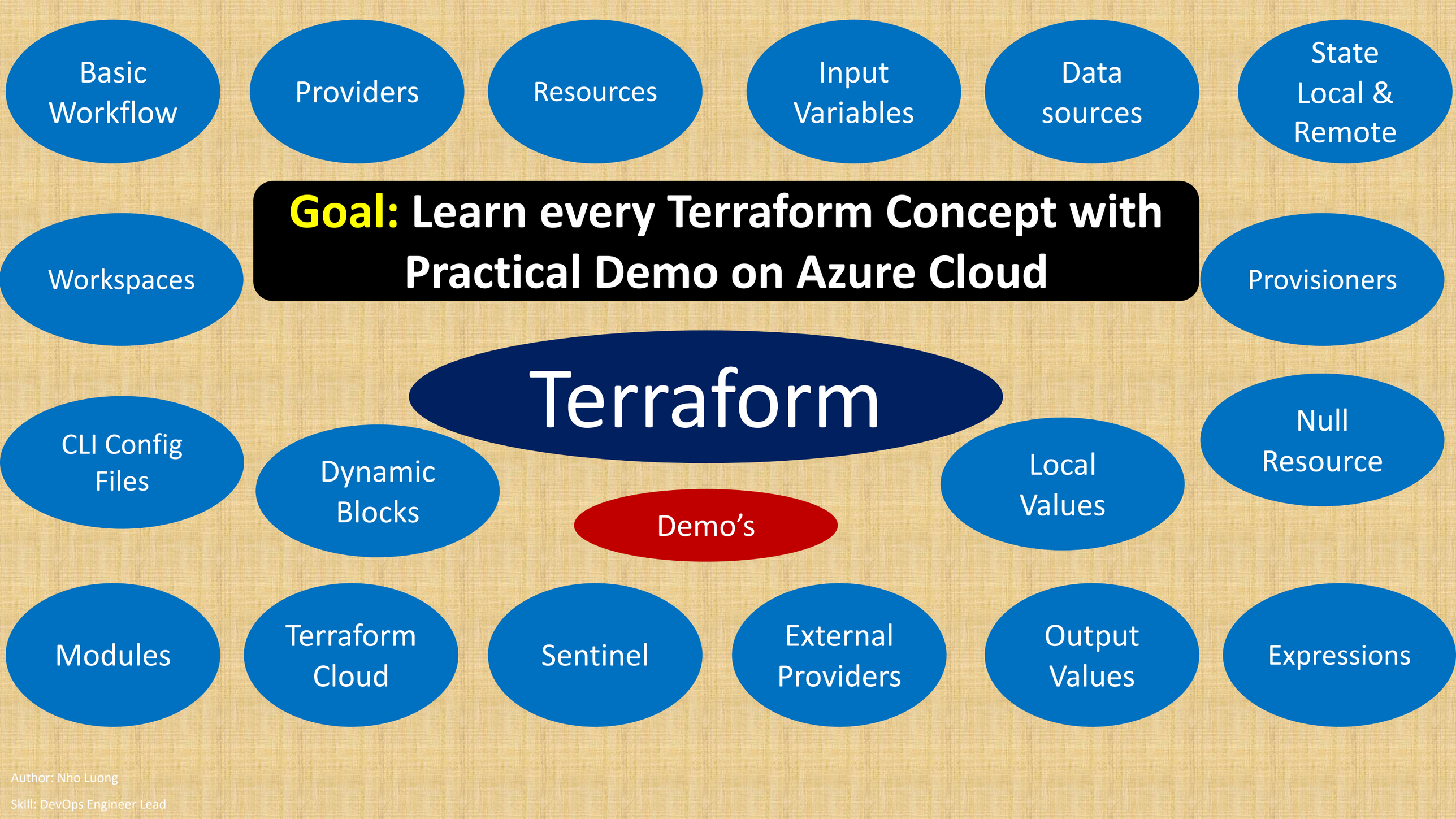
Author: Nho Luong

Skill: DevOps Engineer Lead



Author: Nho Luong

Skill: DevOps Engineer Lead


Basic Workflow
Providers
Resources
Input Variables
Data sources
State Local & Remote
Goal: Learn every Terraform Concept with Practical Demo on Azure Cloud
Workspaces
Provisioners
Terraform
CLI Config Files
Dynamic Blocks
Demo's
Local Values
Null Resource
Modules
Terraform Cloud
Sentinel
External Providers
Output Values
Expressions
Author: Nho Luong
Skill: DevOps Engineer Lead

Azure Cloud

Terraform Concepts

=

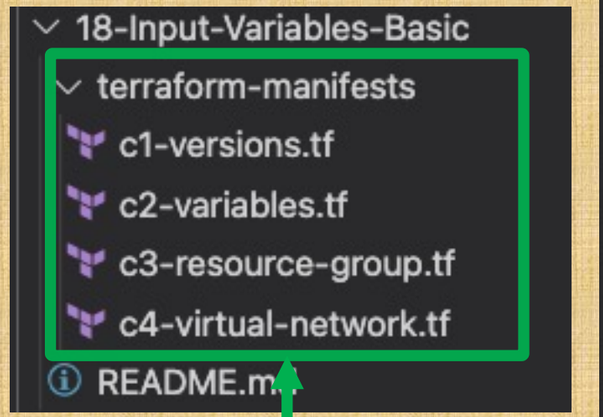
Terraform Demo’s

GitHub Step-by-Step Documentation

Author: Nho Luong

Skill: DevOps Engineer Lead

Github Step-by-Step Documentation with Practical Examples for each Concept



Terraform Configuration Files - well kept on Github

Author: Nho Luong

Skill: DevOps Engineer Lead

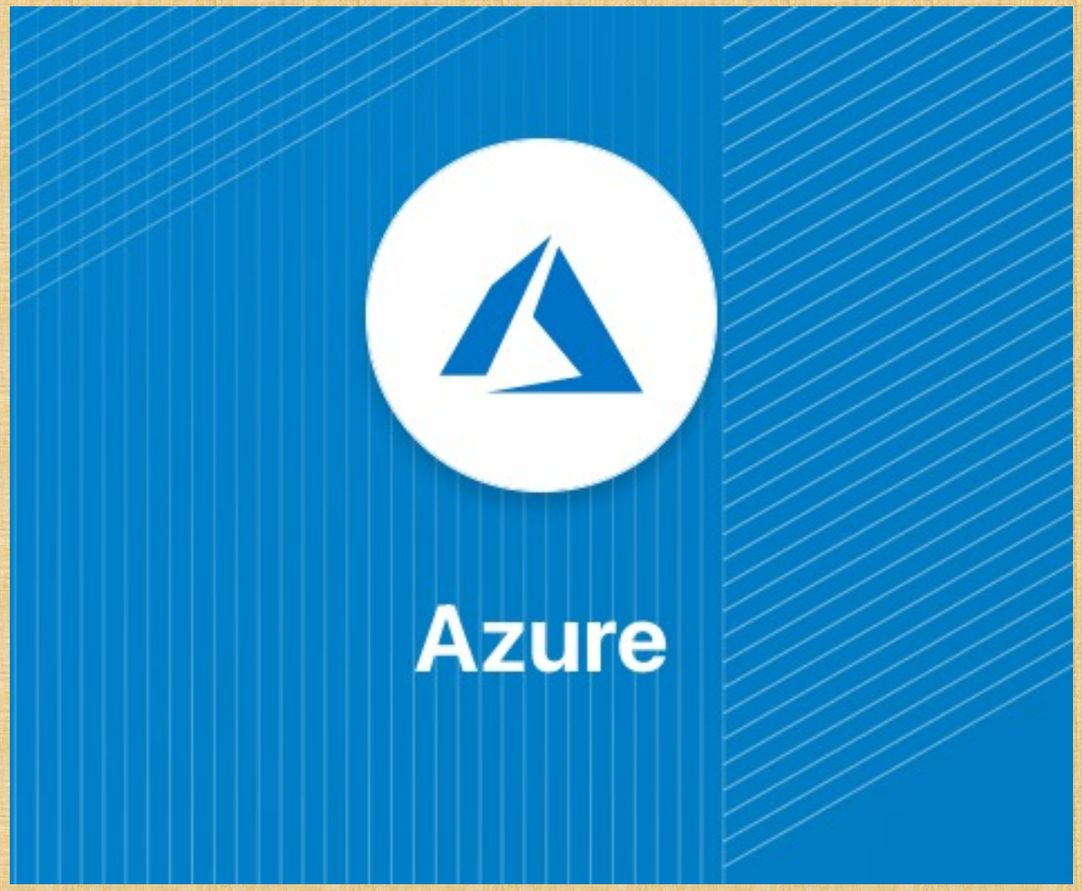
Github Step-by-Step Documentation with Practical Examples for each Concept

- 31-Input-Variables-Collection-Type-Sets
- 32-Output-Values-Basics
- 33-Output-Values-with-count-and-Splat-Expression
- 34-Output-Values-with-for_each-and-for-loops
- 35-Terraform-Local-Values
- 36-Terraform-Conditional-Expressions
- 37-Terraform-Datasources
- 38-Terraform-Remote-State-Storage-and-Locking
- 39-Terraform-Remote-State-Datasource
- 40-Terraform-State-Commands
- 41-Terraform-apply-refresh-only-command
- 42-Terraform-Workspaces-with-Local-Backends
- 43-Terraform-Workspaces-with-Remote-Backends
- 44-Terraform-File-Provisioner
- 45-Terraform-remote-exec-provisioner
- 46-Terraform-local-exec-provisioner
- 47-Terraform-Null-Resource
- 48-Terraform-State-Import
- 49-Terraform-Modules-use-Public-Module
- 50-Terraform-Azure-Static-Website
- 51-Terraform-Modules-Build-Local-Module
- 52-Terraform-Module-Publish-to-Public-Registry
- 53-Terraform-Module-Sources
- 54-Terraform-Cloud-Github-Integration
- 55-Share-Modules-in-Private-Module-Registry
- 56-Terraform-Cloud-CLI-Driven-Workflow
- 57-Migrate-State-to-Terraform-Cloud
- 58-Terraform-Cloud-and-Sentinel-Policies
- 59-Terraform-Foundational-Policies-using-Sentinel
- 60-Terraform-Dynamic-Blocks
- 61-Terraform-Debug
- 62-Terraform-Override-Files
- 63-Terraform-External-Provider-and-Datasource-Demo-1
- 64-Terraform-External-Provider-and-Datasource-Demo-2
- 65-Terraform-CLI-Config-File-MacOS-and-Linux
- 66-Terraform-CLI-Config-File-WindowsOS
- 67-Terraform-Manage-Providers
- 68-Exam-Preparation
- 69-Exam-Registration

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Providers and Azure Resources used



- Azure Resource Group
- Azure Virtual Network
- Azure Network Interfaces
- Azure Linux Virtual Machine
- Azure Public IP
- Azure Storage Account

Core focus will be on mastering Terraform Concepts with Sample Demo’s

Author: Nho Luong

Skill: DevOps Engineer Lead

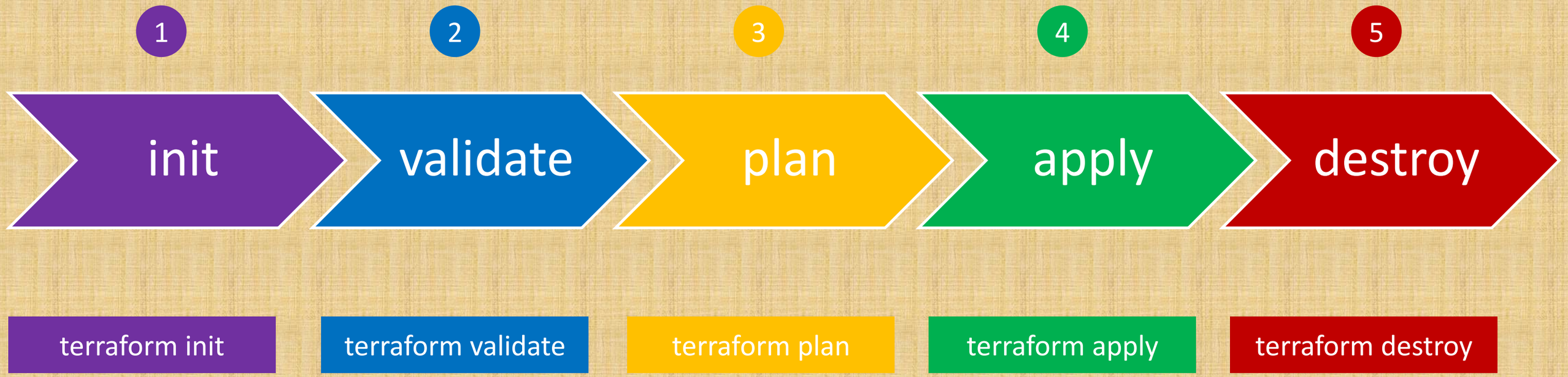
GitHub Repositories

Repository Used For	Repository URL
Course Main Repository with step-by-step documentation	https://github.com/nholuongut/terraform-associate-on-azure
Terraform Cloud Demo	https://github.com/nholuongut/terraform-cloud-azure-demo1
Terraform Module Published to Public Registry	https://github.com/nholuongut/terraform-azurerm-staticwebsitepublic
Terraform Module Published to Terraform Cloud Private Registry	https://github.com/nholuongut/terraform-azurerm-staticwebsiteprivate
Terraform Sentinel Policies Demo	https://github.com/nholuongut/terraform-sentinel-policies-azure

Author: Nho Luong

Skill: DevOps Engineer Lead

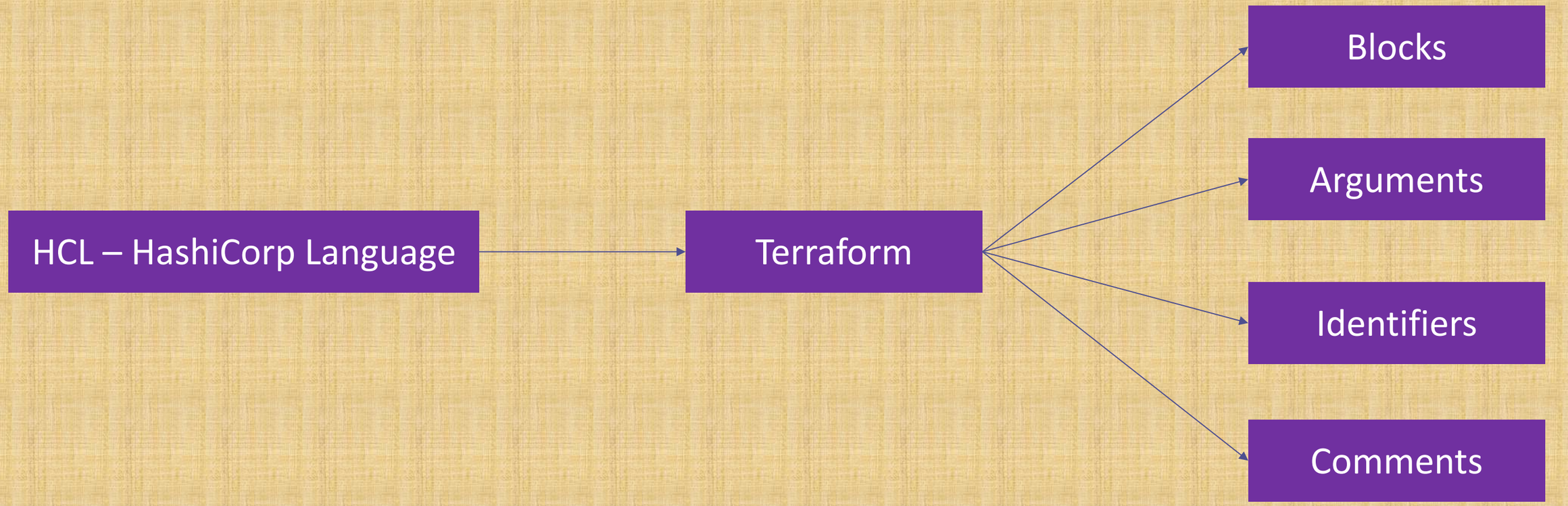
Terraform Workflow



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Language Basics – Configuration Syntax



Author: Nho Luong

Skill: DevOps Engineer Lead

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.

Terraform Top-Level Blocks

Most of **Terraform's features** are implemented as **top-level** blocks.

Fundamental Blocks

- Terraform Block
- Providers Block
- Resources Block

Variable Blocks

- Input Variables Block
- Output Values Block
- Local Values Block

Calling / Referencing Blocks

- Data Sources Block
- Modules Block

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Providers

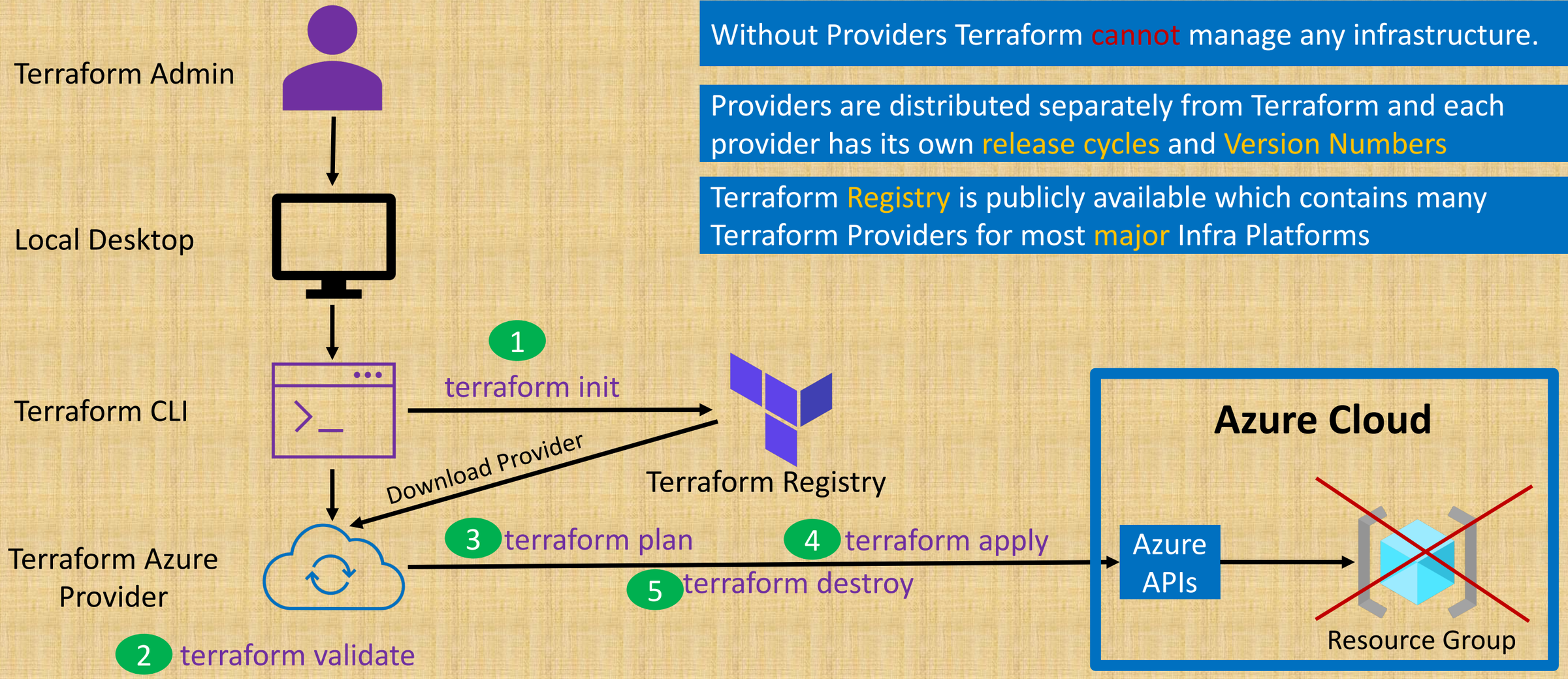
Providers are HEART of Terraform

Every Resource Type (example: Azure Resource Group), is implemented by a Provider

Without Providers Terraform cannot manage any infrastructure.

Providers are distributed separately from Terraform and each provider has its own release cycles and Version Numbers

Terraform Registry is publicly available which contains many Terraform Providers for most major Infra Platforms



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Language Basics

Terraform
Resource Syntax

Terraform
Resource Behavior

Terraform
State

Demos

Terraform
Resources

Resource
Meta-Argument
count

Resource
Meta-Argument
depends_on

Resource
Meta-Argument
for_each

Resource
Meta-Argument
lifecycle

Author: Nho Luong

Skill: DevOps Engineer Lead

Input variables serve as parameters for a Terraform module, allowing aspects of the module to be customized without altering the module's own source code, and allowing modules to be shared between different configurations.

Terraform Input Variables

Demos

1. Input Variables - Basics
2. Provide Input Variables when prompted during terraform plan or apply
3. Override default variable values using CLI argument -var
4. Override default variable values using Environment Variables (TF_var_aa)
5. Provide Input Variables using terraform.tfvars files
6. Provide Input Variables using <any-name>.tfvars file with CLI argument -var-file
7. Provide Input Variables using somefilename.auto.tfvars files
8. Implement complex type constructors like List & Map in Input Variables
9. Implement Custom Validation Rules in Variables
10. Protect Sensitive Input Variables
11. Input Variables using Structural Type: Object
12. Input Variables using Structural Type: tuple
13. Input Variables using Collection Type: set

Author: Nho Luong

Skill: DevOps Engineer Lead

Output Values – Demos

Demo

Demo

Demo

Basics

Count and Splat Expression

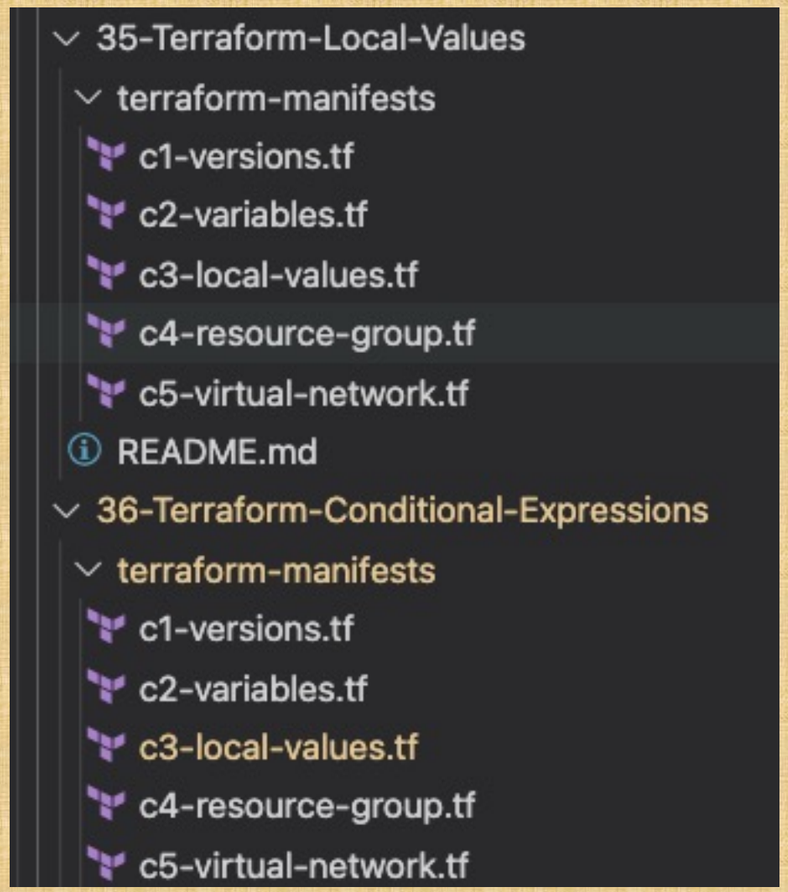
for_each and for loops

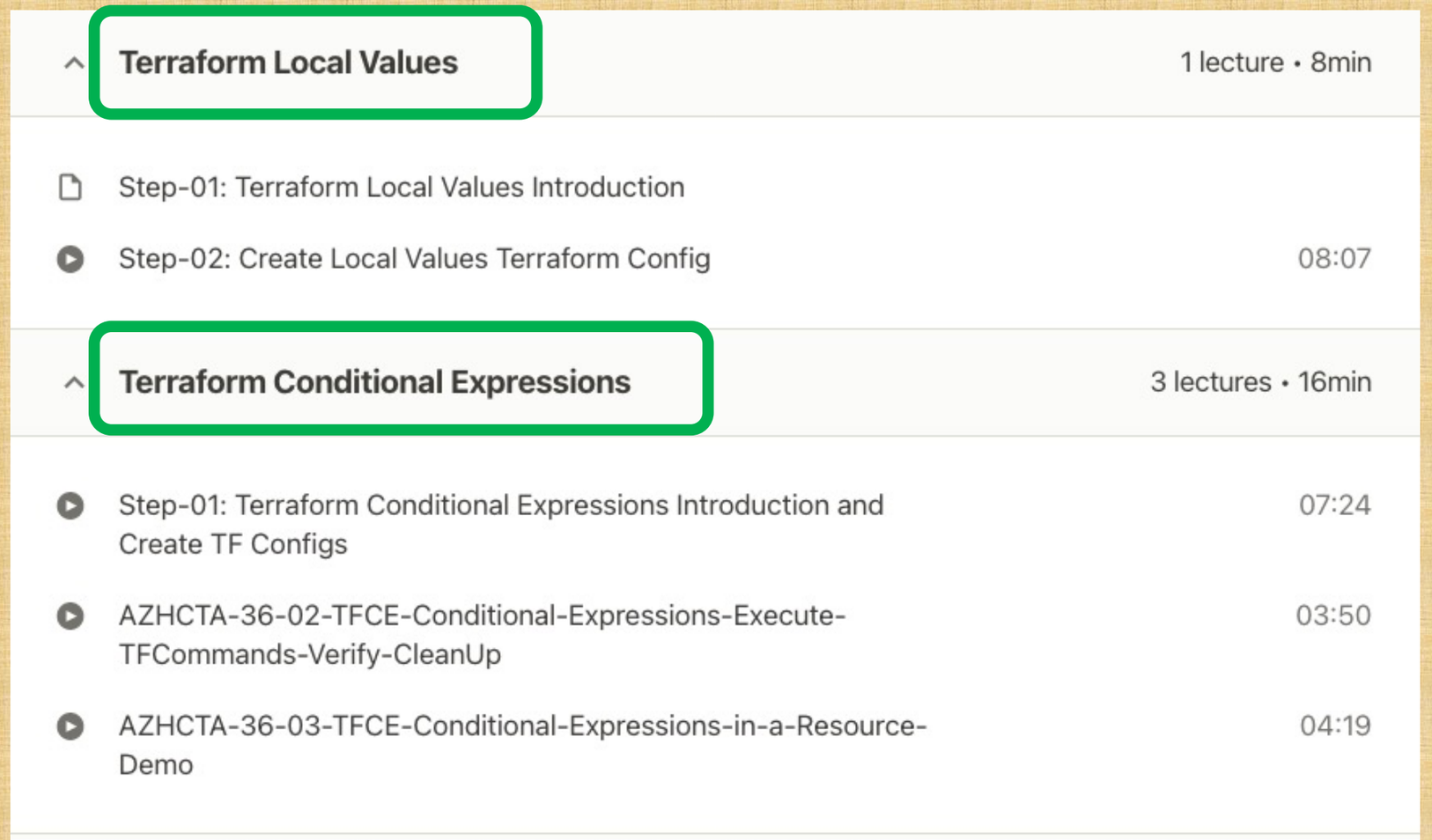
Over the process master the for loops in Terraform with Lists and Maps

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Local Values - Demos





Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform State

- Terraform State Basics
- Terraform Desired and Current States
- Terraform Remote State Storage and Locking
- Terraform State Commands
- Terraform Remote State Datasource

Demos

Author: Nho Luong

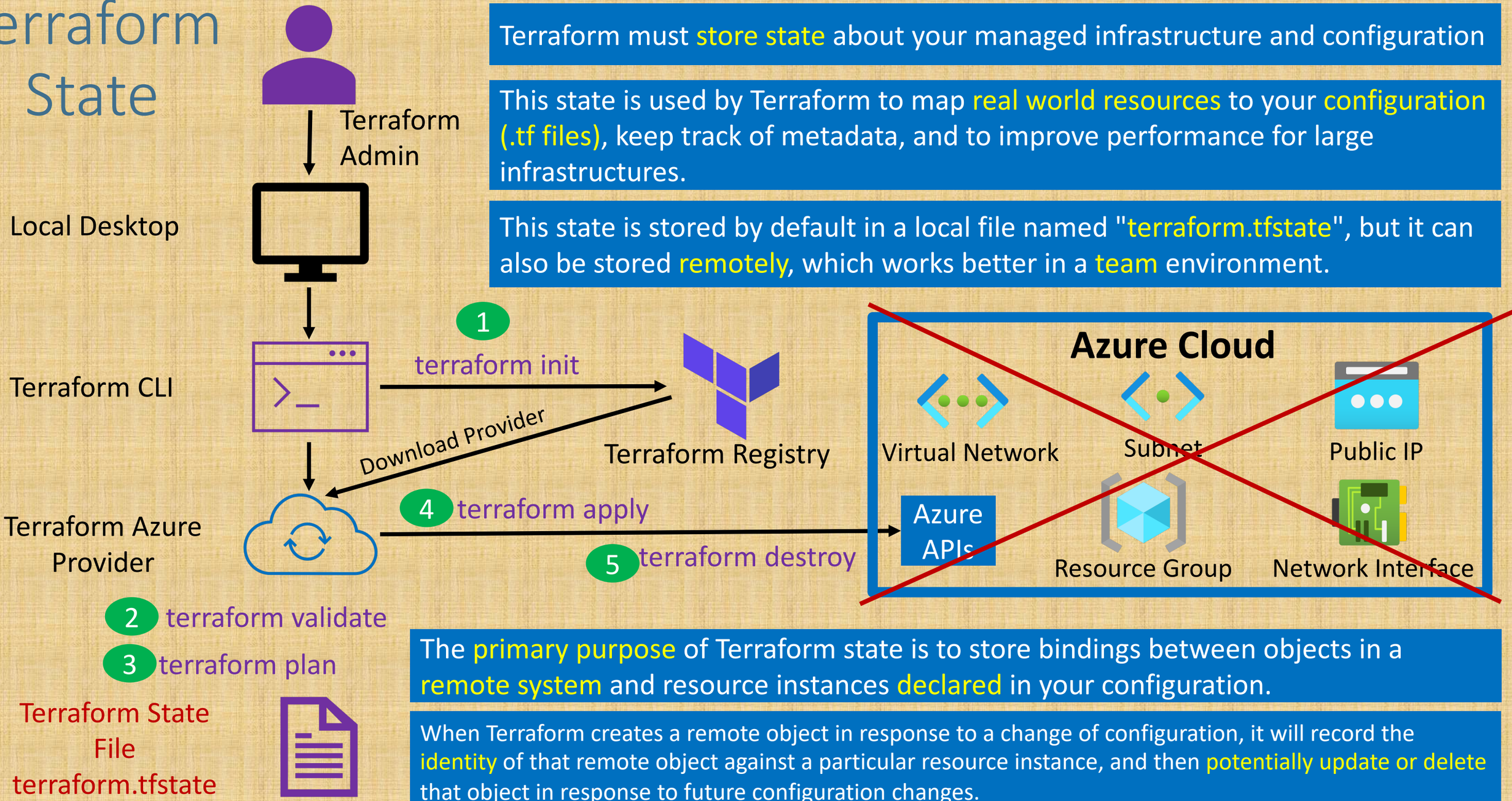
Skill: DevOps Engineer Lead

Terraform State

Terraform must store state about your managed infrastructure and configuration

This state is used by Terraform to map real world resources to your configuration (.tf files), keep track of metadata, and to improve performance for large infrastructures.

This state is stored by default in a local file named "terraform.tfstate", but it can also be stored remotely, which works better in a team environment.



The primary purpose of Terraform state is to store bindings between objects in a remote system and resource instances declared in your configuration.

When Terraform creates a remote object in response to a change of configuration, it will record the identity of that remote object against a particular resource instance, and then potentially update or delete that object in response to future configuration changes.

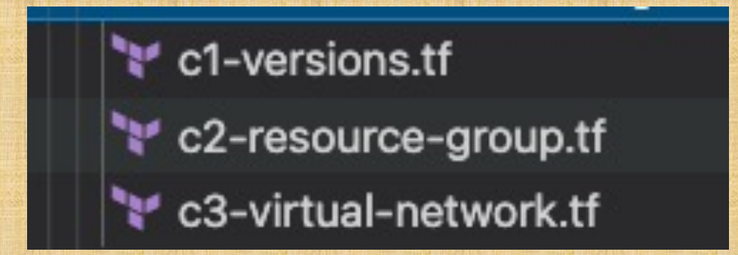
Author: Nho Luong

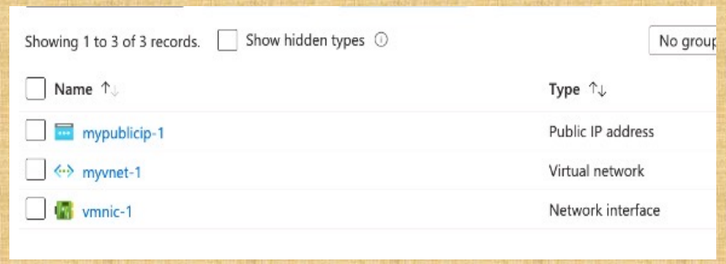
Skill: DevOps Engineer Lead

Desired & Current Terraform States

Terraform Configuration Files

Real World Resources





Desired State

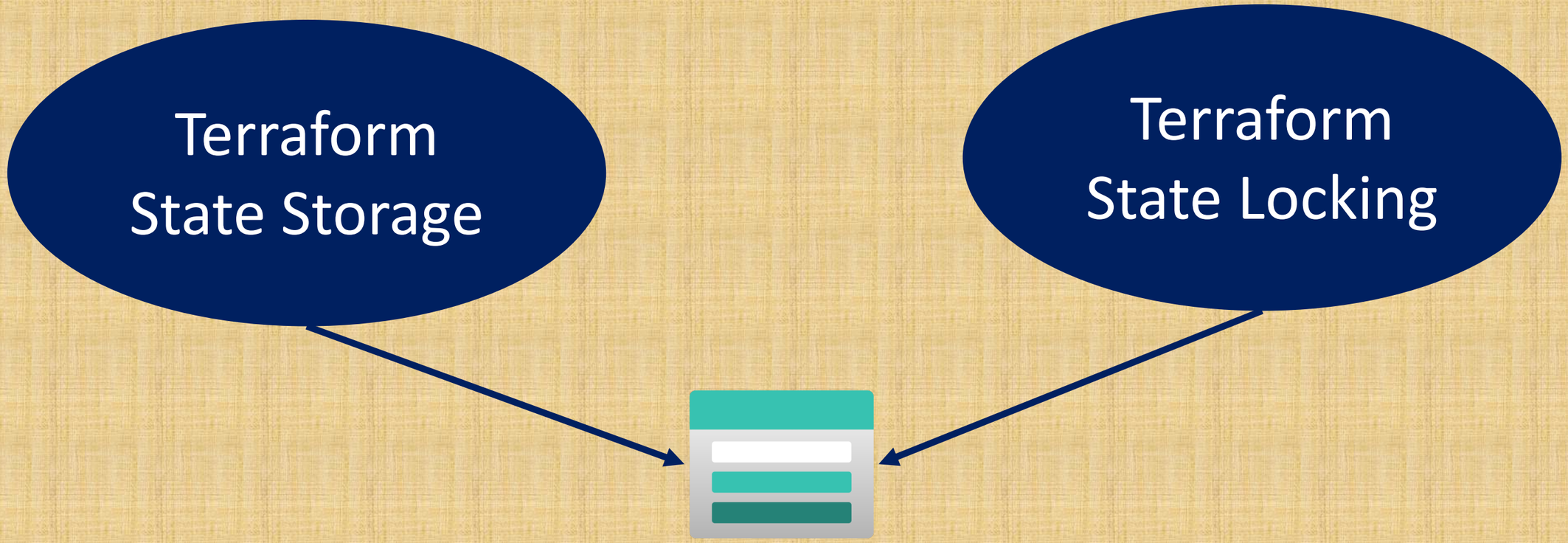
Current State

Author: Nho Luong

Skill: DevOps Engineer Lead

What is Terraform Backend ?

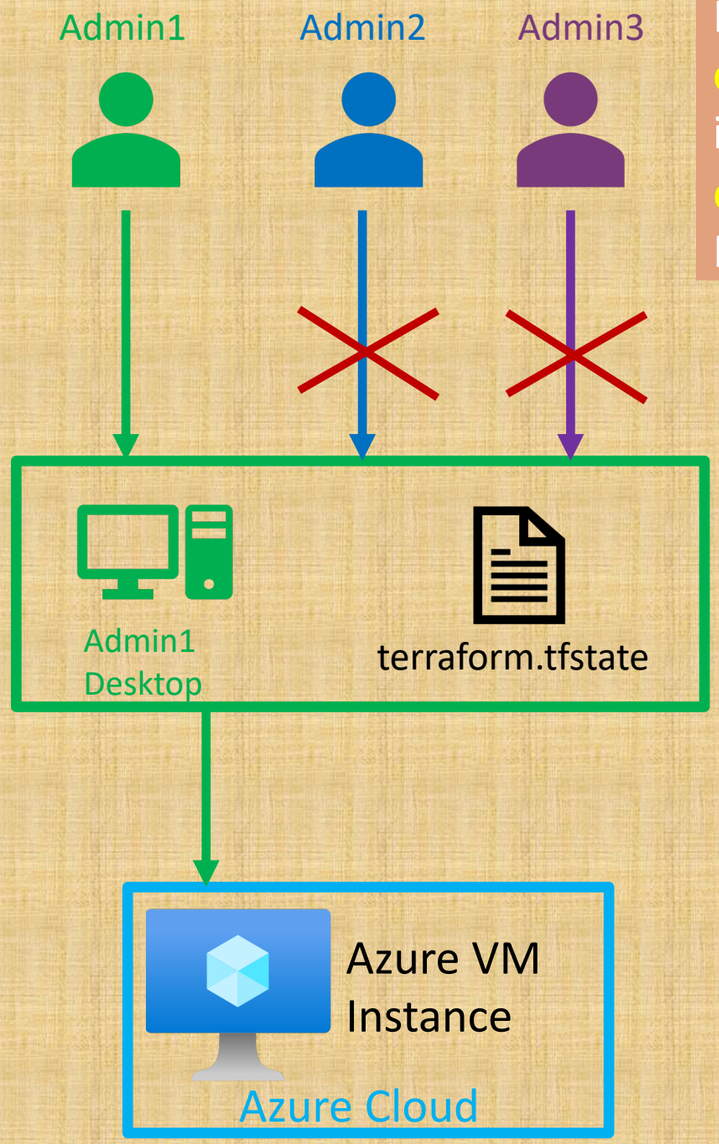
Backends are responsible for storing state and providing an API for state locking.



Author: Nho Luong

Skill: DevOps Engineer Lead

Local State File



Multiple Team members cannot update the infrastructure as they don't have access to State File

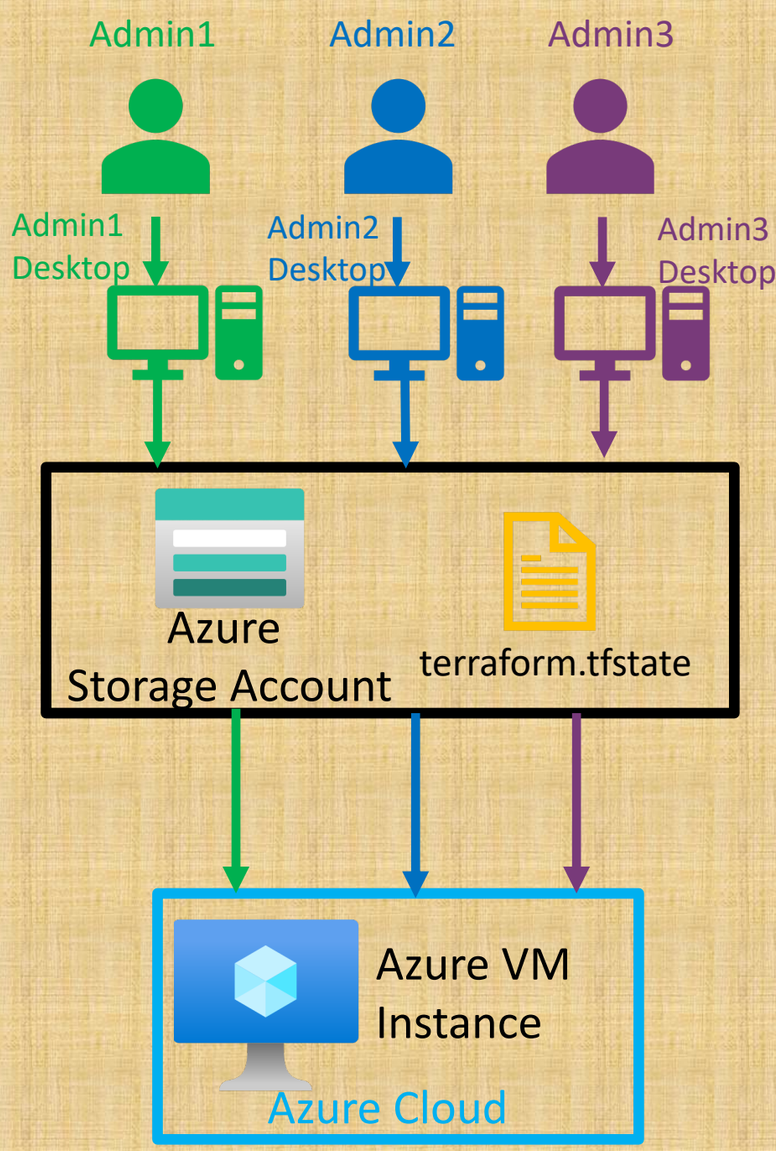
This means we need store the state file in a shared location.

Remote State File

Using Terraform Backend concept we can use Azure SA as the shared storage for State Files

If two team members are running Terraform at the same time, you may run into race conditions as multiple Terraform processes make concurrent updates to the state files, leading to conflicts, data loss, and state file corruption.

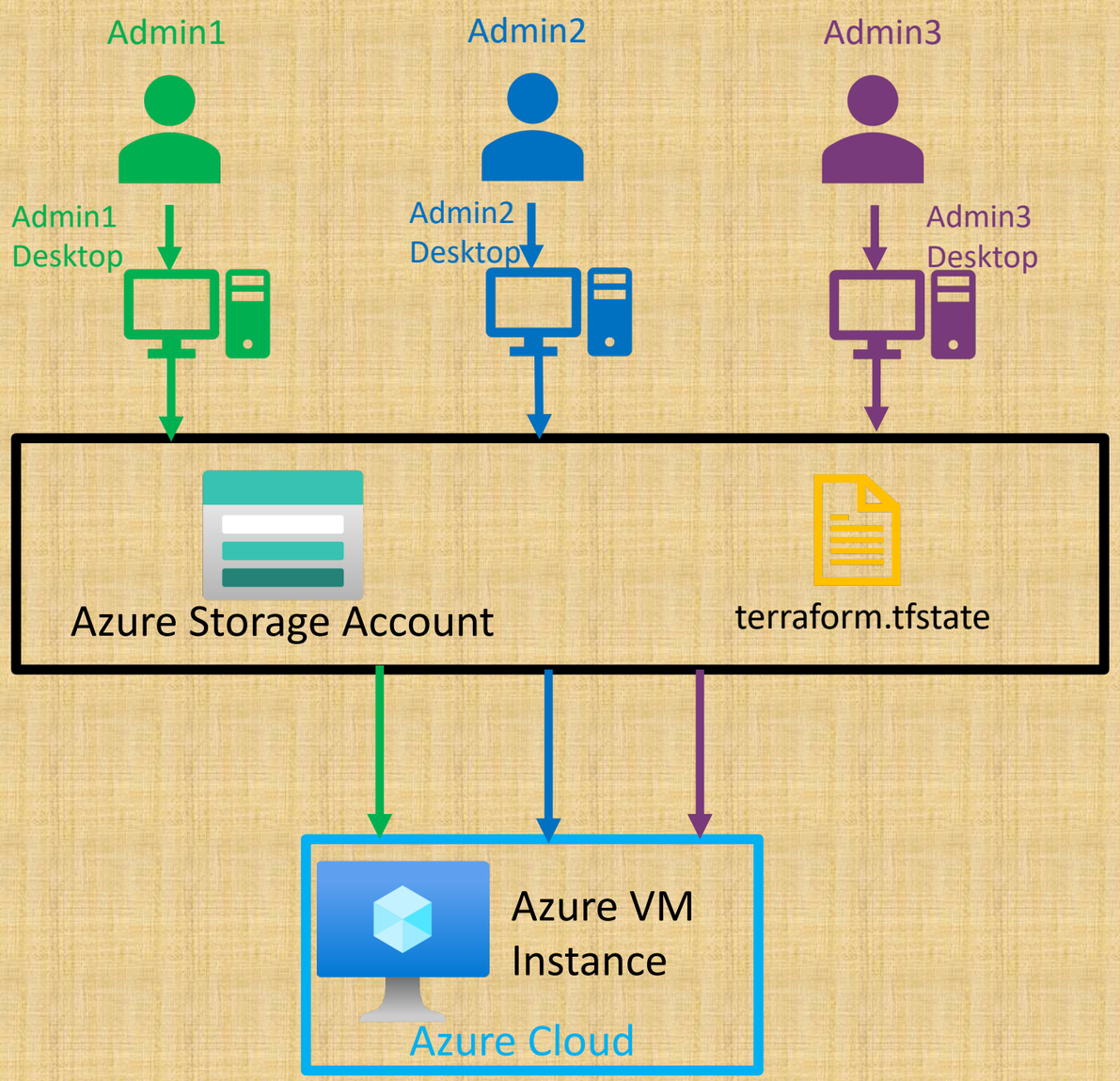
State Locking



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Remote State File with State Locking



Not all backends support State Locking. Azure Storage Account supports State Locking

State locking happens automatically on all operations that could write state.

If state locking fails, Terraform will not continue.

You can disable state locking for most commands with the -lock flag but it is not recommended.

If acquiring the lock is taking longer than expected, Terraform will output a status message.

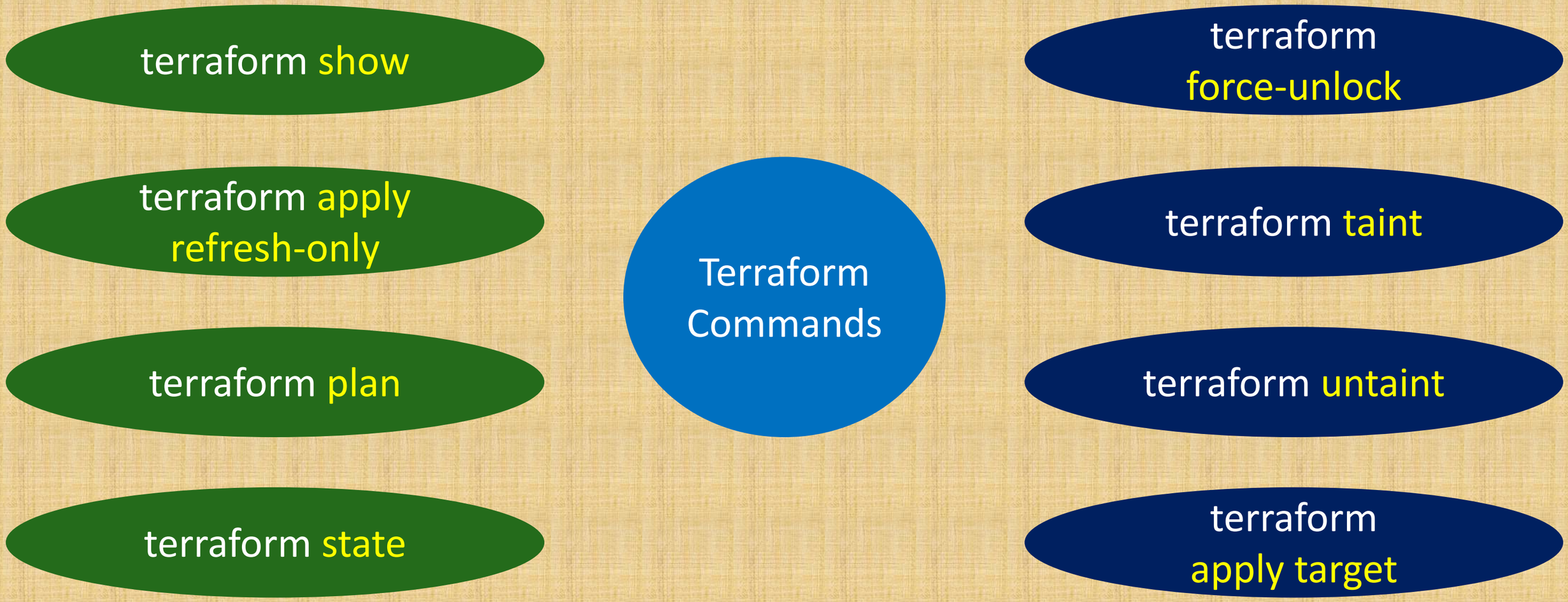
If Terraform doesn't output a message, state locking is still occurring if your backend supports it.

Terraform has a force-unlock command to manually unlock the state if unlocking failed.

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Commands – State Perspective

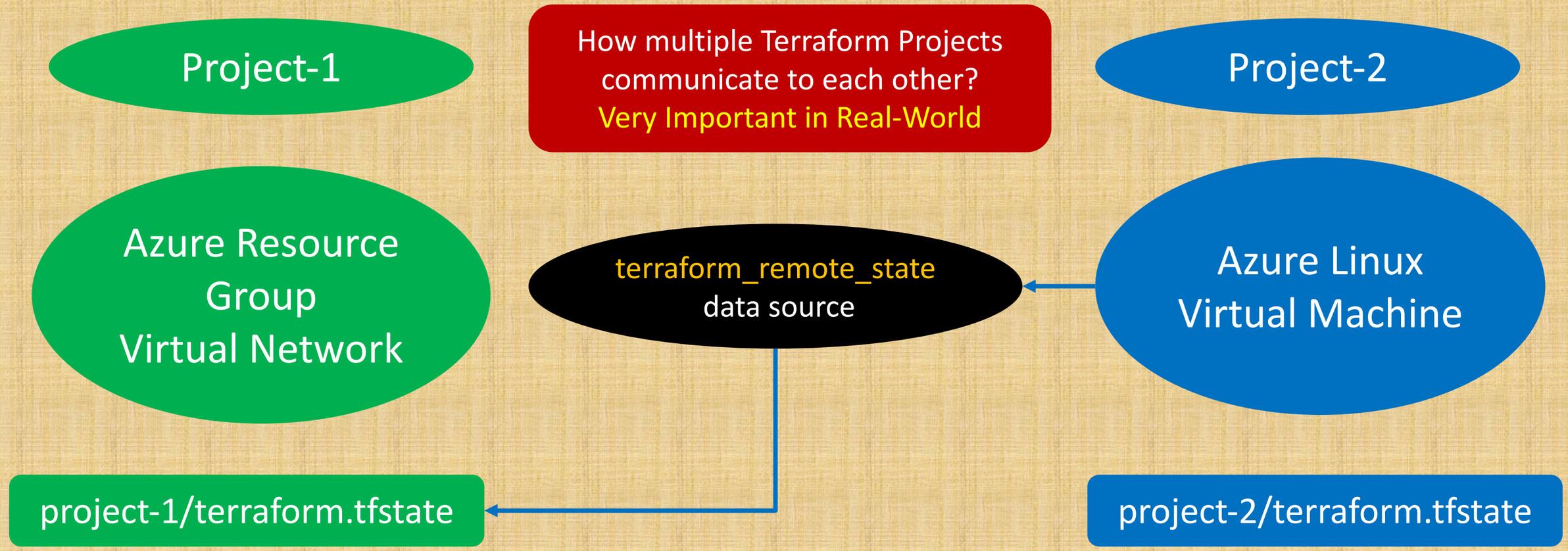


Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Remote State Datasource

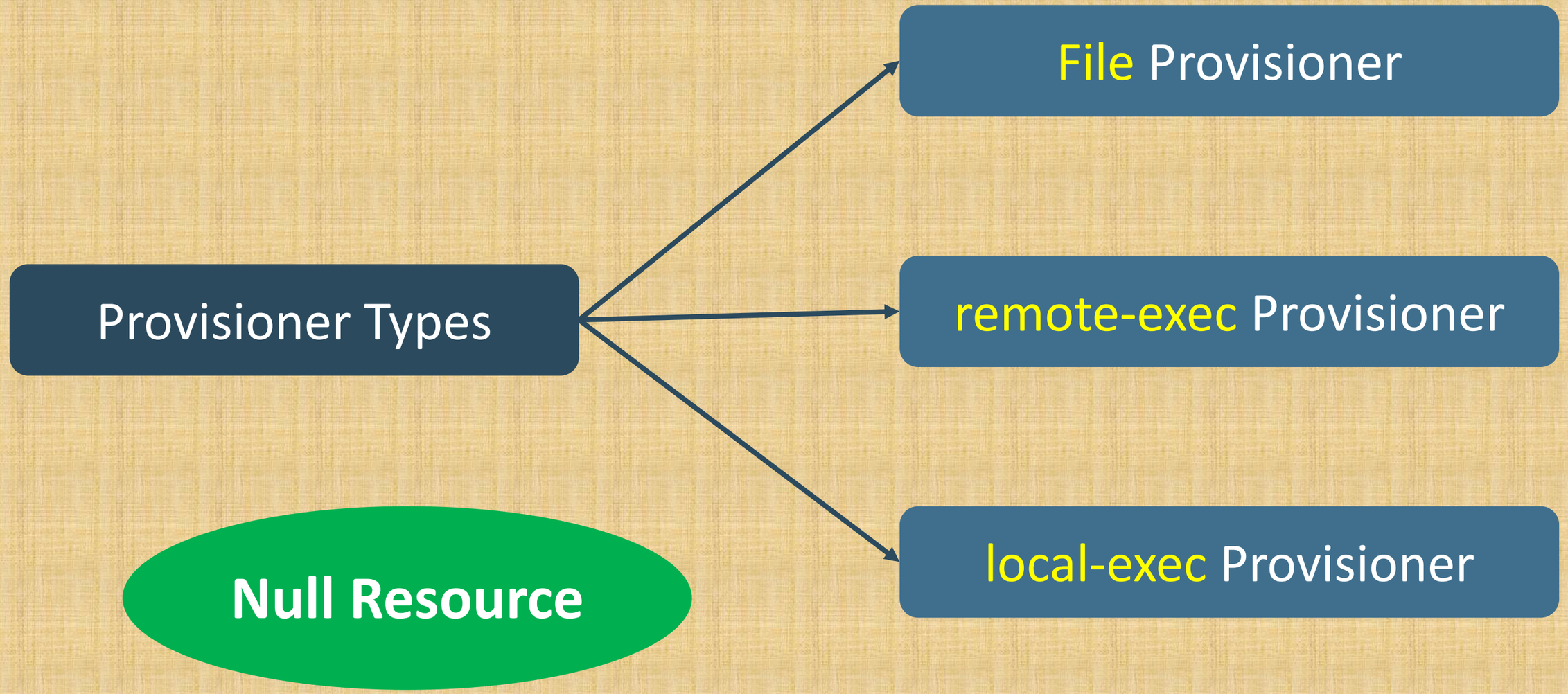
The terraform_remote_state data source retrieves the root module output values from some other Terraform configuration, using the latest state snapshot from the remote backend.



Author: Nho Luong

Skill: DevOps Engineer Lead

Types of Provisioners



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Modules

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Registry – Use Publicly Available Modules

The Terraform Registry hosts a broad collection of publicly available Terraform modules for configuring many kinds of common infrastructure.

Demo

These modules are free to use, and Terraform can download them automatically if you specify the appropriate source and version in a module call block.

```
# Create Virtual Network and Subnets using Terraform Public Registry Mo
module "vnet" {
  source  = "Azure/vnet/azurerm"
  version = "2.5.0" # No versions constraints for production grade imp
  vnet_name = local.vnet_name
  resource_group_name = azurerm_resource_group.myrg.name
  address_space       = ["10.0.0.0/16"]
  subnet_prefixes     = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  subnet_names        = ["subnet1", "subnet2", "subnet3"]
  subnet_service_endpoints = {
    subnet2 = ["Microsoft.Storage", "Microsoft.Sql"],
    subnet3 = ["Microsoft.AzureActiveDirectory"]
  }
  tags = {
    environment = "dev"
    costcenter  = "it"
  }
  depends_on = [azurerm_resource_group.myrg]
}
```

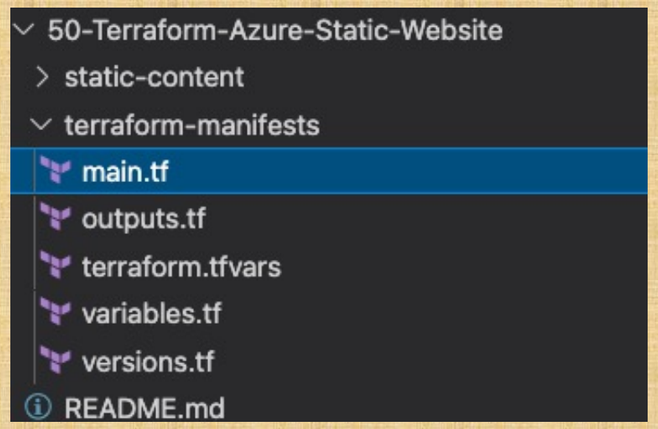
Author: Nho Luong

Skill: DevOps Engineer Lead

Build manually and then automate using Terraform

1. Build a Static Website using Azure manually using Azure Portal.
2. Automate it using Terraform Resources

Demo




```
# Create Azure Storage account
resource "azurerm_storage_account" "storage_account" {
  name                     = "${var.storage_account_name}${random_string.myrandom.id}"
  resource_group_name = azurerm_resource_group.resource_group.name

  location                 = var.location
  account_tier             = var.storage_account_tier
  account_replication_type = var.storage_account_replication_type
  account_kind             = var.storage_account_kind

  static_website {
    index_document     = var.static_website_index_document
    error_404_document = var.static_website_error_404_document
  }
}
```

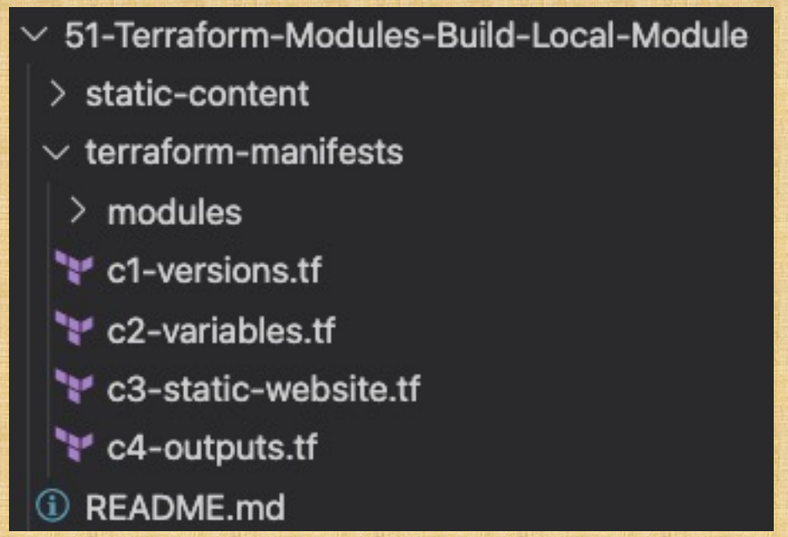
Author: Nho Luong

Skill: DevOps Engineer Lead

Build a Local Terraform Module

Demo

Build a Local Terraform Module and call it from Root Module and Test




```hcl
# Call our Custom Terraform Module which we built earlier
module "azure_static_website" {
  source = "./modules/azure-static-website" # Mandatory

  # Resource Group
  location                        = "eastus"
  resource_group_name             = "myrg1"

  # Storage Account
  storage_account_name            = "staticwebsite"
  storage_account_tier            = "Standard"
  storage_account_replication_type  = "LRS"
  storage_account_kind            = "StorageV2"
  static_website_index_document     = "index.html"
  static_website_error_404_document = "error.html"
}
```

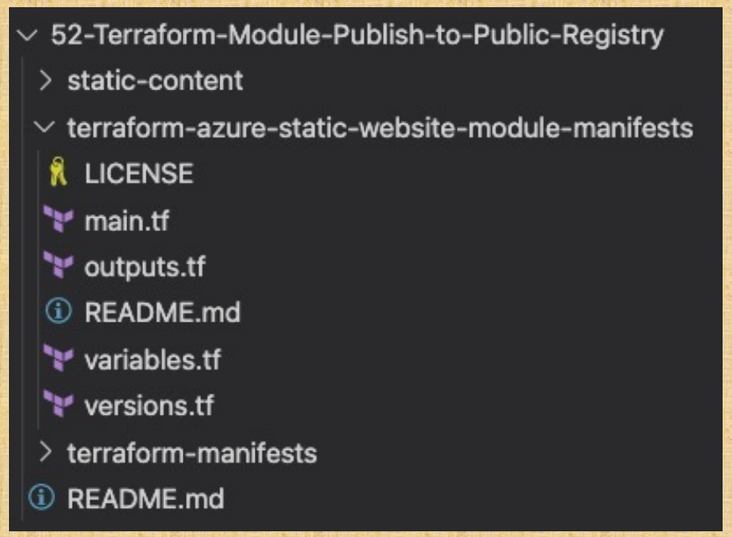
Author: Nho Luong

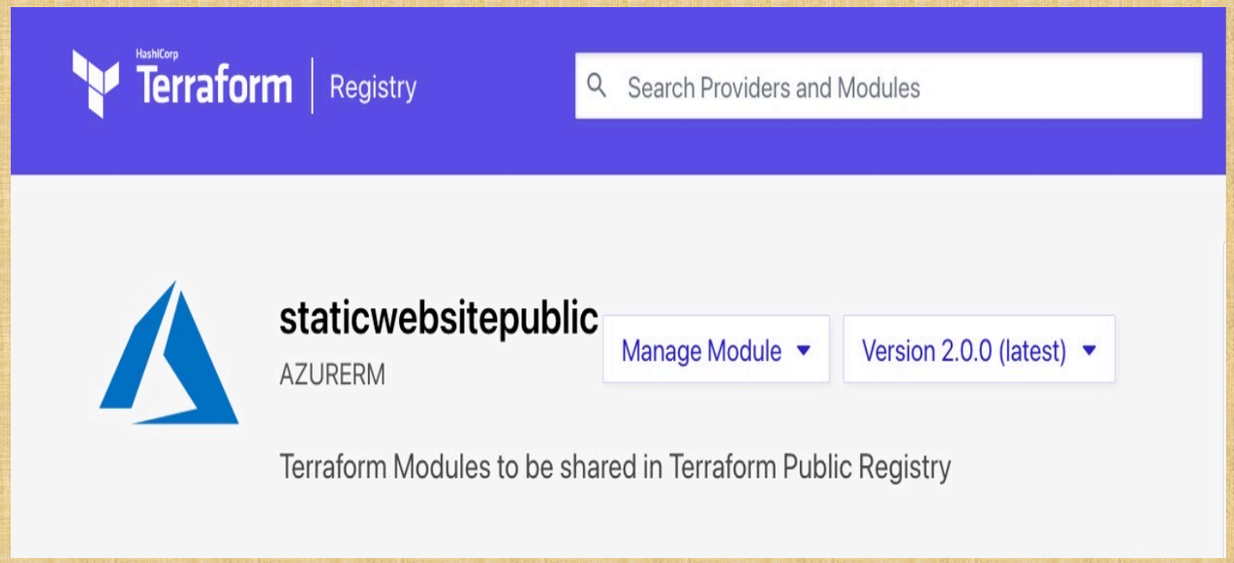
Skill: DevOps Engineer Lead

Publish to Public Terraform Registry

Demo

Publish to Public Terraform Registry and Access it from Public Registry and Test.





Author: Nho Luong

Skill: DevOps Engineer Lead

Use various Module Sources

Demo

In addition to Terraform Public and Private Registry use various module sources.

```
# Call our Custom Terraform Module which we built earlier
module "azure_static_website" {

  # Terraform Local Module (Local Paths)
  #source = "./modules/azure-static-website"
```

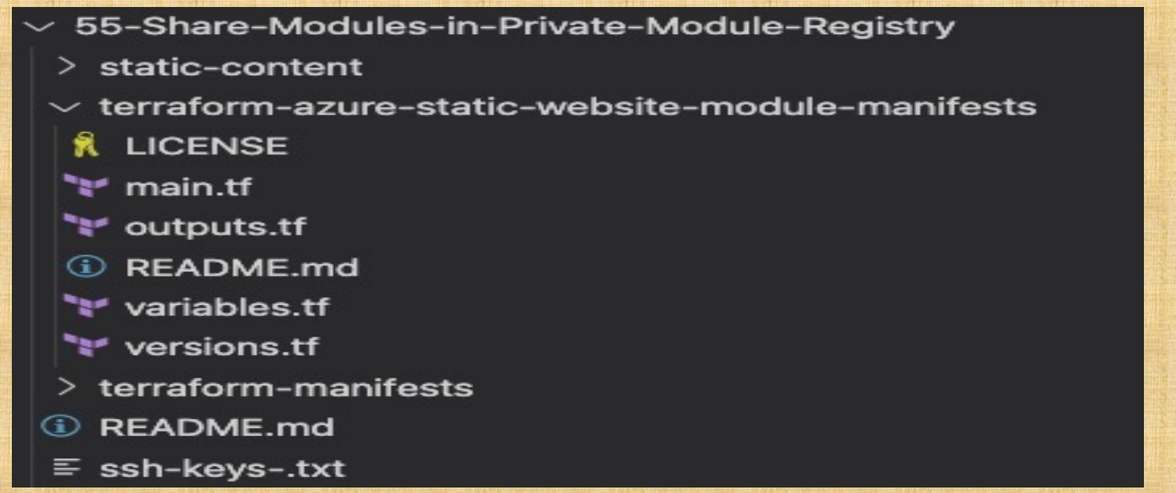
Author: Nho Luong

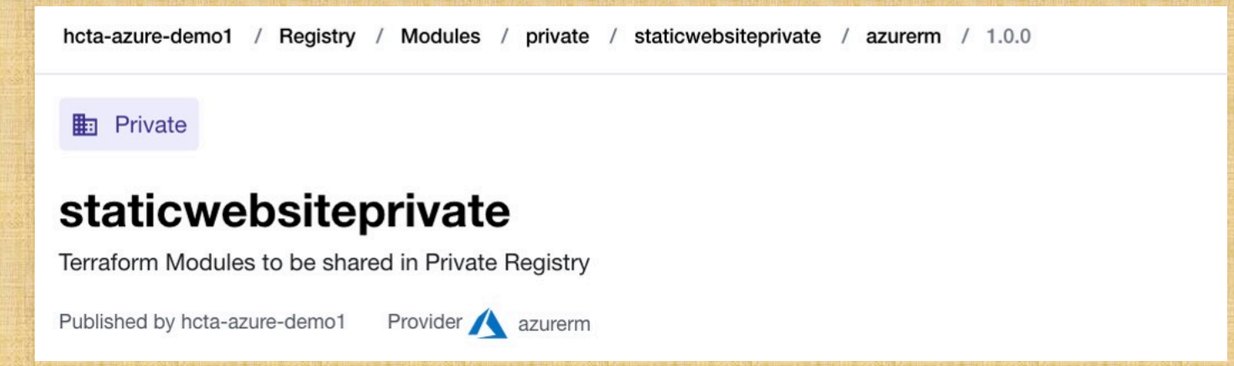
Skill: DevOps Engineer Lead

Private Module Registry in Terraform Cloud & Enterprise

Members of your organization might produce modules specifically crafted for your own infrastructure needs.

Terraform Cloud and Terraform Enterprise both include a private module registry for sharing modules internally within your organization.





Demo

Author: Nho Luong

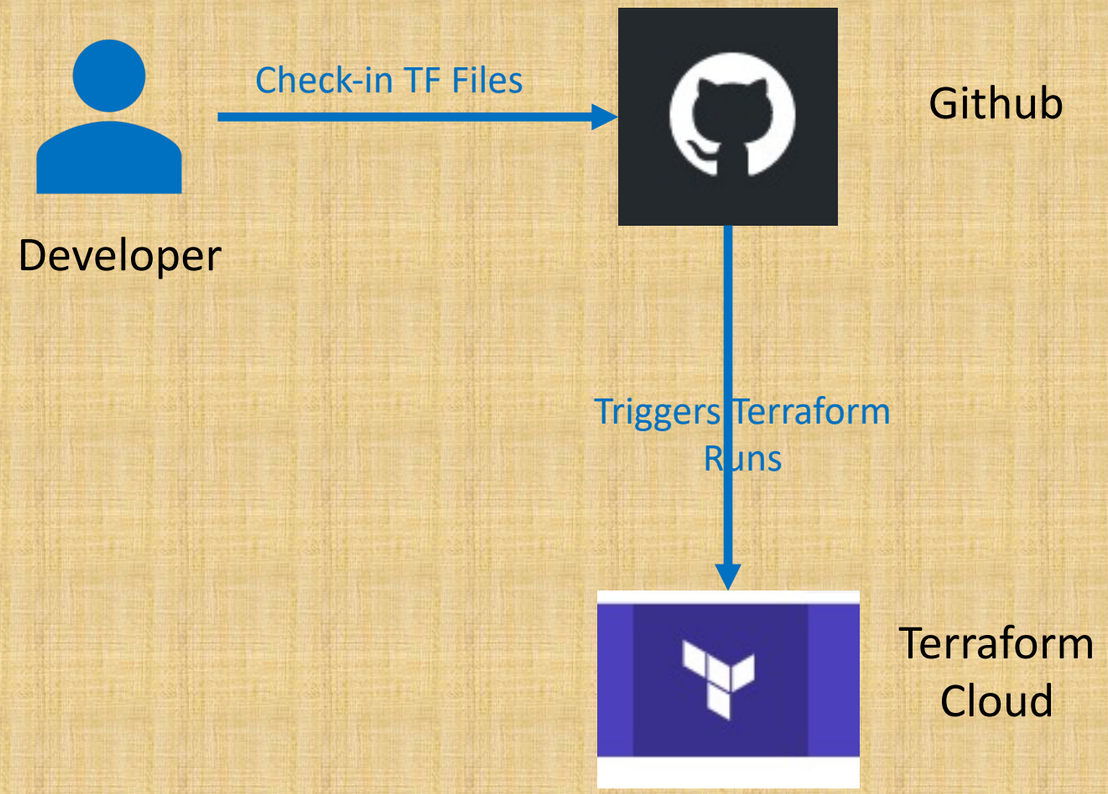
Skill: DevOps Engineer Lead

Terraform Cloud & Sentinel Policies

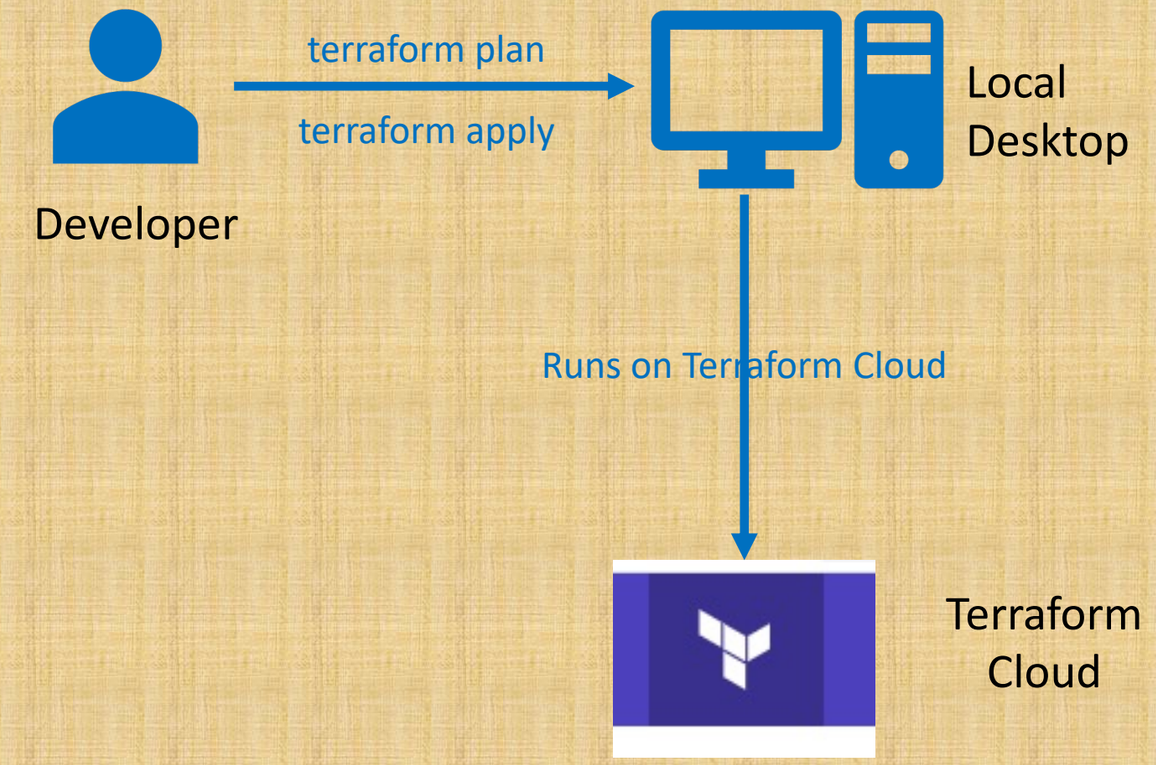
Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform VCS-Driven Workflow



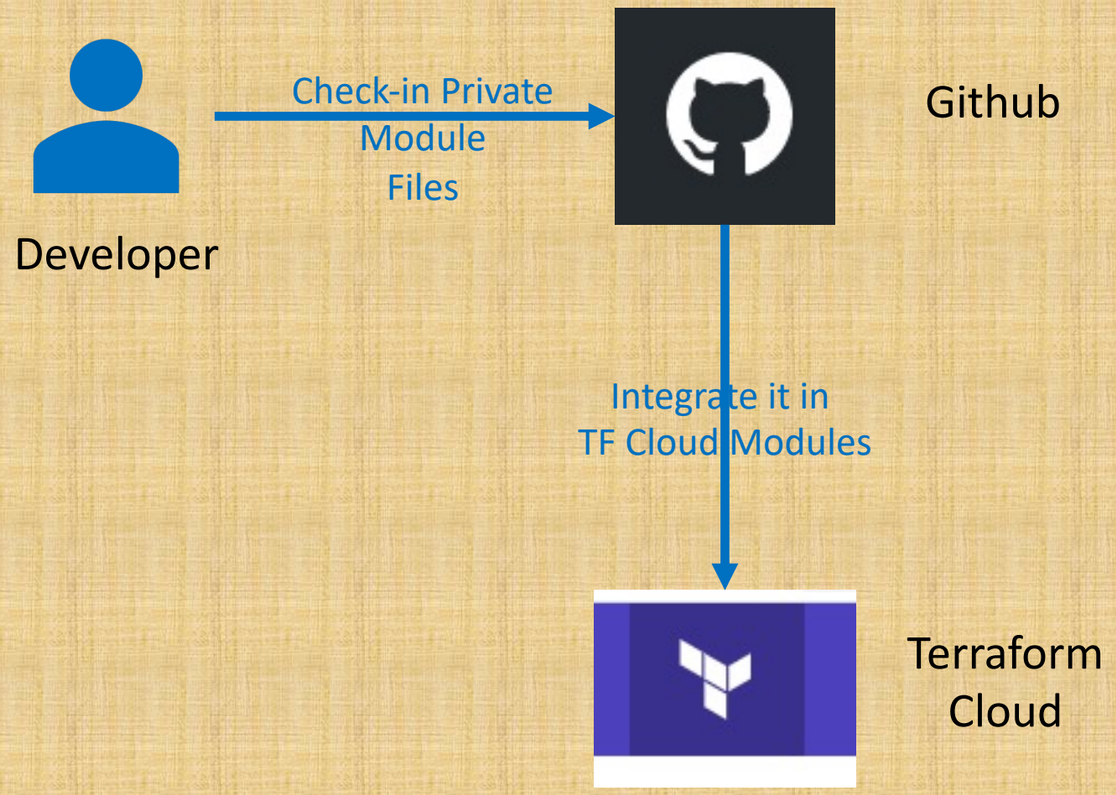
Terraform CLI-Driven Workflow



Author: Nho Luong

Skill: DevOps Engineer Lead

Publish Private Module Registry in Terraform Cloud



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Cloud & Sentinel

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Cloud

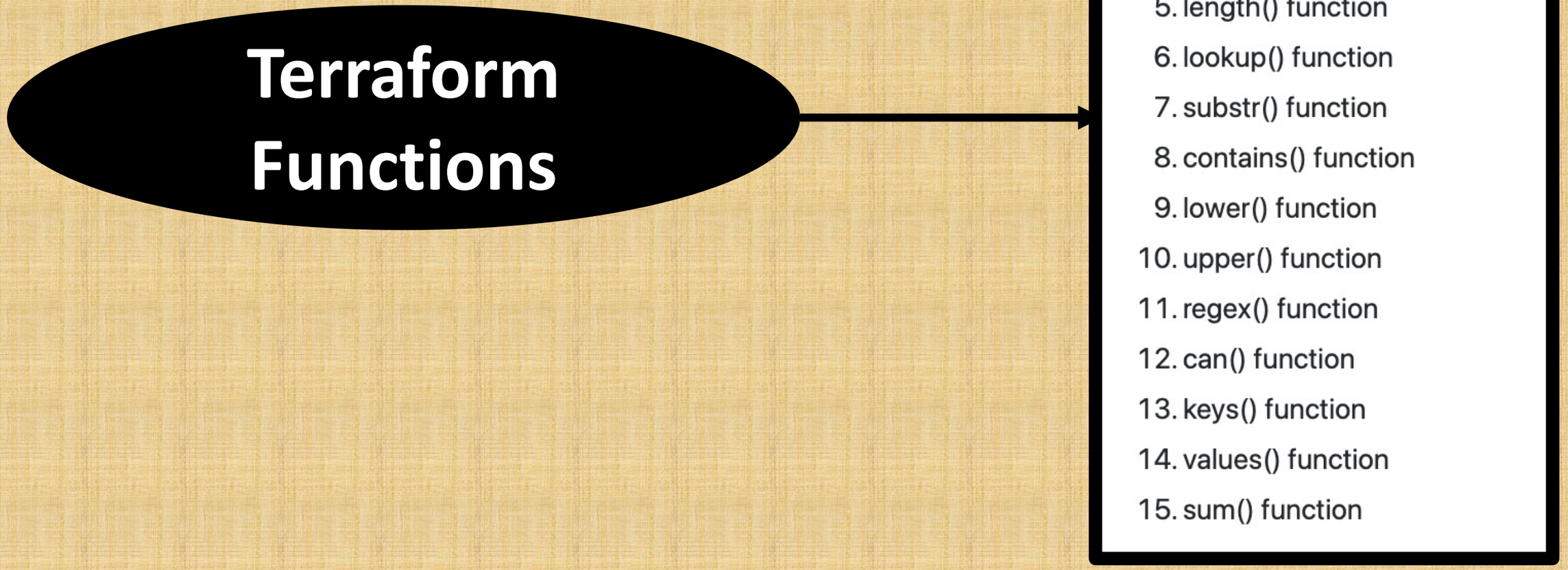
- 54-Terraform-Cloud-Github-Integration
- 55-Share-Modules-in-Private-Module-Registry
- 56-Terraform-Cloud-CLI-Driven-Workflow
- 57-Migrate-State-to-Terraform-Cloud
- 58-Terraform-Cloud-and-Sentinel-Policies
- 59-Terraform-Foundational-Policies-using-Sentinel

Terraform Cloud - Version Control Workflow	8 lectures • 1hr 18min
Terraform Cloud - Private Module Registry	4 lectures • 34min
Terraform Cloud - CLI Driven Workflow	3 lectures • 27min
Terraform Cloud - Migrate State to Terraform Cloud	3 lectures • 22min
Terraform Cloud - Sentinel Policies	5 lectures • 51min
Terraform Cloud - Sentinel Foundational Policies	2 lectures • 11min

Author: Nho Luong

Skill: DevOps Engineer Lead

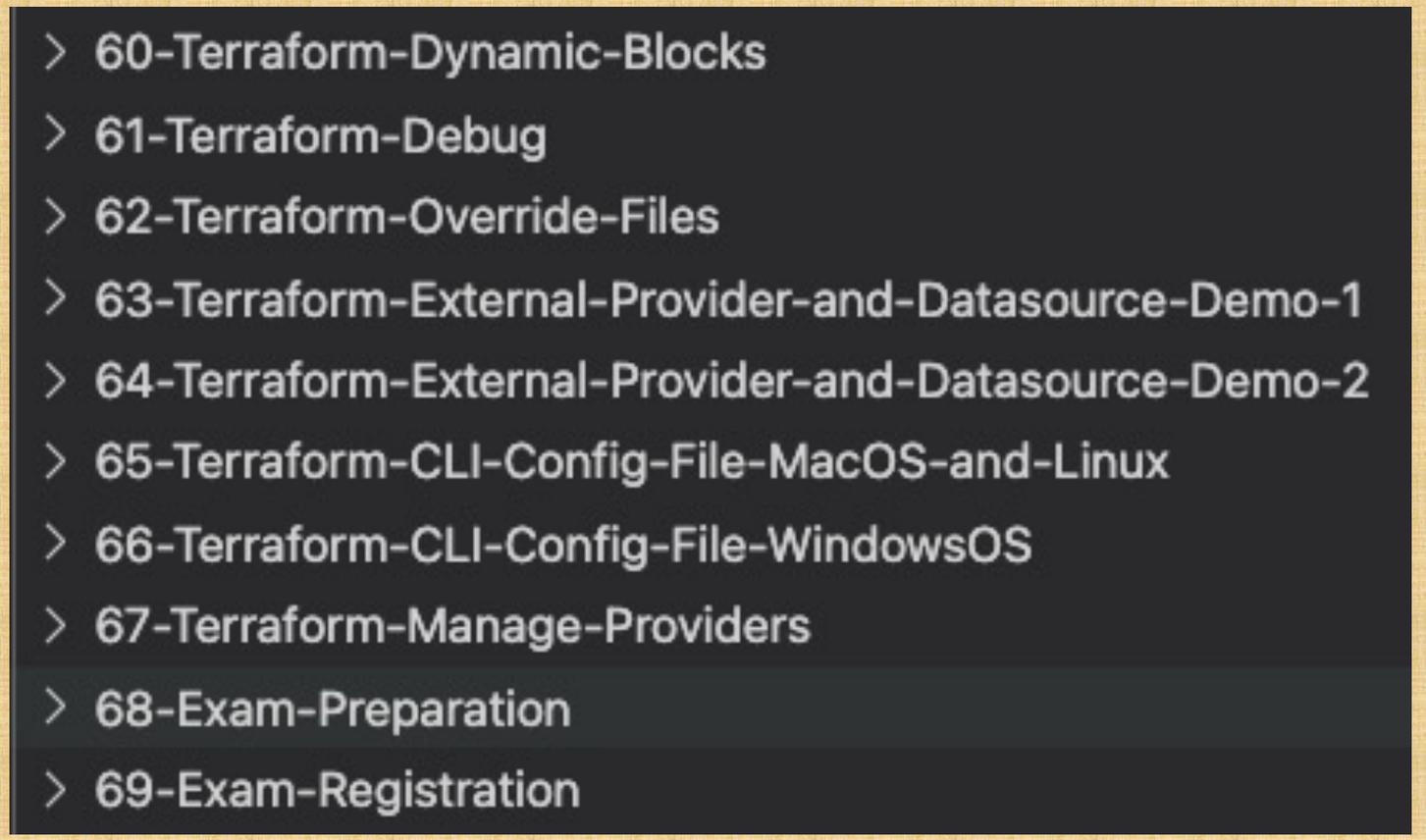
Terraform Functions



Author: Nho Luong

Skill: DevOps Engineer Lead

Additional Terraform Concepts



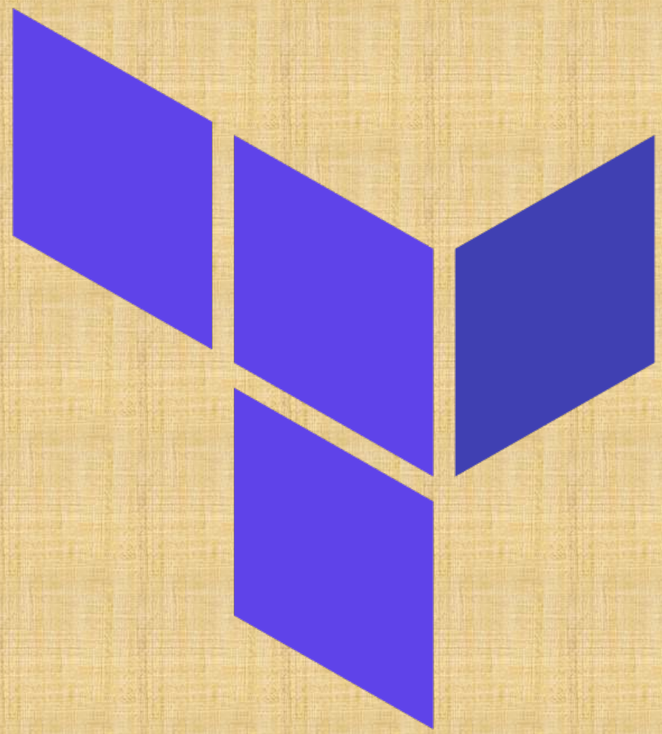
Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform

Infrastructure as Code

IaC



Author: Nho Luong

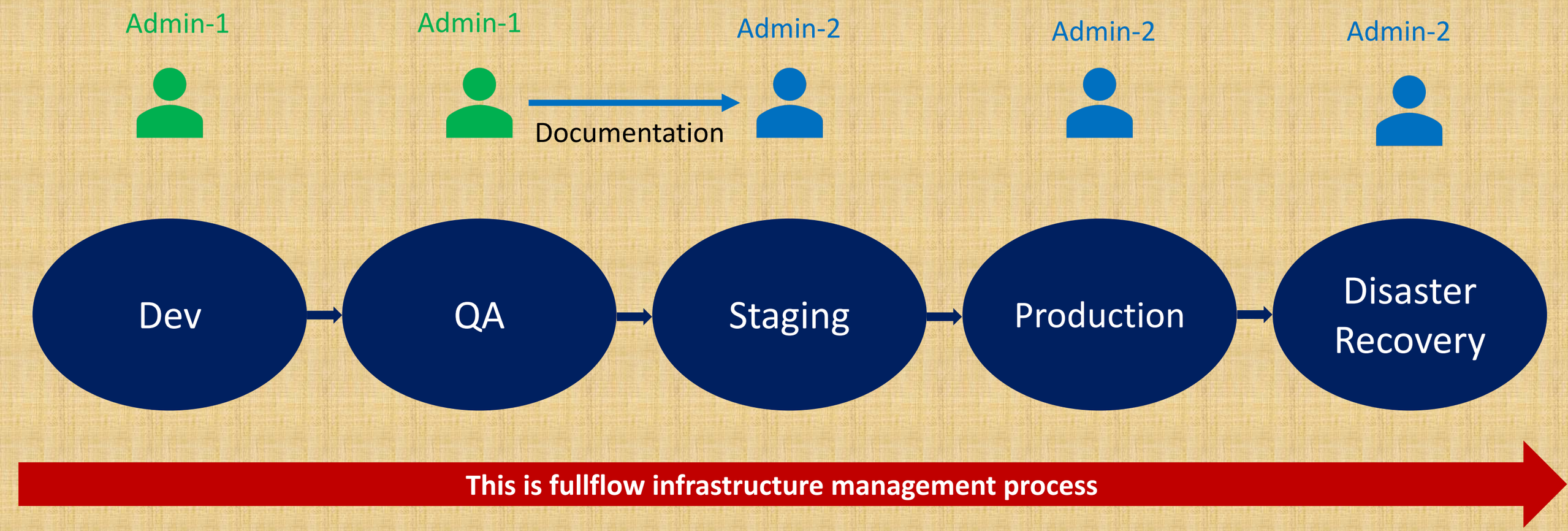
Skill: DevOps Engineer Lead

What is Infrastructure as Code ?

Author: Nho Luong

Skill: DevOps Engineer Lead

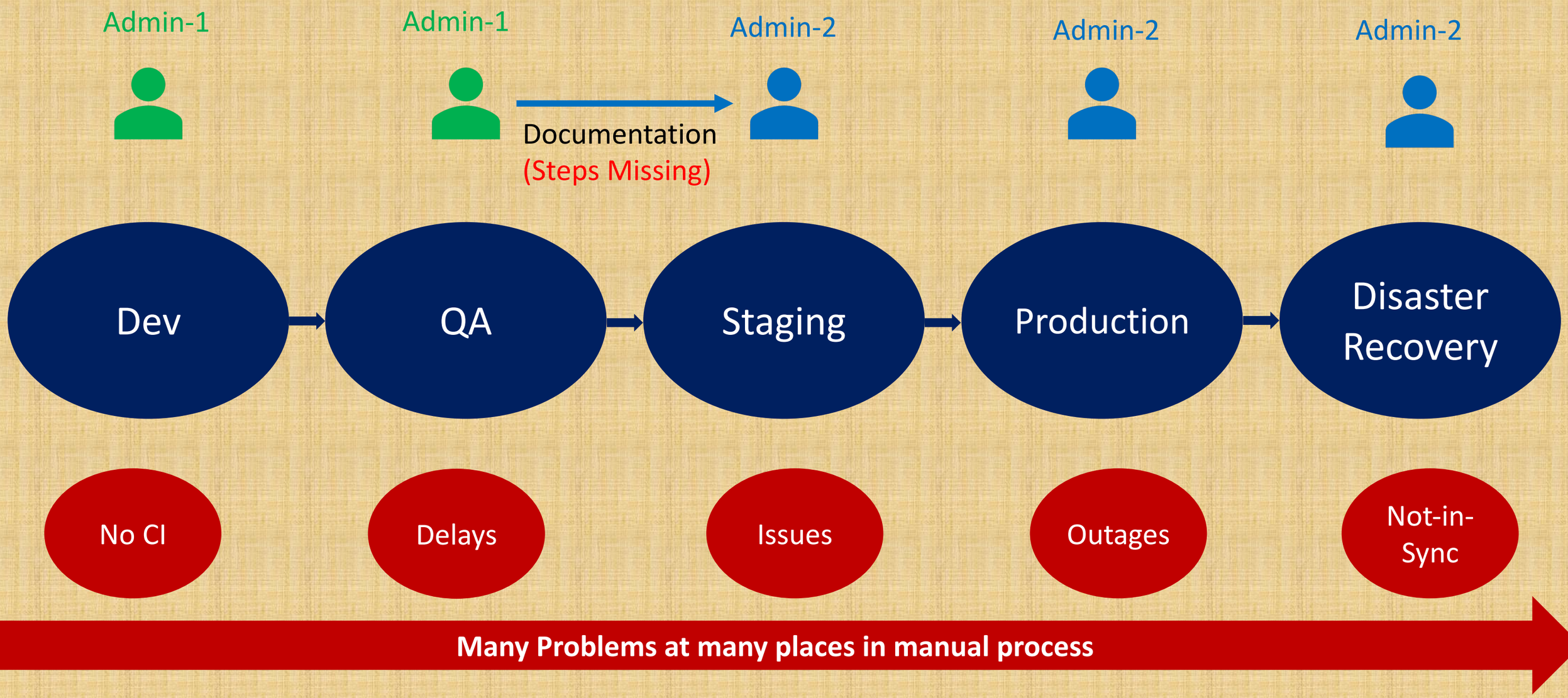
Traditional Way of Managing Infrastructure



Author: Nho Luong

Skill: DevOps Engineer Lead

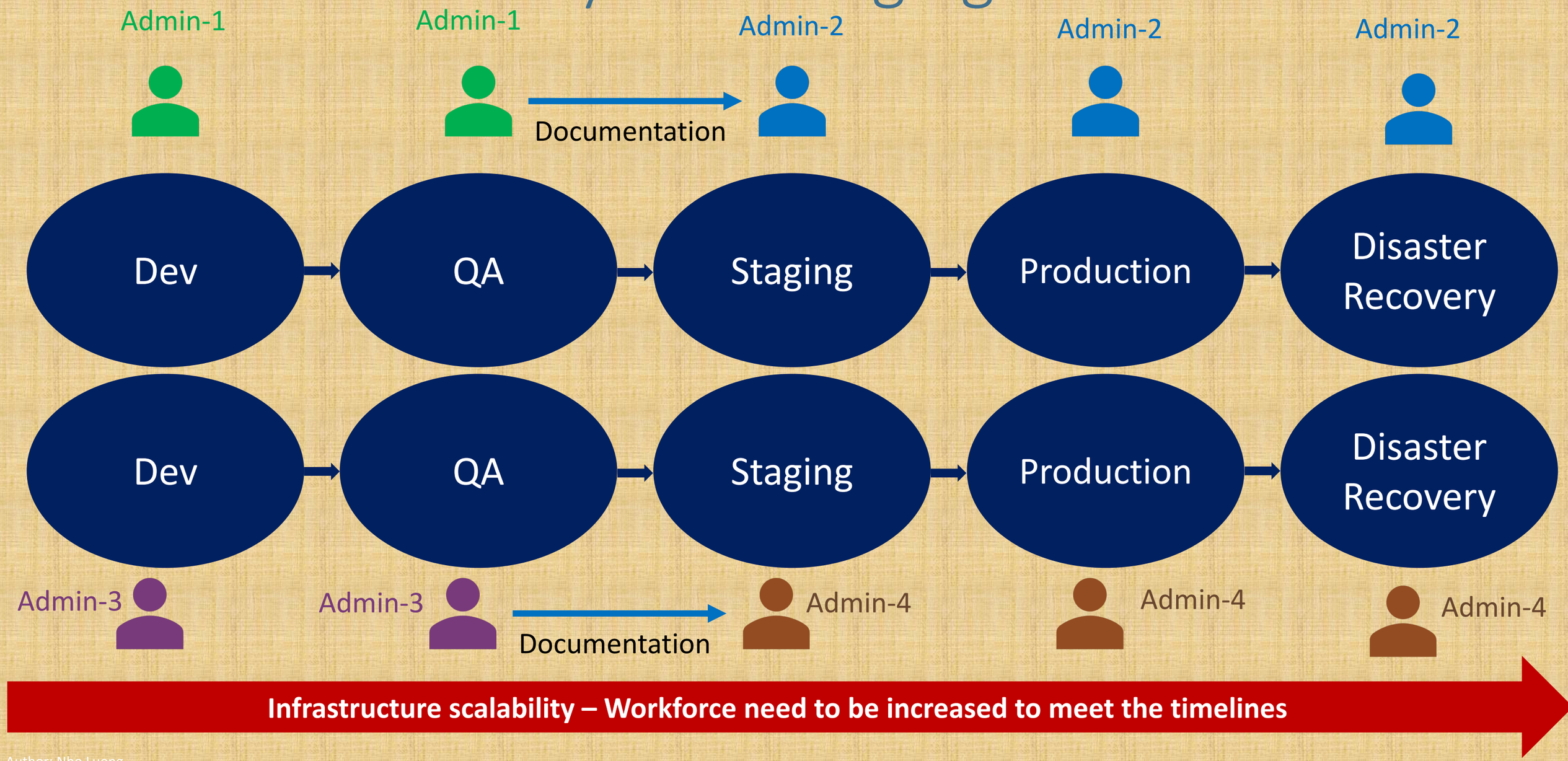
Traditional Way of Managing Infrastructure



Author: Nho Luong

Skill: DevOps Engineer Lead

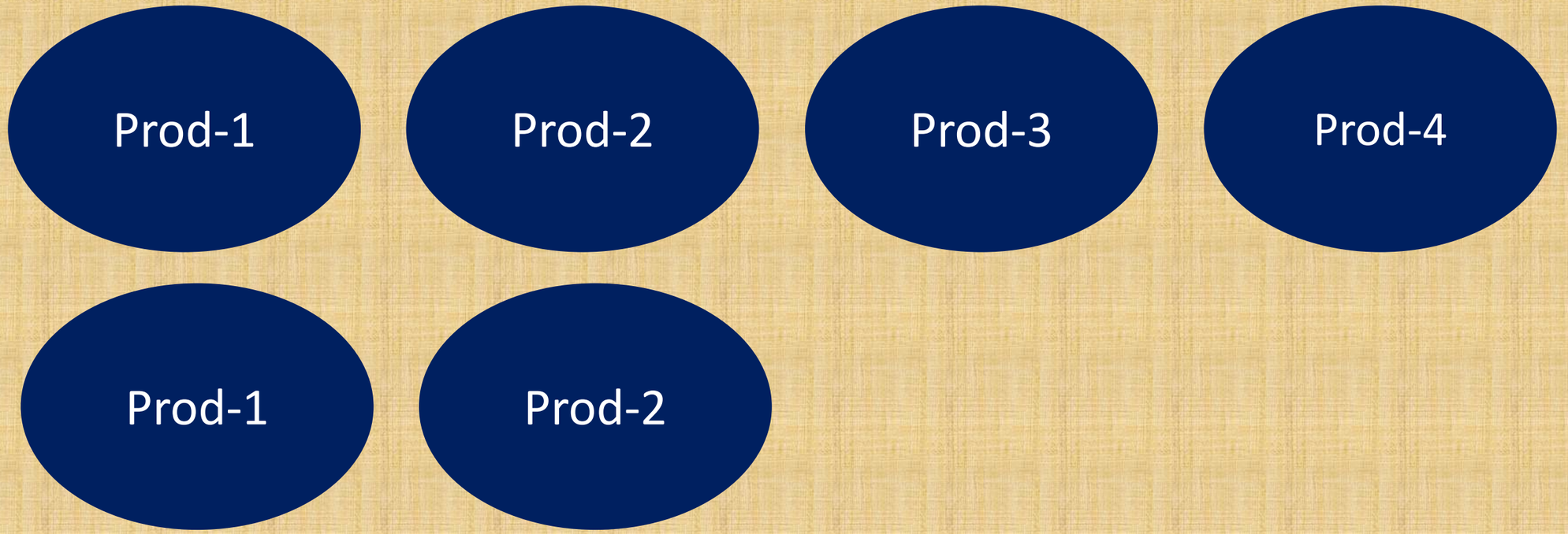
Traditional Way of Managing Infrastructure



Author: Nho Luong

Skill: DevOps Engineer Lead

Traditional Way of Managing Infrastructure



Scale Up

Scale Down

On-Demand Scale-Up and Scale-Down is not an option

Author: Nho Luong

Skill: DevOps Engineer Lead

5 Days

Manage using IaC with Terraform

Admin-1

Check-In TF Code

Github

Triggers
TF Runs

Terraform
Cloud

DevOps / CI CD for IaC

Scale-Up and Scale-Down On-Demand

Creates Infra

Dev → QA → Staging → Production → Disaster Recovery

One-Time Work

Re-Use Templates

Quick & Fast

Reliable

Tracked for Audit

Provisioning environments will be in minutes or seconds

Author: Nho Luong

Skill: DevOps Engineer Lead

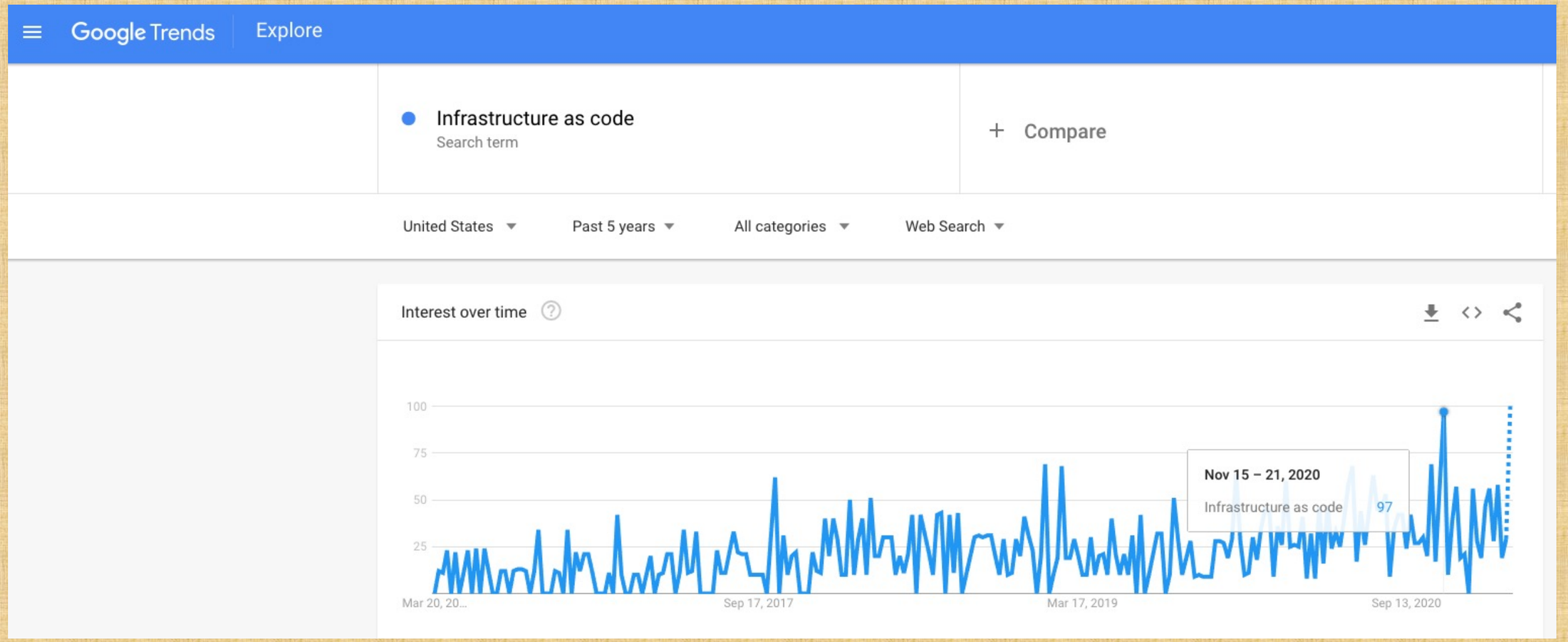
Manage using IaC with Terraform

Visibility	IaC serves as a very clear reference of what resources we created, and what their settings are. We don't have to navigate to the web console to check the parameters.
Stability	If you accidentally change the wrong setting or delete the wrong resource in the web console you can break things. IaC helps solve this, especially when it is combined with version control, such as Git.
Scalability	With IaC we can write it once and then reuse it many times. This means that one well written template can be used as the basis for multiple services, in multiple regions around the world, making it much easier to horizontally scale.
Security	Once again IaC gives you a unified template for how to deploy our architecture. If we create one well secured architecture we can reuse it multiple times, and know that each deployed version is following the same settings.
Audit	Terraform not only creates resources it also maintains the record of what is created in real world cloud environments using its State files.

Author: Nho Luong

Skill: DevOps Engineer Lead

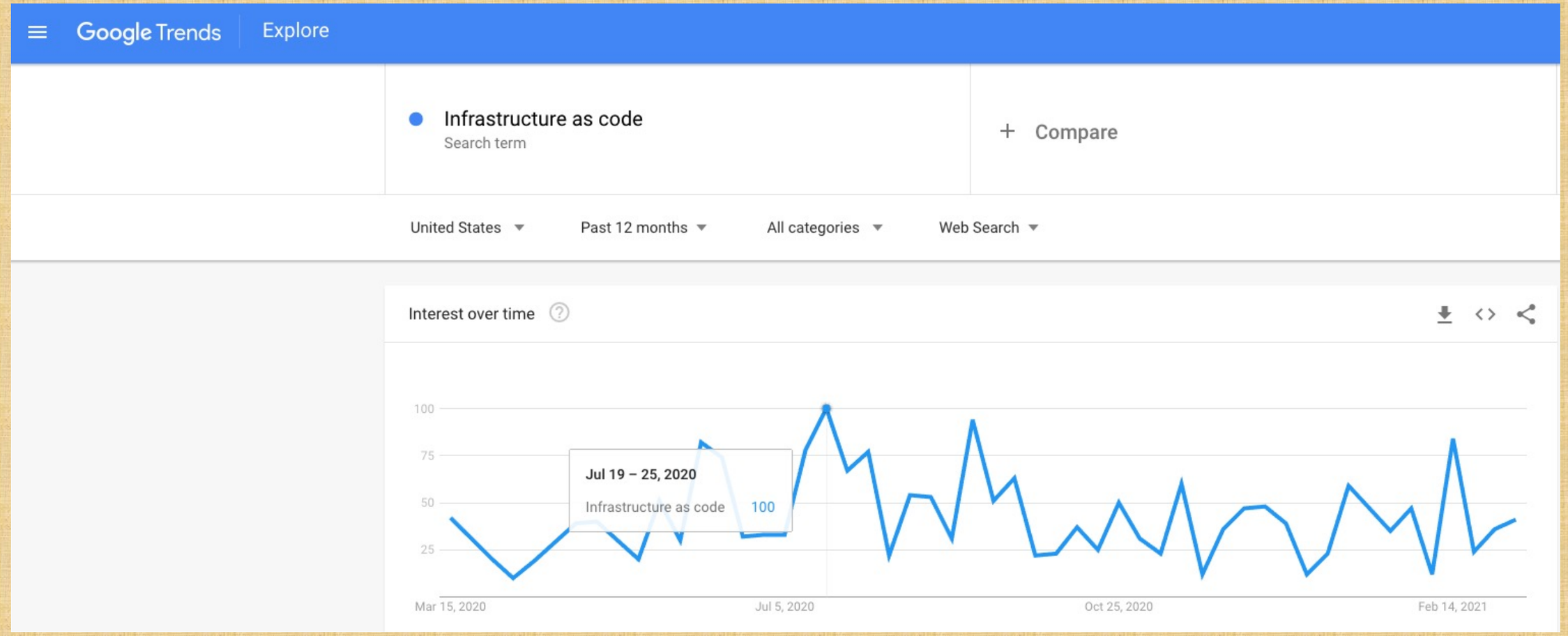
Google Trends – Past 5 Years



Author: Nho Luong

Skill: DevOps Engineer Lead

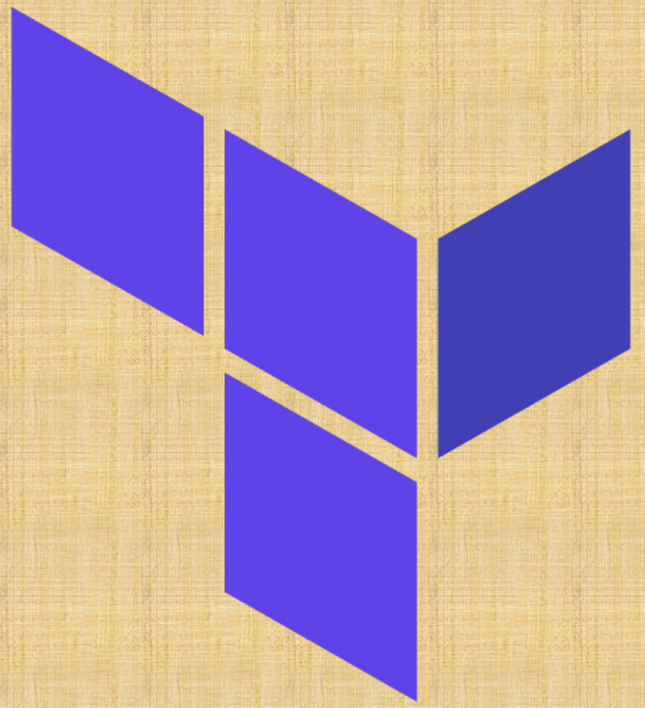
Google Trends – Past 1 Year



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Installation



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Installation

Terraform CLI

Azure CLI

VS Code Editor

Terraform plugin for VS Code

GIT Client

Mac OS

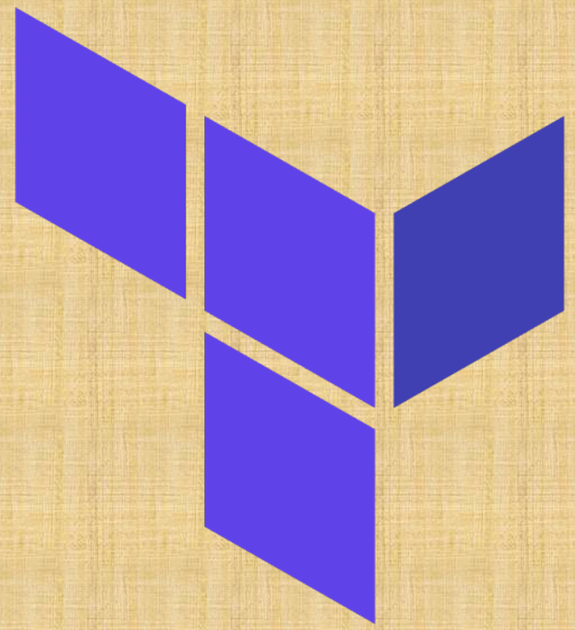
Windows OS

Linux OS

Author: Nho Luong

Skill: DevOps Engineer Lead

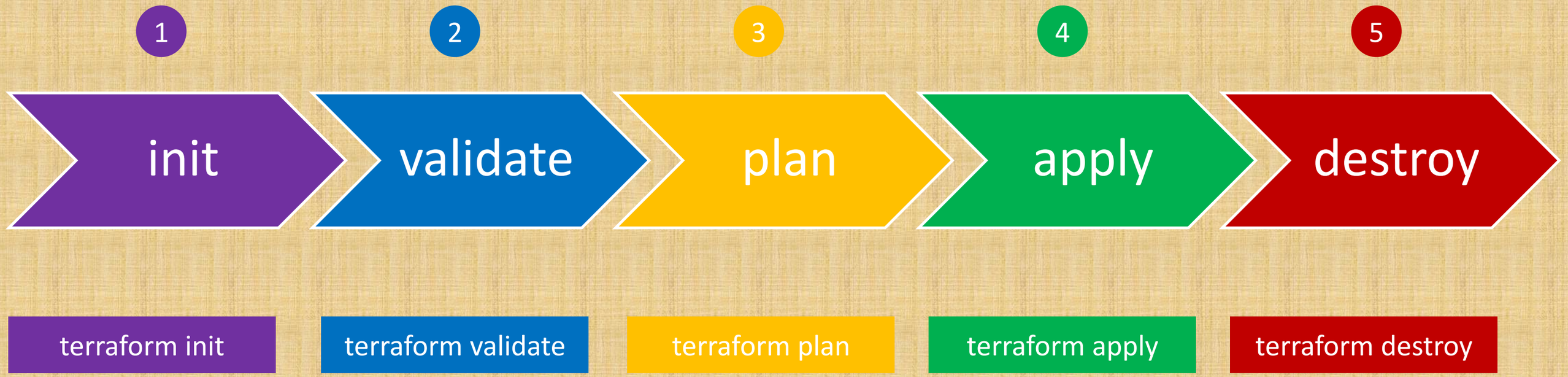
Terraform
Command Basics



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Workflow



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Workflow

1 2 3 4 5

init

- Used to Initialize a working directory containing terraform config files
- This is the first command that should be run after writing a new Terraform configuration
- Downloads Providers

validate

- Validates the terraform configurations files in that respective directory to ensure they are syntactically valid and internally consistent.

plan

- Creates an execution plan
- Terraform performs a refresh and determines what actions are necessary to achieve the desired state specified in configuration files

apply

- Used to apply the changes required to reach the desired state of the configuration.
- By default, apply scan s the current directory for the configuration and applies the changes appropriately.

destroy

- Used to destroy the Terraform-managed infrastructure
- This will ask for confirmation before destroying.

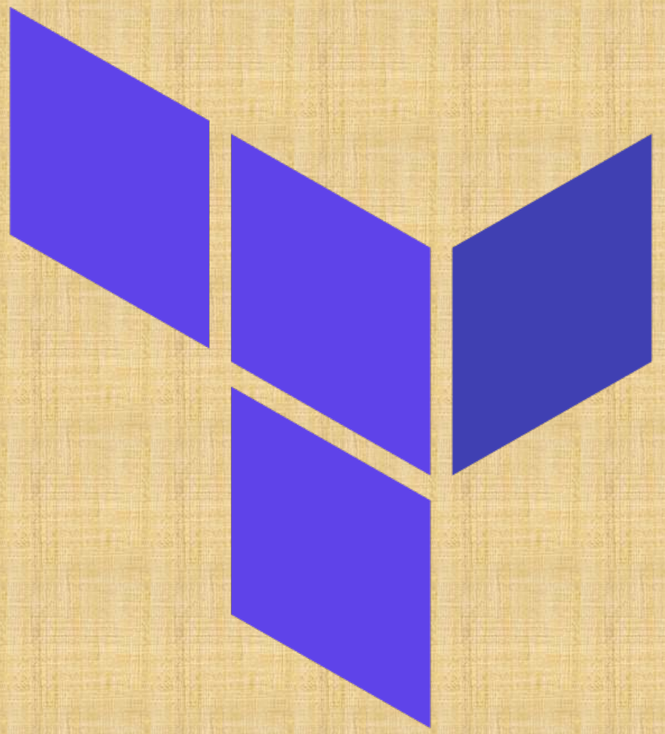
Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform

Language Basics

Theoretical but Very Important

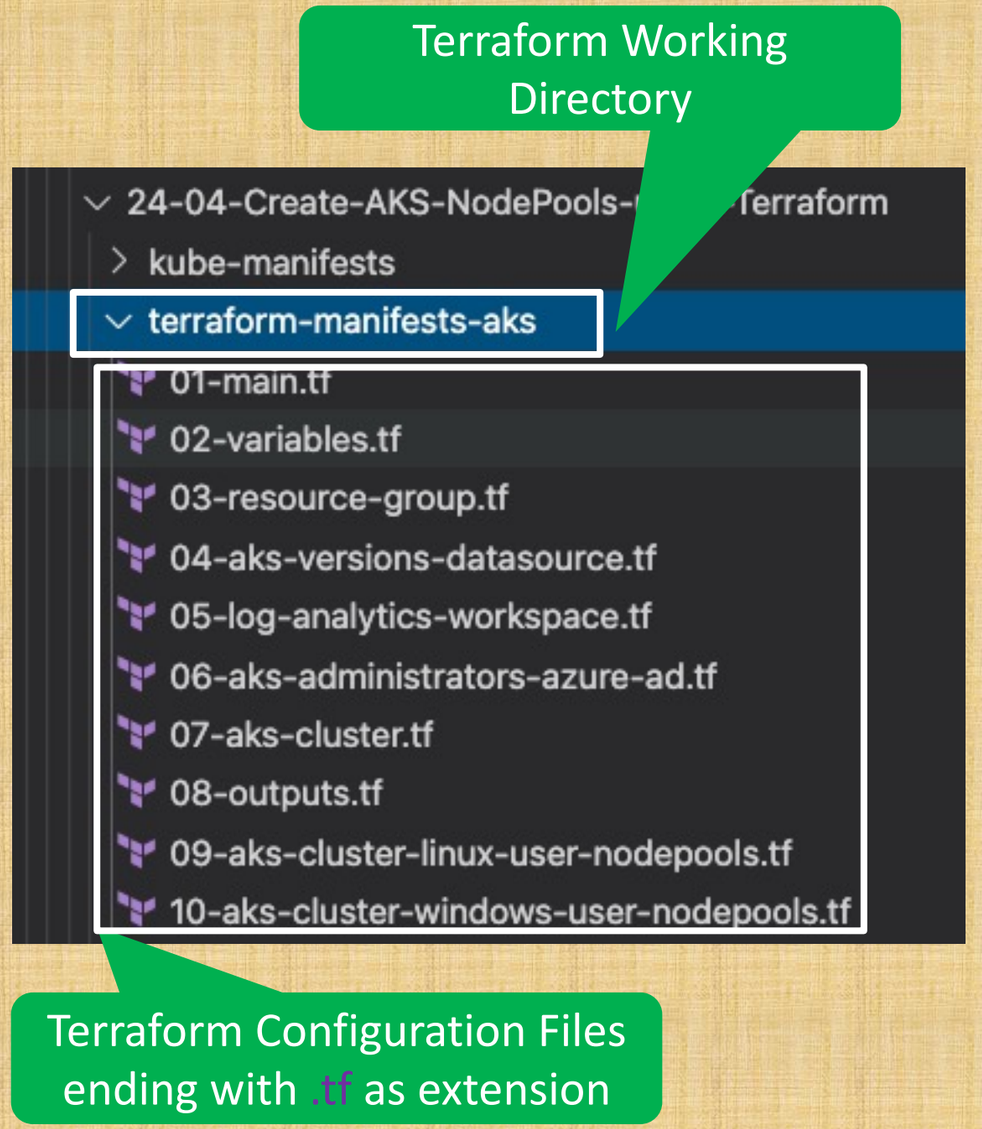


Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Language Basics – Files

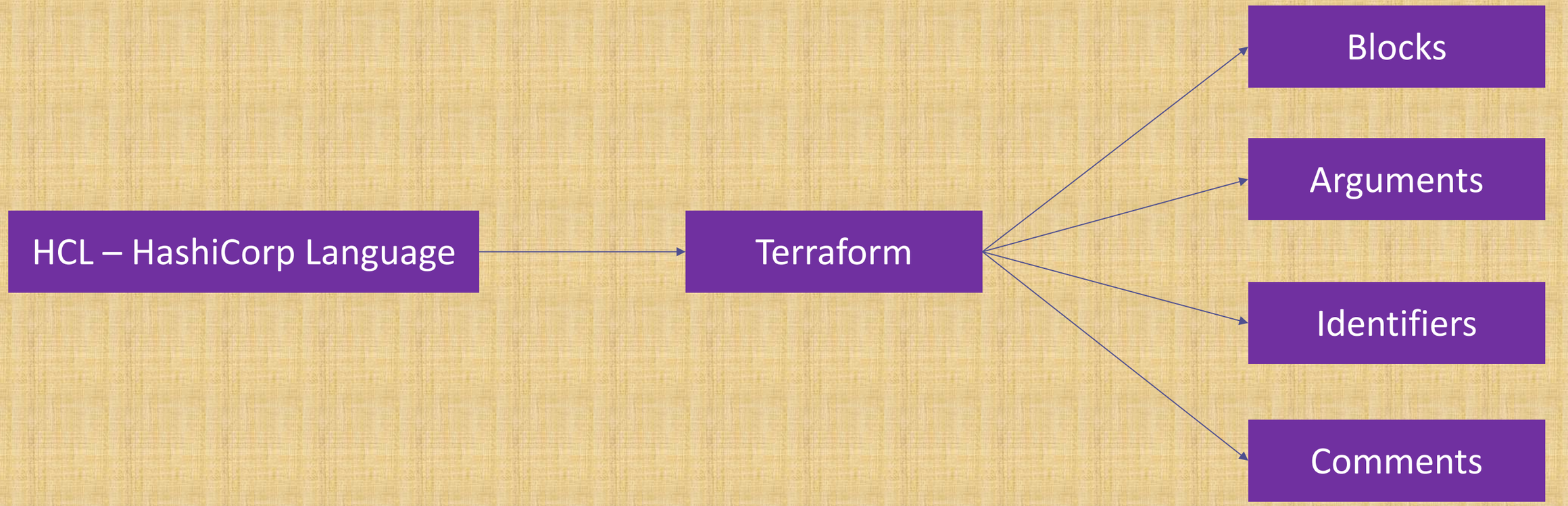
- Code in the Terraform language is stored in plain text files with the .tf file extension.
- There is also a JSON-based variant of the language that is named with the .tf.json file extension.
- We can call the files containing terraform code as Terraform Configuration Files or Terraform Manifests



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Language Basics – Configuration Syntax



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Language Basics – Configuration Syntax

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

# Azure Example
# Create a resource group
resource 'azurerm_resource_group" "myrg" { # Resource BLOCK
  name = "myrg-1" # Argument
  location = "East US" # Argument
}
```

Block Type

Top Level & Block inside Blocks

Top Level Blocks: resource, provider

Block Inside Block: provisioners, resource specific blocks like tags

Block Labels

Arguments

Based on Block Type block labels will be 1 or 2
Example:
Resource – 2 labels
Variables – 1 label

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Language Basics – Configuration Syntax

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

# Azure Example
# Create a resource group
resource "azurerm_resource_group" "myrg" { # Resource BLOCK
  name     = "myrg-1" # Argument
  location = "East US" # Argument
}
```

Argument Name [or] Identifier

Argument Value [or] Expression

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Language Basics – Configuration Syntax

Single Line Comments with # or //

Multi-line comment

```
# Create a resource group
resource "azurerm_resource_group" "myrg" { // Azure Resource Group
  name = "myrg-1"
  location = "eastus"
}
/*
Multi-line Comments
Line-1
Line-2
*/
```

Author: Nho Luong

Skill: DevOps Engineer Lead

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.

Terraform Top-Level Blocks

Most of **Terraform's features** are implemented as **top-level** blocks.

Fundamental Blocks

- Terraform Block
- Providers Block
- Resources Block

Variable Blocks

- Input Variables Block
- Output Values Block
- Local Values Block

Calling / Referencing Blocks

- Data Sources Block
- Modules Block

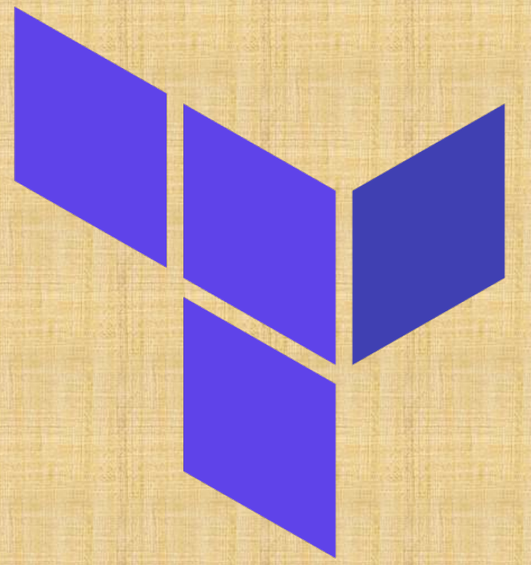
Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform

Fundamental Blocks

Terraform, Provider, Resources



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Basic Blocks

Terraform Block

- Special block used to configure some behaviors
- Specifying a required Terraform CLI Version
- Specifying Provider Requirements & Versions
- Configuring a Terraform Backend (Terraform State)

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

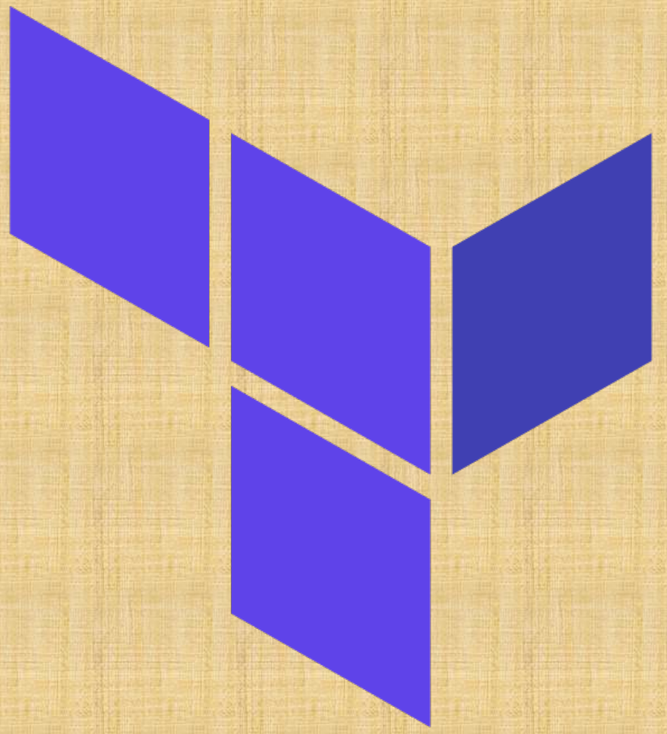
Resource Block

- 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

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Block



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Block

- This block can be called in 3 ways. All means the same.
  - Terraform Block
  - Terraform Settings Block
  - Terraform Configuration Block
- Each terraform block can contain a number of settings related to Terraform's behavior.
- Within a terraform block, only constant values can be used; arguments may not refer to named objects such as resources, input variables, etc, and may not use any of the Terraform language built-in functions.

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Block from 0.13 onwards

Terraform 0.12 and earlier:

```
# Configure the AWS Provider
provider "aws" {
  version = "~> 3.0"
  region  = "us-east-1"
}

# Create a VPC
resource "aws_vpc" "example" {
  cidr_block = "10.0.0.0/16"
}
```

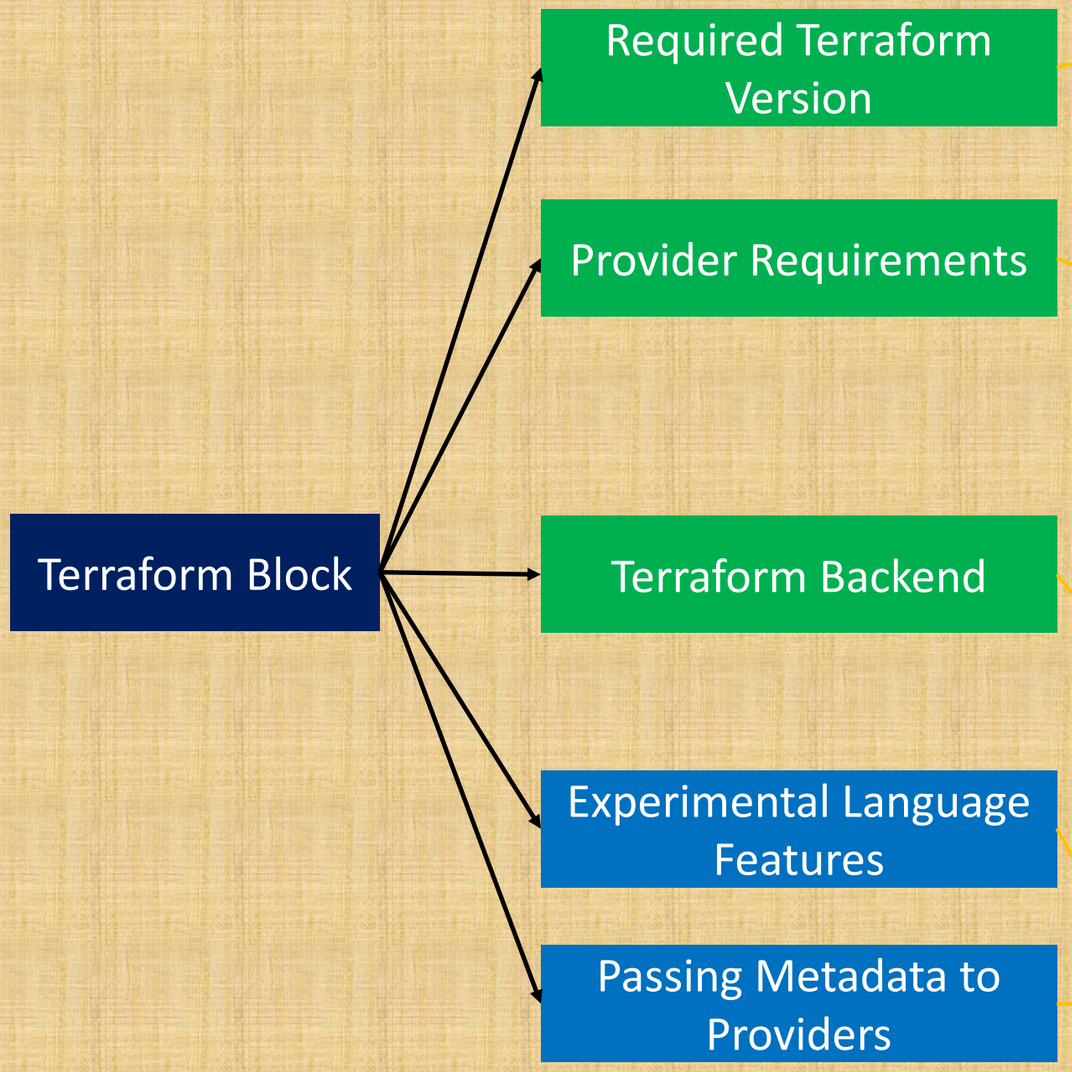
Terraform 0.13 and later:

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 3.0"
    }
  }
}

# Configure the AWS Provider
provider "aws" {
  region = "us-east-1"
}

# Create a VPC
resource "aws_vpc" "example" {
  cidr_block = "10.0.0.0/16"
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Block

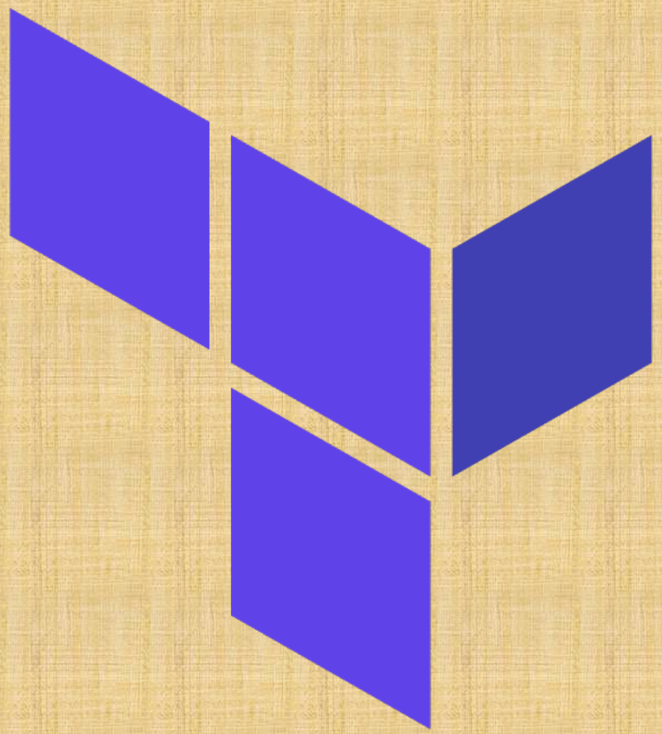



```hcl
terraform {
  # Required Terraform Version
  required_version = ">= 1.0.0"

  # Required Providers and their Versions
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">= 2.0" # Optional but recommended
    }
  }

  # Terraform State Storage to Azure Storage Container
  backend "azurerm" {
    resource_group_name   = "terraform-storage-rg"
    storage_account_name  = "terraformstate201"
    container_name        = "tfstatefiles"
    key                   = "terraform.tfstate"
  }

  experiments = [ example ]# Experimental (Not required)
  provider_meta "my-provider" { # Super Advanced (Not required)
    hello = "world"
  }
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Providers



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Providers

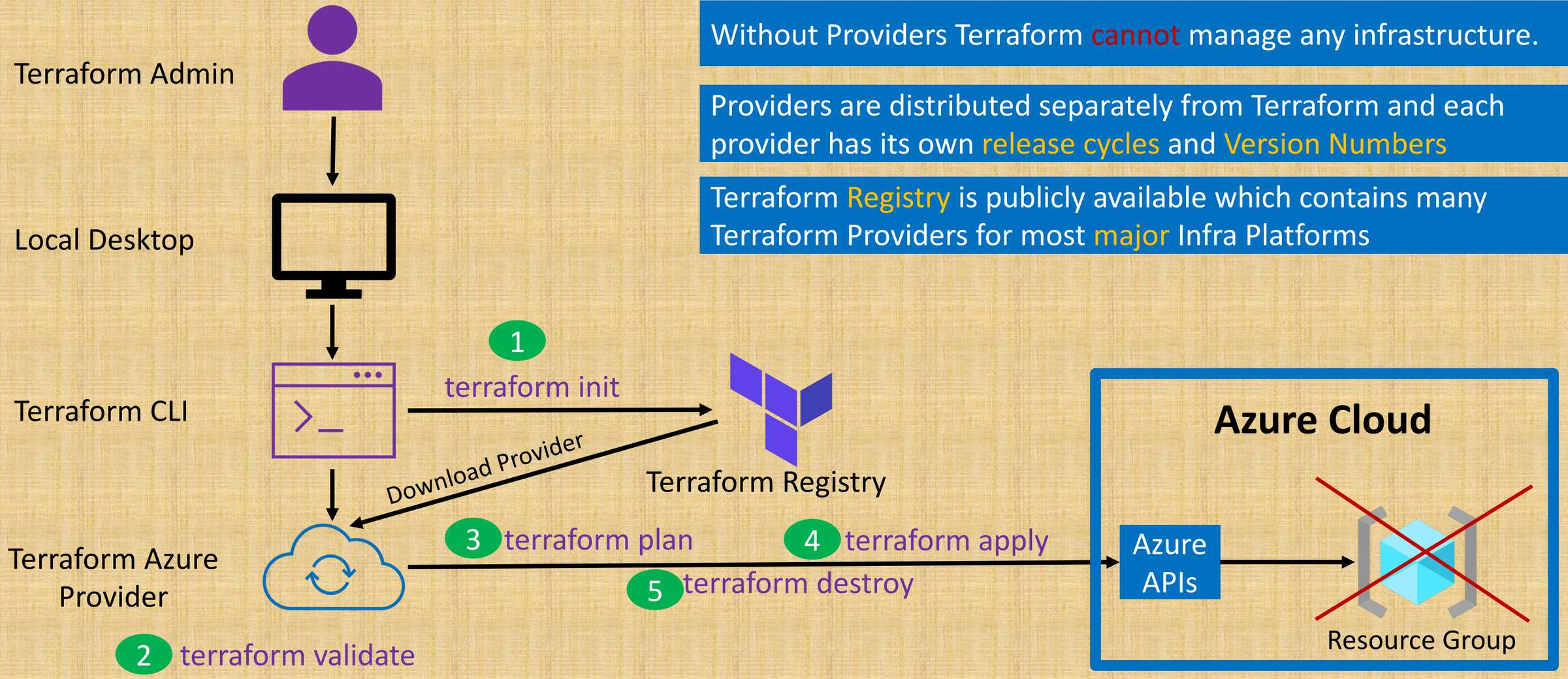
Providers are HEART of Terraform

Every Resource Type (example: Azure Resource Group), is implemented by a Provider

Without Providers Terraform cannot manage any infrastructure.

Providers are distributed separately from Terraform and each provider has its own release cycles and Version Numbers

Terraform Registry is publicly available which contains many Terraform Providers for most major Infra Platforms



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Providers

Provider Requirements

```
# Terraform Block
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = ">= 2.0"
    }
  }
}
```

Provider Configuration

```
# Provider Block
provider "azurerm" {
features {}
}
```

Dependency Lock File


```
# This file is maintained automatically by "terraform init".
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/azurerm" {
  version     = "2.64.0"
  constraints = ">= 2.0.0"
  hashes = [
    "h1:6qwrh6hD19XCreyps0ojY1Vp9VI5vr1bCbl1EZGy6Io=",
    "zh:048da64c3d173f3467e908ca8b28962bbf6e3e06597614474b46d4c09ec8ca6c",
    "zh:08d5baa31e498b2e7761d88a8ff5875066566f9b3644a5c12c8ea305e1a7c85d",
    "zh:0a452f95795f56c16f5b0febe05539f44638895f387973f594ac3de179f22150",
    "zh:1b0dd54a023ef22c9fadea2cb6e8e66e2c9a29d23921706e10b35b1bd2a47ed3",
    "zh:3260fdc14d2a33c0cc0e7b230687d303e12852c54aea1120918e7b77f954ed1b",
    "zh:b36dd823f543fb45b31a623ff68be5fd49fddcb50c2e032dd44828a26f9d41b9",
    "zh:ba6514590b1be102438cc3632795965f3e271044635bf05ec0f0e46c4795b06c",
    "zh:bf663c286ba4198111d8b9f4987a277f1a378d0ef40a824c8fc25f6e4de1866d",
    "zh:d8c122295c29c90788b4cdbcab53e1fc5c75071d6600b5389372dc8f8bda1967",
    "zh:dffbb1af6ab6d1b875cbd4fd198d3b12c28261a216c1a32f393b8c795b62bb30",
    "zh:f8dca9bf566e7869412a5c10e44f64f3a19eee7c4aa18e7ce5db9a6f145a2a4f",
  ]
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Dependency Lock File

```
# This file is maintained automatically by "terraform init".
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/azurerm" {
  version     = "2.64.0"
  constraints = ">= 2.0.0"
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Required Providers

```
# Terraform Block
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = ">= 2.0"
    }
  }
}
# Provider Block
provider "azurerm" {
features {}
}
```

Local Names

Local Names are Module specific and should be unique per-module

Terraform configurations always refer to local name of provider outside required_provider block

Users of a provider can choose any local name for it (myazure, azure1, azure2).

Recommended way of choosing local name is to use preferred local name of that provider (For Azure Provider: hashicorp/azurerm, preferred local name is azurerm)

Source

It is the primary location where we can download the Terraform Provider

Source addresses consist of three parts delimited by slashes (/)

[<HOSTNAME>/]<NAMESPACE>/<TYPE>

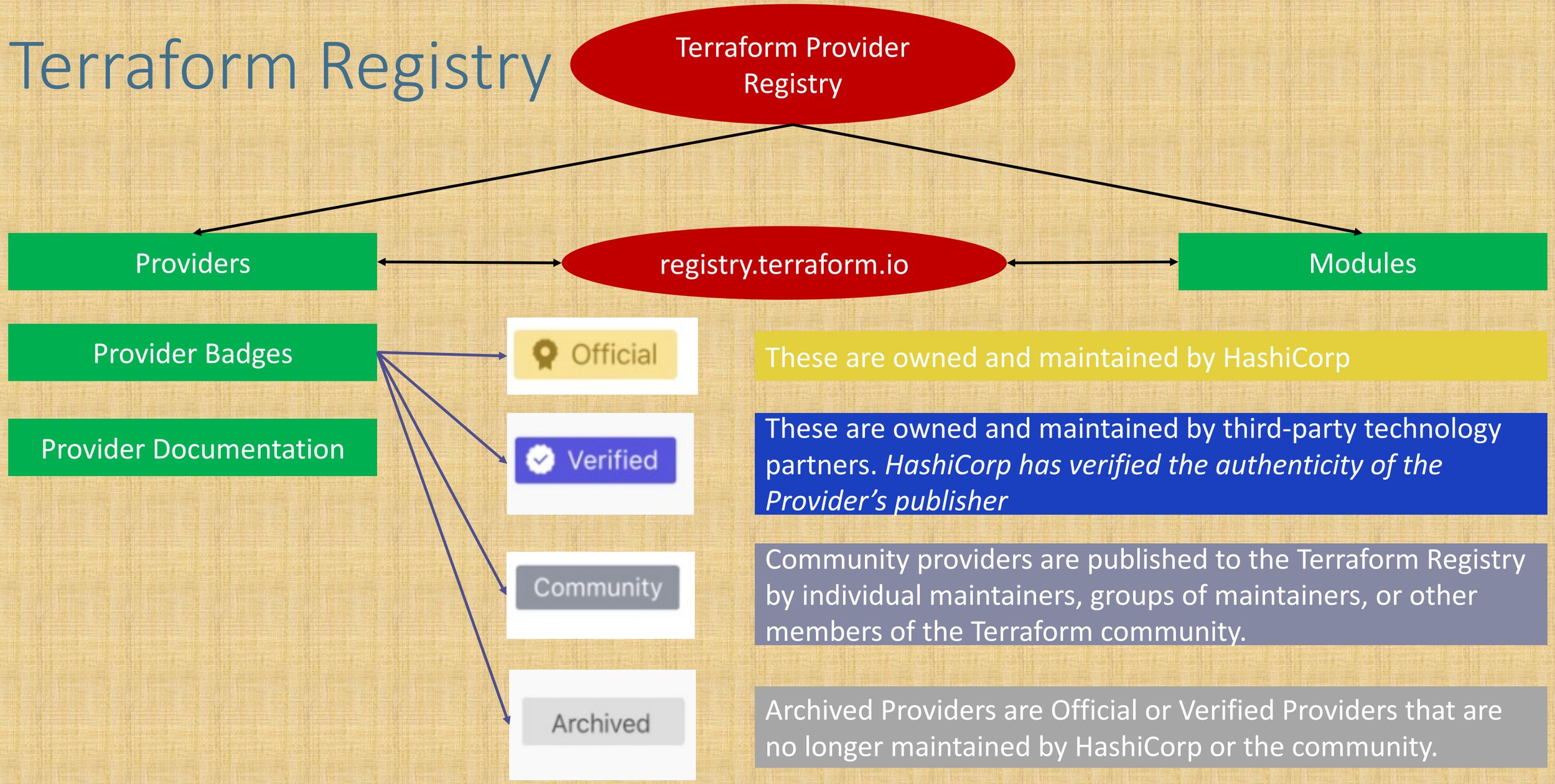
registry.terraform.io/hashicorp/azurerm

Registry Name is optional as default is going to be Terraform Public Registry

Author: Nho Luong

Skill: DevOps Engineer Lead

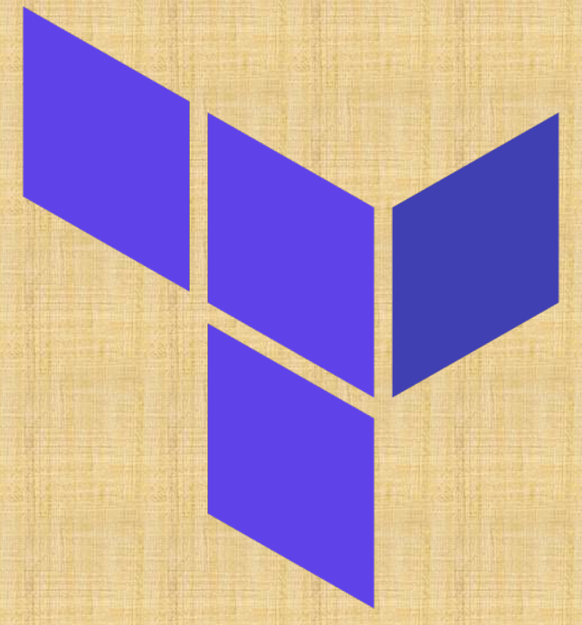
Terraform Registry



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Multiple Providers



Author: Nho Luong

Skill: DevOps Engineer Lead

Multiple Providers

We can define multiple configurations for the same provider, and select which one to use on a per-resource or per-module basis.

The primary reason for this is to support multiple regions for a cloud platform

```
# Provider-1 for EastUS
# Default Provider
provider "azurerm" {
  features {}
}

# Provider-2 for WestUS
provider "azurerm" {
  features {}
  alias = "provider2-westus"
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Multiple Providers

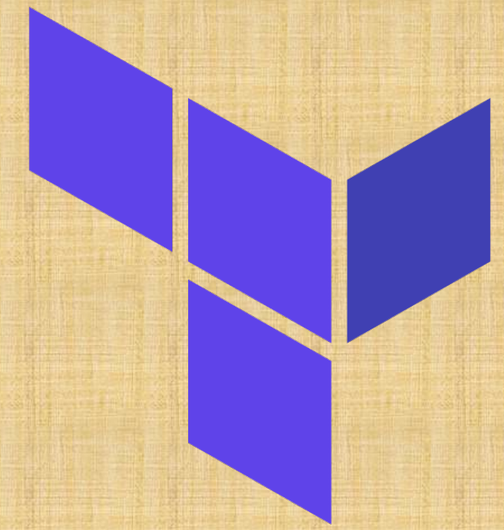
We can use the alternate provider in a resource, data or module by referencing it as <PROVIDER NAME>.<ALIAS>

```
# Uses Default Provider
resource "azurerm_resource_group" "myrg1" {
  name = "myrg-1"
  location = "East US"
}
# Uses "provider2-westus" provider
resource "azurerm_resource_group" "myrg2" {
  name = "myrg-2"
  location = "West US"
  provider = azurerm.provider2-westus
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Dependency Lock File



Author: Nho Luong

Skill: DevOps Engineer Lead

Dependency Lock File

Terraform

New feature added from Terraform v0.14 & later

Providers

Terraform configuration refers to two different kinds of external dependency that come from outside of its own codebase

Modules

Version Constraints within the configuration itself determine which versions of dependencies are *potentially* compatible

Dependency Lock File: After selecting a specific version of each dependency using Version Constraints Terraform remembers the decisions it made in a dependency lock file so that it can (by default) make the same decisions again in future.

Location of Lock File: Current Working Directory

Very Important: Lock File currently tracks only Provider Dependencies. For modules continue to use exact version constraint to ensure that Terraform will always select the same module version.

Checksum Verification: Terraform will also verify that each package it installs matches at least one of the checksums it previously recorded in the lock file, if any, returning an error if none of the checksums match

Author: Nho Luong

Skill: DevOps Engineer Lead

Dependency Lock File

```
# This file is maintained automatically by "terraform init".
# Manual edits may be lost in future updates.

provider "registry.terraform.io/hashicorp/azurerm" {
  version     = "2.64.0"
  constraints = ">= 2.0.0"
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Importance of Dependency Lock File

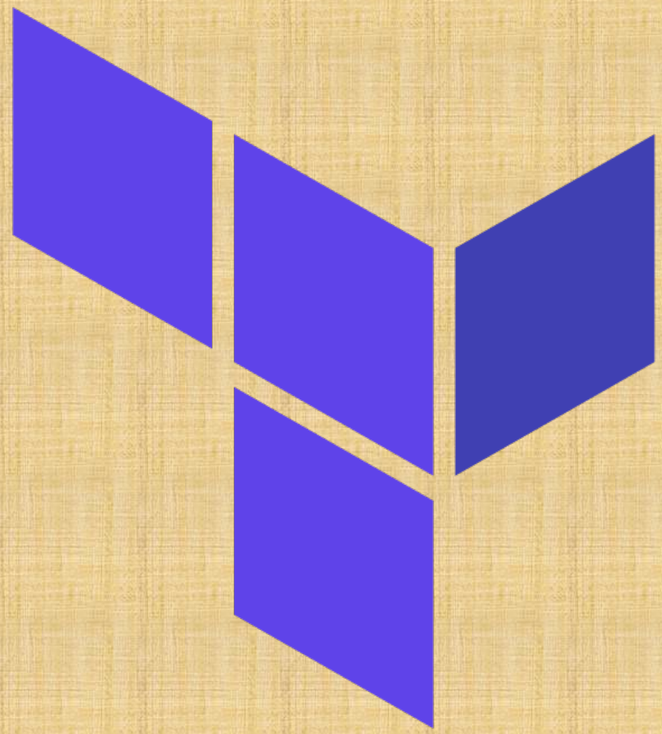
If Terraform did not find a lock file, it would download the latest versions of the providers that fulfill the version constraints you defined in the required_providers block inside Terraform Settings Block.

If we have lock file, the lock file causes Terraform to always install the same provider version, ensuring that runs across your team or remote sessions will be consistent.

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Resources Introduction



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Language Basics

Terraform
Resource Syntax

Terraform
Resource Behavior

Terraform
State

Terraform
Resources

Resource
Meta-Argument
count

Resource
Meta-Argument
depends_on

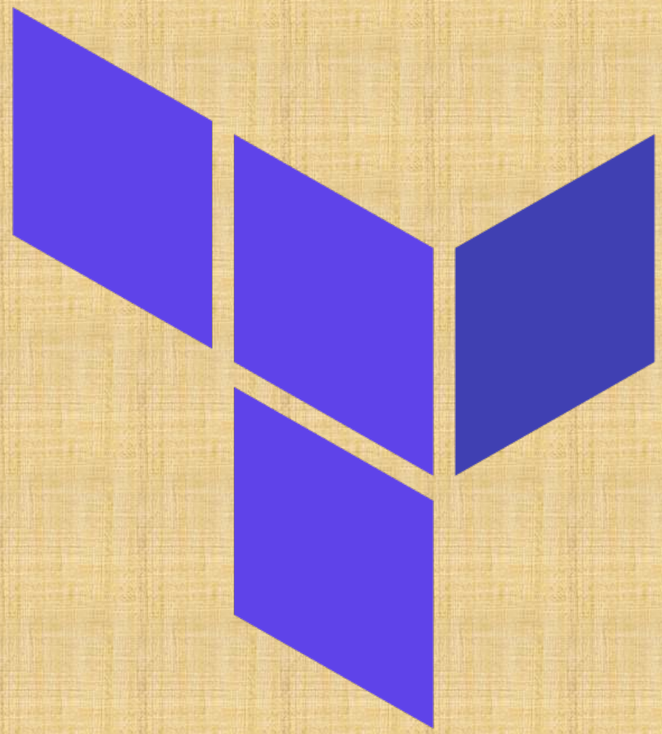
Resource
Meta-Argument
for_each

Resource
Meta-Argument
lifecycle

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Resource Syntax



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Language Basics – Configuration Syntax

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

# Azure Example
# Create a resource group
resource 'azurerm_resource_group" "myrg" { # Resource BLOCK
  name = "myrg-1" # Argument
  location = "East US" # Argument
}
```

Block Type

Top Level & Block inside Blocks

Top Level Blocks: resource, provider

Block Inside Block: provisioners, resource specific blocks like tags

Block Labels

Arguments

Based on Block Type block labels will be 1 or 2
Example:
Resource – 2 labels
Variables – 1 label

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Language Basics – Configuration Syntax

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

# Azure Example
# Create a resource group
resource "azurerm_resource_group" "myrg" { # Resource BLOCK
  name     = "myrg-1" # Argument
  location = "East US" # Argument
}
```

Argument Name [or] Identifier

Argument Value [or] Expression

Author: Nho Luong

Skill: DevOps Engineer Lead

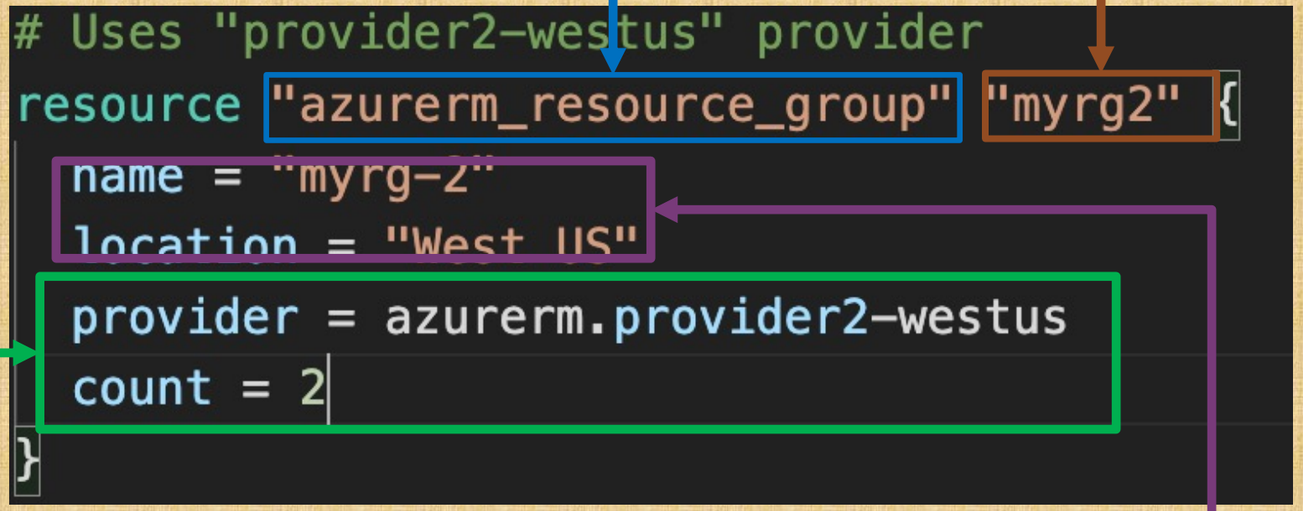
Resource Syntax

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

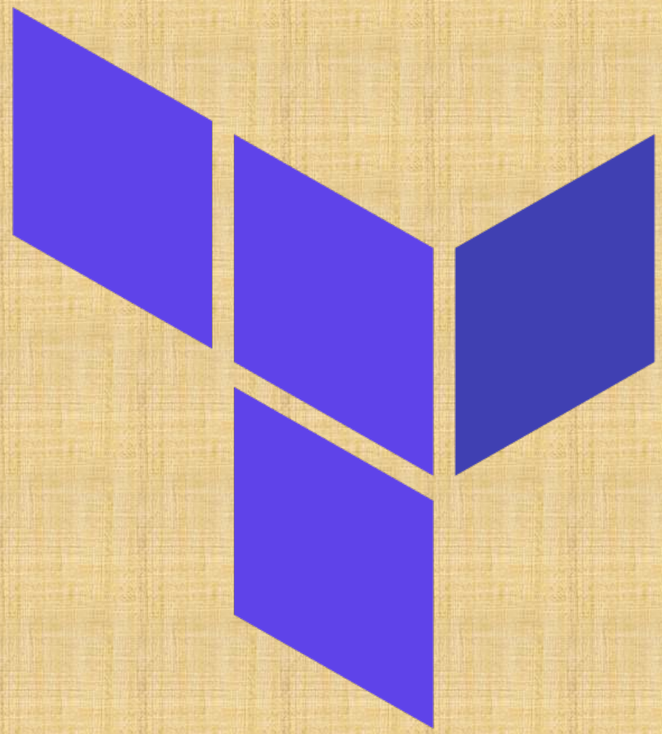


Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform

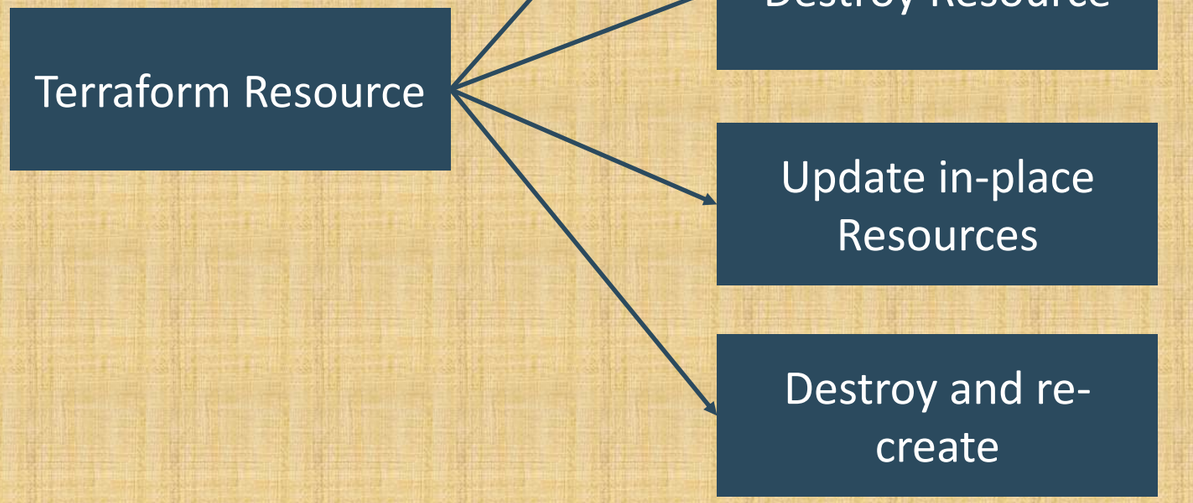
Resource Behavior



Author: Nho Luong

Skill: DevOps Engineer Lead

Resource Behavior



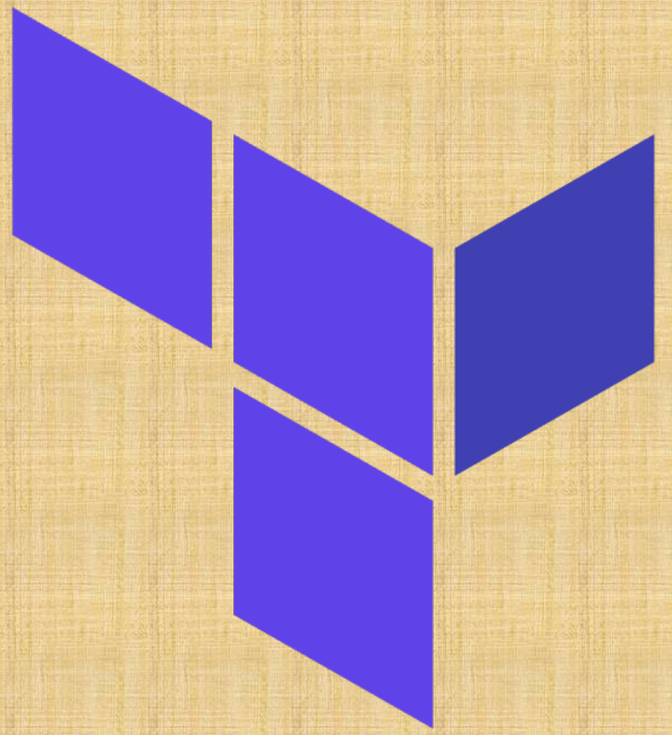
- **Create Resource**: Create resources that exist in the configuration but are not associated with a real infrastructure object in the state.
- **Destroy Resource**: Destroy resources that exist in the state but no longer exist in the configuration.
- **Update in-place Resources**: Update in-place resources whose arguments have changed.
- **Destroy and re-create**: Destroy and re-create resources whose arguments have changed but which cannot be updated in-place due to remote API limitations.

Terraform State

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
State



Author: Nho Luong

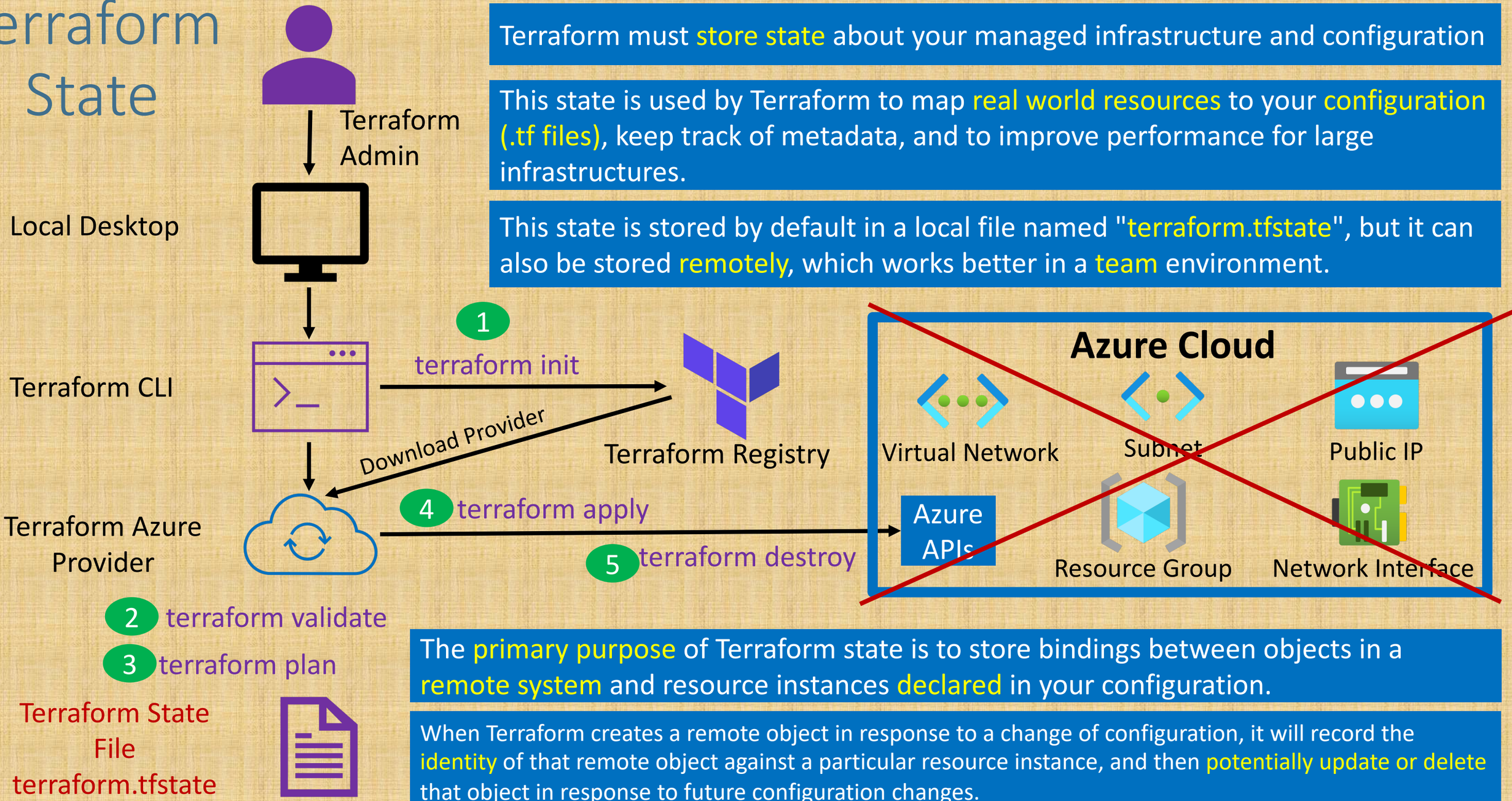
Skill: DevOps Engineer Lead

Terraform State

Terraform must store state about your managed infrastructure and configuration

This state is used by Terraform to map real world resources to your configuration (.tf files), keep track of metadata, and to improve performance for large infrastructures.

This state is stored by default in a local file named "terraform.tfstate", but it can also be stored remotely, which works better in a team environment.



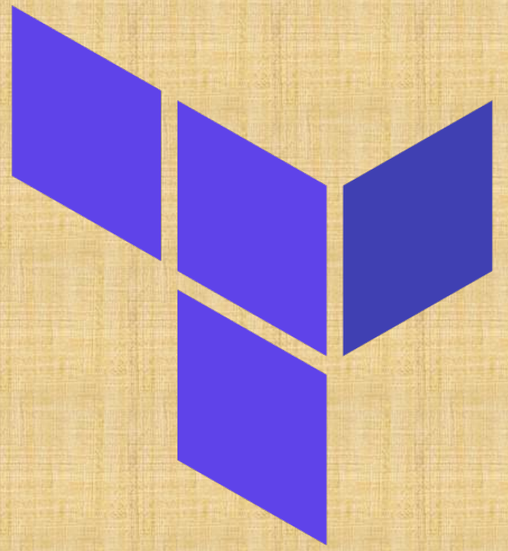
The primary purpose of Terraform state is to store bindings between objects in a remote system and resource instances declared in your configuration.

When Terraform creates a remote object in response to a change of configuration, it will record the identity of that remote object against a particular resource instance, and then potentially update or delete that object in response to future configuration changes.

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
State
Desired & Current



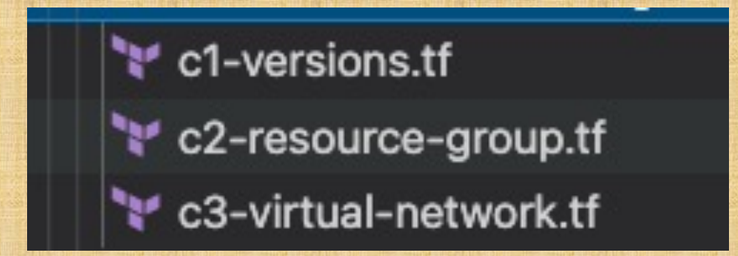
Author: Nho Luong

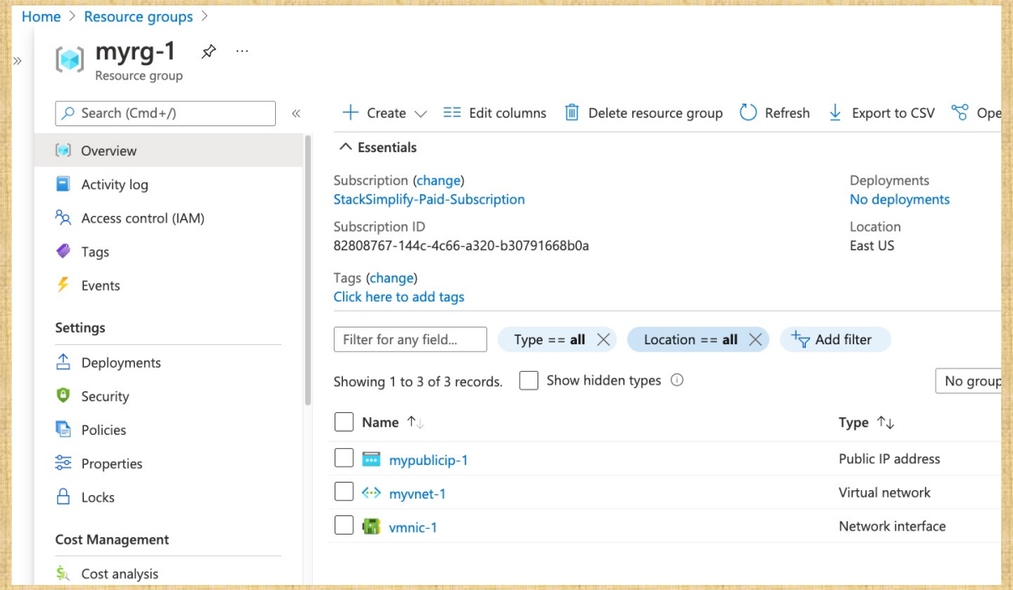
Skill: DevOps Engineer Lead

Desired & Current Terraform States

Terraform Configuration Files

Real World Resources





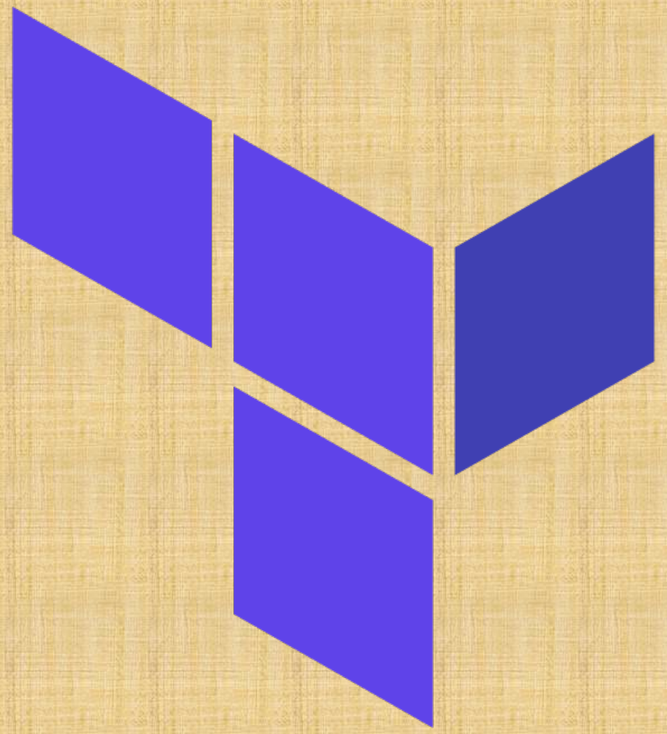
Desired State

Current State

Author: Nho Luong

Skill: DevOps Engineer Lead

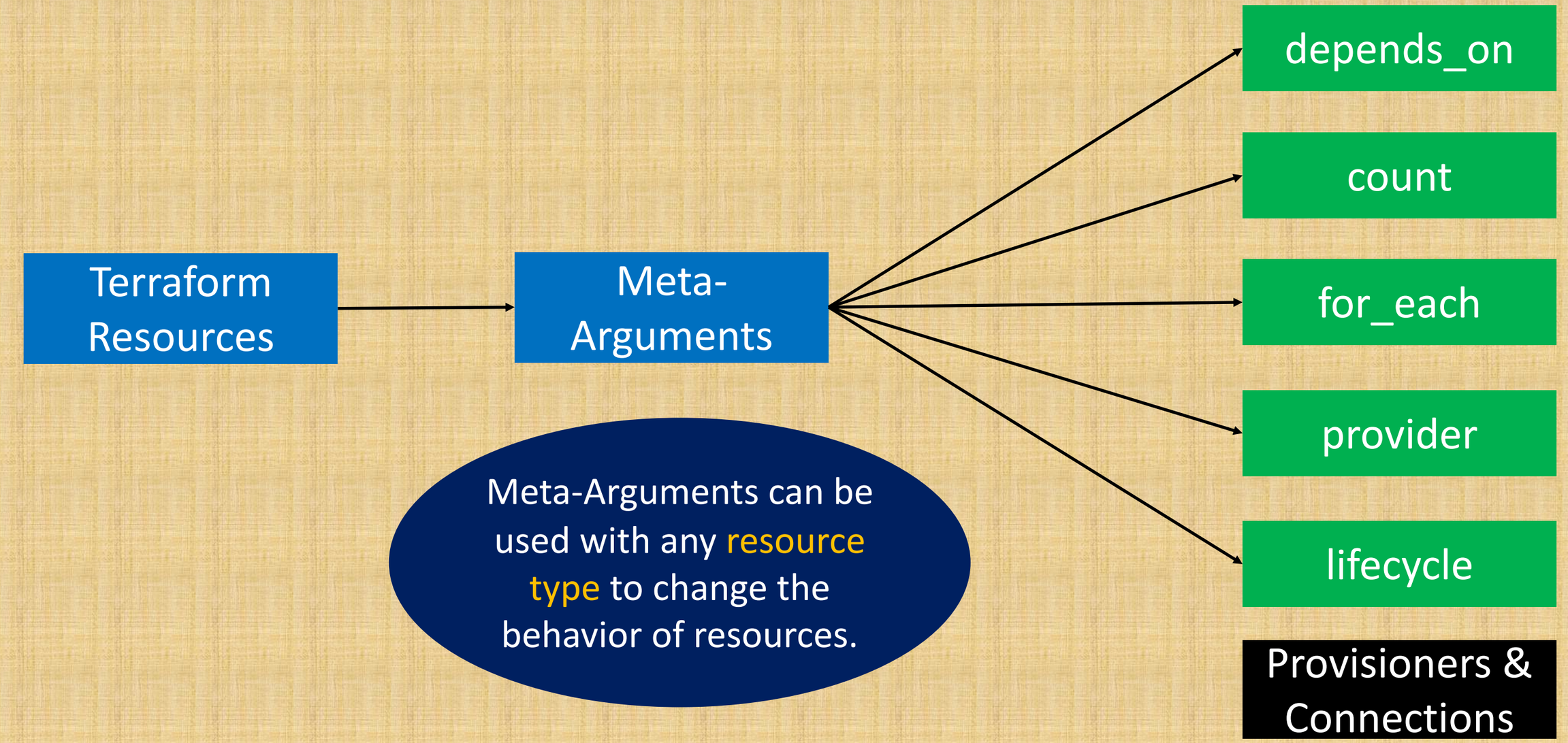
Terraform
Resource
Meta-Arguments



Author: Nho Luong

Skill: DevOps Engineer Lead

Resource Meta-Arguments



Author: Nho Luong

Skill: DevOps Engineer Lead

Practical Example with Step-by-Step Documentation on Github

Demos for Meta-Arguments

- 10-Meta-Argument-depends_on
- 11-01-Terraform-Azure-Linux-Virtual-Machine
- 11-02-Meta-Argument-count
- 12-Meta-Argument-for_each-Maps
- 13-Meta-Argument-for_each-ToSet
- 14-Meta-Argument-for_each-Chaining
- 15-Meta-Argument-lifecycle-create_before_destroy
- 16-Meta-Argument-lifecycle-prevent_destroy
- 17-Meta-Argument-lifecycle-ignore_changes

Terraform Meta-Argument depends_on	3 lectures • 21min
Provision Azure Linux VM using Terraform, file and filebase64 functions	4 lectures • 25min
Terraform Meta-Argument count with Element Function and Splat Expression	6 lectures • 31min
Terraform Meta-Argument for_each with Maps, Set of Strings and Chaining	6 lectures • 39min
Meta-Argument lifecycle create_before_destroy, prevent_destroy and ignore_change	6 lectures • 34min

Author: Nho Luong

Skill: DevOps Engineer Lead

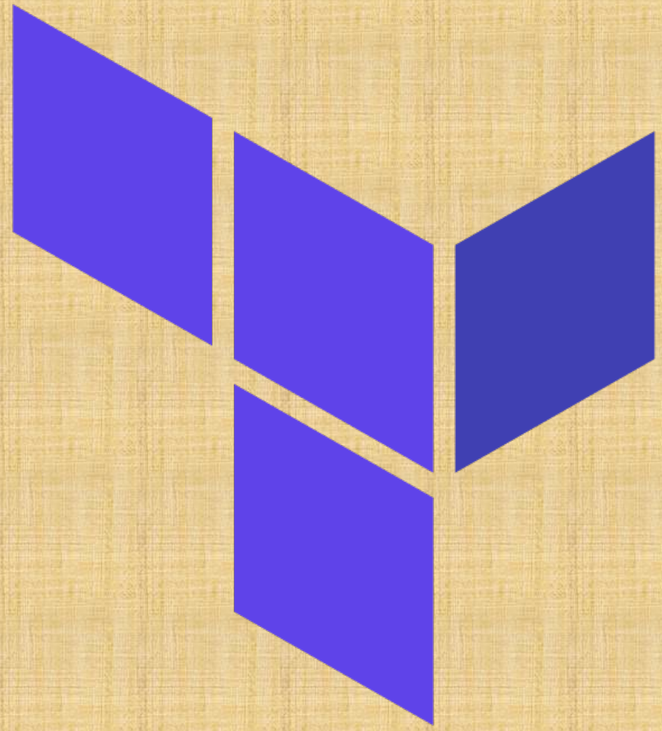
Resource Meta-Arguments

depends_on	To handle hidden resource or module dependencies that Terraform can't automatically infer.
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	Standard Resource behavior can be altered using special nested lifecycle block within a resource block body
Provisioners & Connections	For taking extra actions after resource creation (Example: install some app on server or do something on local desktop after resource is created at remote destination)

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Resource
Meta-Argument
depends_on



Author: Nho Luong

Skill: DevOps Engineer Lead

Resource Meta-Arguments – depends_on

Use the depends_on meta-argument to handle hidden resource or module dependencies that Terraform can't automatically infer.

Explicitly specifying a dependency is only necessary when a resource or module relies on some other resource's behavior but *doesn't* access any of that resource's data in its arguments.

This argument is available in module blocks and in all resource blocks, regardless of resource type.

Resource Meta-Argument depends_on

The depends_on meta-argument, if present, must be a list of references to other resources or child modules in the same calling module.

Arbitrary expressions are not allowed in the depends_on argument value, because its value must be known before Terraform knows resource relationships and thus before it can safely evaluate expressions.

The depends_on argument should be used only as a last resort. Add comments for future reference about why we added this.

Author: Nho Luong

Skill: DevOps Engineer Lead

Use case: What are we going implement?

Resource-1: Create Random String Resource

Resource-2: Create Azure Resource Group

Resource-3: Create Azure Virtual Network

Resource-4: Create Azure Subnet

Resource-5: Create Azure Public IP

Resource-6: Create Azure Network Interface

Resource-5: Create Azure Public IP only after VNET and Subnet got created using depends_on

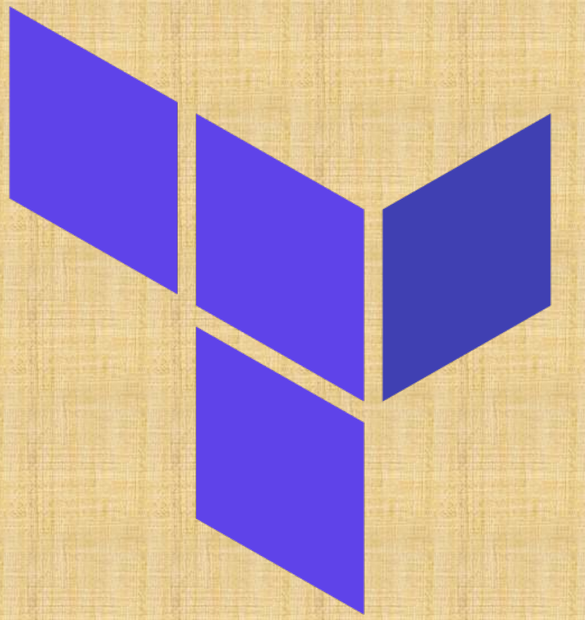
```
# Create Public IP Address
resource "azurerm_public_ip" "mypublicip" {
  # Add Explicit Dependency to have this resource created only
  depends_on = [
    azurerm_virtual_network.myvnet,
    azurerm_subnet.mysubnet
  ]
  name                = "mypublicip-1"
  resource_group_name = azurerm_resource_group.myrg.name
  location            = azurerm_resource_group.myrg.location
  allocation_method   = "Static"
  domain_name_label = "app1-vm-${random_string.myrandom.id}"
  tags = {
    environment = "Dev"
  }
}
```

Explicitly specifying a dependency is only necessary when a resource or module relies on some other resource's behavior but *doesn't* access any of that resource's data in its arguments.

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Resource
Meta-Argument
count



Author: Nho Luong

Skill: DevOps Engineer Lead

Resource Meta-Arguments – count

If a resource or module block includes a count argument whose value is a whole number, Terraform will create that many instances.

count.index: The distinct index number (starting with 0) corresponding to this instance.

Each instance has a distinct infrastructure object associated with it, and each is separately created, updated, or destroyed when the configuration is applied.

Resource Meta-Argument count

When count is set, Terraform distinguishes between the block itself and the multiple resource or module instances associated with it. Instances are identified by an index number, starting with 0.

The count meta-argument accepts numeric expressions. The count value must be known *before* Terraform performs any remote resource actions.

Module support for count was added in Terraform 0.13, and previous versions can only use it with resources.

A given resource or module block cannot use both count and for_each

Author: Nho Luong

Skill: DevOps Engineer Lead

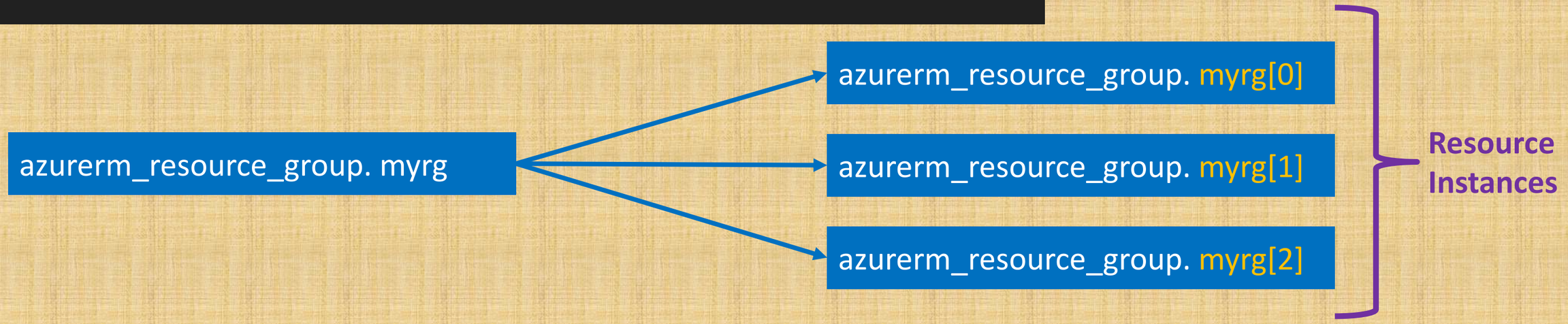
Use case: What are we going implement?

Resource-1: Use Meta-Argument Count to create multiple Resource Groups using single Resource

```
# Resource-1: Azure Resource Group
resource "azurerm_resource_group" "myrg" {
  count = 3
  name = "myrg-${count.index}"
  location = "East US"
}
```

count

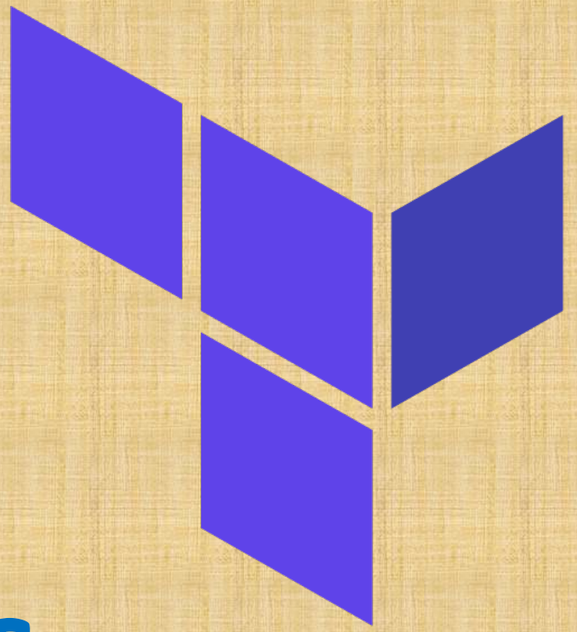
count.index



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Resource
Meta-Argument
for_each with Maps



Author: Nho Luong

Skill: DevOps Engineer Lead

Resource Meta-Arguments – for_each

If a resource or module block includes a for_each argument whose value is a map or a set of strings, Terraform will create one instance for each member of that map or set.

A given resource or module block cannot use both count and for_each

For set of Strings, each.key = each.value
for_each = toset(["Jack", "James"])
each.key = Jack
each.key = James

Each instance has a distinct infrastructure object associated with it, and each is separately created, updated, or destroyed when the configuration is applied.

Resource Meta-Argument for_each

For Maps, we use each.key & each.value
for_each = {
dev = ”myapp1"
}
each.key = dev
each.value = myapp1

In blocks where for_each is set, an additional each object is available in expressions, so you can modify the configuration of each instance.
each.key — The map key (or set member) corresponding to this instance.
each.value — The map value corresponding to this instance. (If a set was provided, this is the same as each.key.)

Module support for for_each was added in Terraform 0.13, and previous versions can only use it with resources.

Author: Nho Luong

Skill: DevOps Engineer Lead

Usecase-1: for_each Maps

Resource-1: Use Meta-Argument for_each with Maps to create multiple Resource Groups using single Resource

Define for_each with Map with Key Value pairs

Use each.key for Resource Group Name

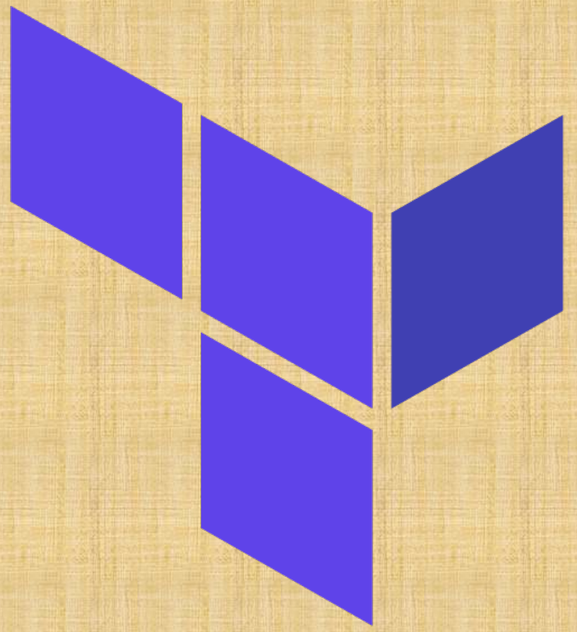
Use each.value for Resource Group Location

```
# Resource-1: Azure Resource Group
resource "azurerm_resource_group" "myrg" {
  for_each = {
    dc1apps = "eastus"
    dc2apps = "eastus2"
    dc3apps = "westus"
  }
  name = "${each.key}-rg"
  location = each.value
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Resource
Meta-Argument
for_each with Set of Strings & toset function



Author: Nho Luong

Skill: DevOps Engineer Lead

Usecase-2: for_each Set of Strings (toset)

Resource-1: Use Meta-Argument for_each with Set of Strings to create multiple Resource Groups using single Resource

Define for_each with Set of Strings

Use each.value for Resource Group Name

Use each.key for Resource Group Location

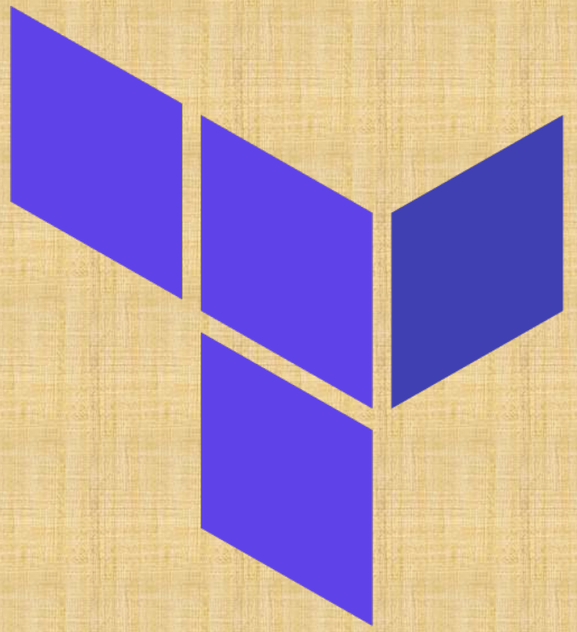
each.key == each.value

```
# Resource-1: Azure Resource Group
resource "azurerm_resource_group" "myrg" {
  for_each = toset(["eastus", "eastus2", "westus"])
  name = "myrg-${each.value}"
  location = each.key
}
/*
we can also use each.value as each.key = each.value
in this case
*/
```

Author: Nho Luong

Skill: DevOps Engineer Lead

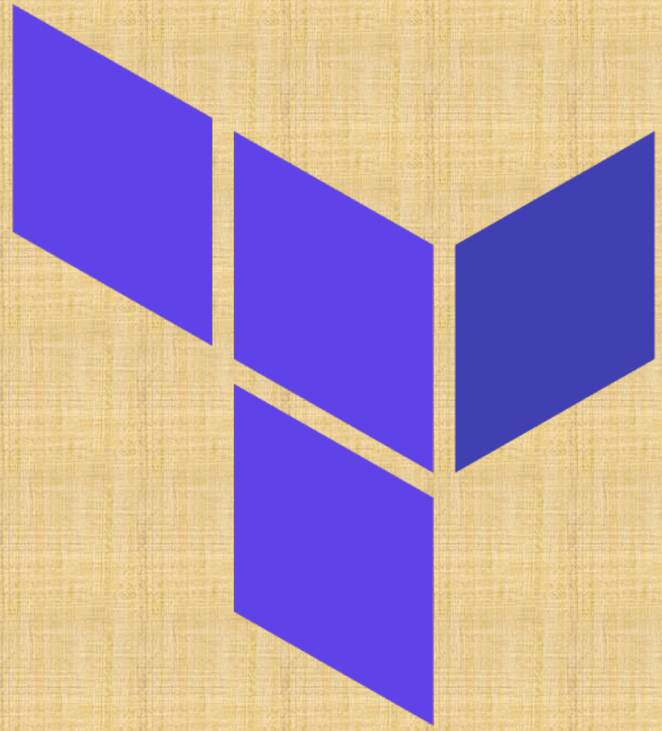
Terraform
Resource
Meta-Argument
for_each chaining



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Resource
Meta-Argument
lifecycle



Author: Nho Luong

Skill: DevOps Engineer Lead

lifecycle is a nested block that can appear within a resource block

Resource Meta-Argument lifecycle

The lifecycle block and its contents are meta-arguments, available for all resource blocks regardless of type.

create_before_destroy

```
# Lifecycle Changes
  lifecycle {
    create_before_destroy = true
  }
```

prevent_destroy

```
# Lifecycle Changes
  lifecycle {
    prevent_destroy = true
  }
```

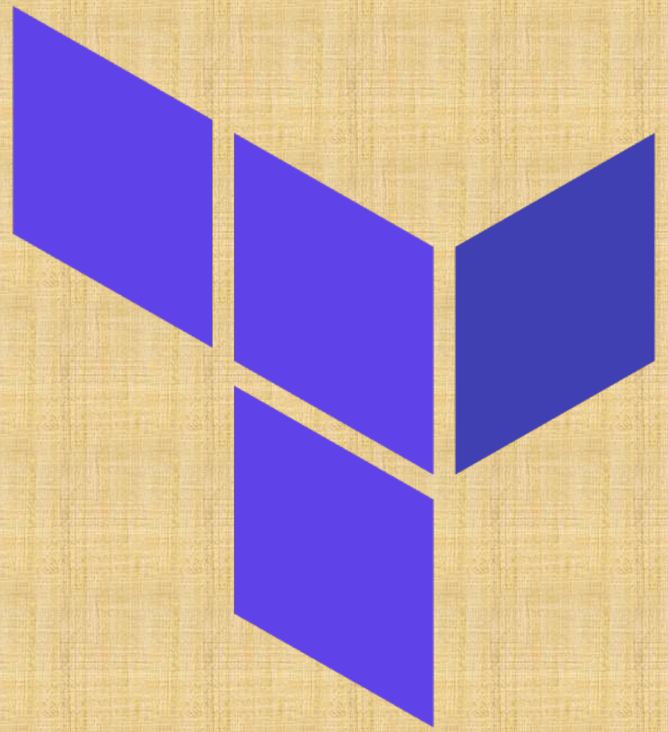
ignore_changes

```
# Lifecycle Changes
lifecycle {
  ignore_changes = [
    tags,
  ]
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Variables
Input Variables



Author: Nho Luong

Skill: DevOps Engineer Lead

Input variables serve as parameters for a Terraform module, allowing aspects of the module to be customized without altering the module's own source code, and allowing modules to be shared between different configurations.

Terraform Input Variables

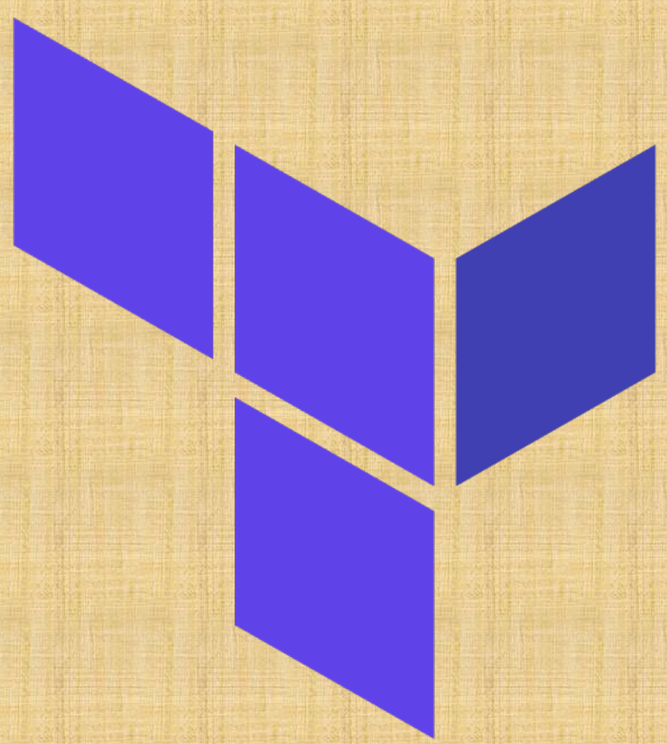
Demos

1. Input Variables - Basics
2. Provide Input Variables when prompted during terraform plan or apply
3. Override default variable values using CLI argument -var
4. Override default variable values using Environment Variables (TF_var_aa)
5. Provide Input Variables using terraform.tfvars files
6. Provide Input Variables using <any-name>.tfvars file with CLI argument -var-file
7. Provide Input Variables using somefilename.auto.tfvars files
8. Implement complex type constructors like List & Map in Input Variables
9. Implement Custom Validation Rules in Variables
10. Protect Sensitive Input Variables
11. Input Variables using Structural Type: Object
12. Input Variables using Structural Type: tuple
13. Input Variables using Collection Type: set

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Variables
Output Values



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Variables – Output Values

Output values are like the return values of a Terraform module and have several uses

Terraform Variables Outputs

1

A root module can use outputs to print certain values in the CLI output after running terraform apply.

2

A child module can use outputs to expose a subset of its resource attributes to a parent module.

Advanced

3

When using remote state, root module outputs can be accessed by other configurations via a terraform_remote_state d ata source.

Author: Nho Luong

Skill: DevOps Engineer Lead

Output Values – 3 Demos

Demo-1

Demo-2

Demo-3

Basics

Count and Splat Expression

for_each and for loops

Over the process master the for loops in Terraform with Lists and Maps

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Variables
Output Values
Basics

```
# 1. Output Values for Resource Group Resource
output "resource_group_id" {
  description = "Resource Group ID"
  # Attribute Reference
  value = azurerm_resource_group.myrg.id
}
output "resource_group_name" {
  description = "Resource Group Name"
  # Argument Reference
  value = azurerm_resource_group.myrg.name
}

# 2. Output Values for Virtual Network Resource
output "virtual_network_name" {
  description = "Virtal Network Name"
  value = azurerm_virtual_network.myvnet.name
  #sensitive = true  # Enable during Step-08
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Variables
Output Values
Count and Splat Expression

```
# 1. Output Values - Resource Group
output "resource_group_id" {
  description = "Resource Group ID"
  # Atrribute Reference
  value = azurerm_resource_group.myrg.id
}
output "resource_group_name" {
  description = "Resource Group name"
  # Argument Reference
  value = azurerm_resource_group.myrg.name
}

# 2. Output Values - Virtual Network
output "virtual_network_name" {
  description = "Virutal Network Name"
  #value = azurerm_virtual_network.myvnet.name
  value = azurerm_virtual_network.myvnet[*].name
  #sensitive = true
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Variables Output Values for_each and for loops with Lists

```
# Output - For Loop One Input and List Output with VNET Name
output "virtual_network_name_list_one_input" {
  description = "Virtual Network - For Loop One Input and List Output with VNET Name "
  value = [for vnet in azurerm_virtual_network.myvnet: vnet.name ]
}

# Output - For Loop Two Inputs, List Output which is Iterator i (var.environment)
output "virtual_network_name_list_two_inputs" {
  description = "Virtual Network - For Loop Two Inputs, List Output which is Iterator i (
  #value = [for i, vnet in azurerm_virtual_network.myvnet: i ]
  value = [for env, vnet in azurerm_virtual_network.myvnet: env ]
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Variables Output Values
for_each and for loops with Maps

```
# Output - For Loop One Input and Map Output with VNET ID and VNET Name
output "virtual_network_name_map_one_input" {
  description = "Virtual Network - For Loop One Input and Map Output with VNET ID
  value = {for vnet in azurerm_virtual_network.myvnet: vnet.id => vnet.name }
}

# Output - For Loop Two Inputs and Map Output with Iterator env and VNET Name
output "virtual_network_name_map_two_inputs" {
  description = "Virtual Network - For Loop Two Inputs and Map Output with Iterat
  value = {for env, vnet in azurerm_virtual_network.myvnet: env => vnet.name }
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

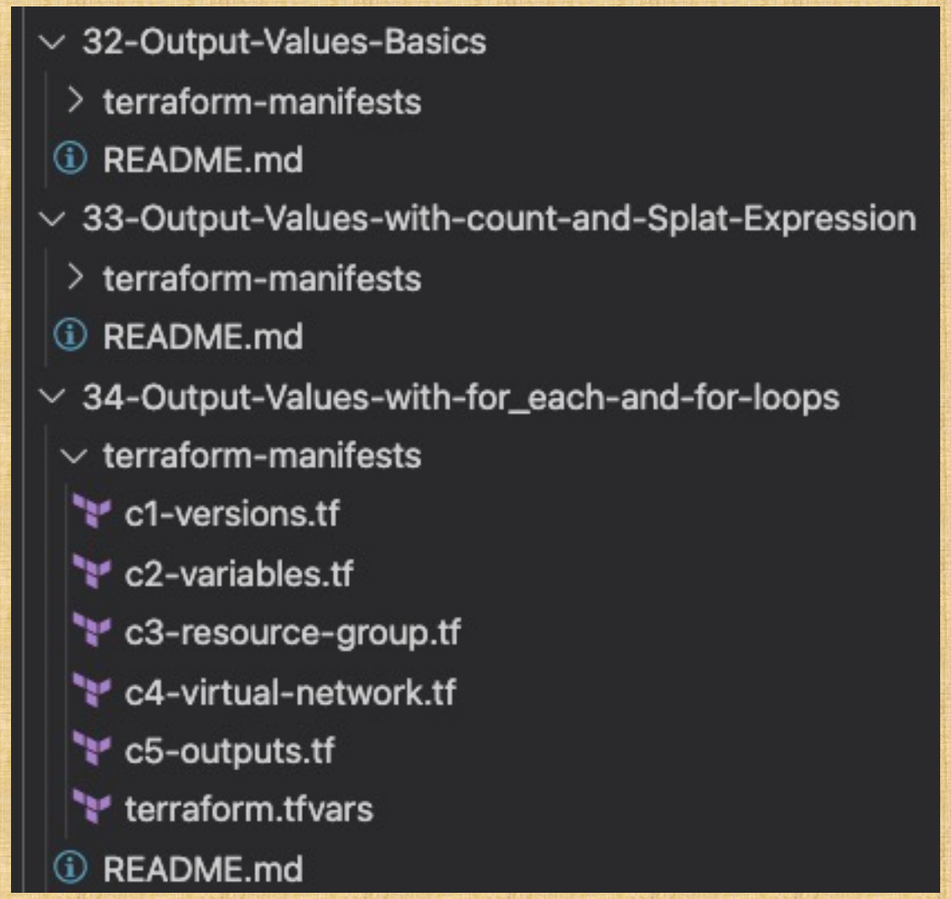
Terraform Variables Output Values
for_each and for loops with keys() and values() functions

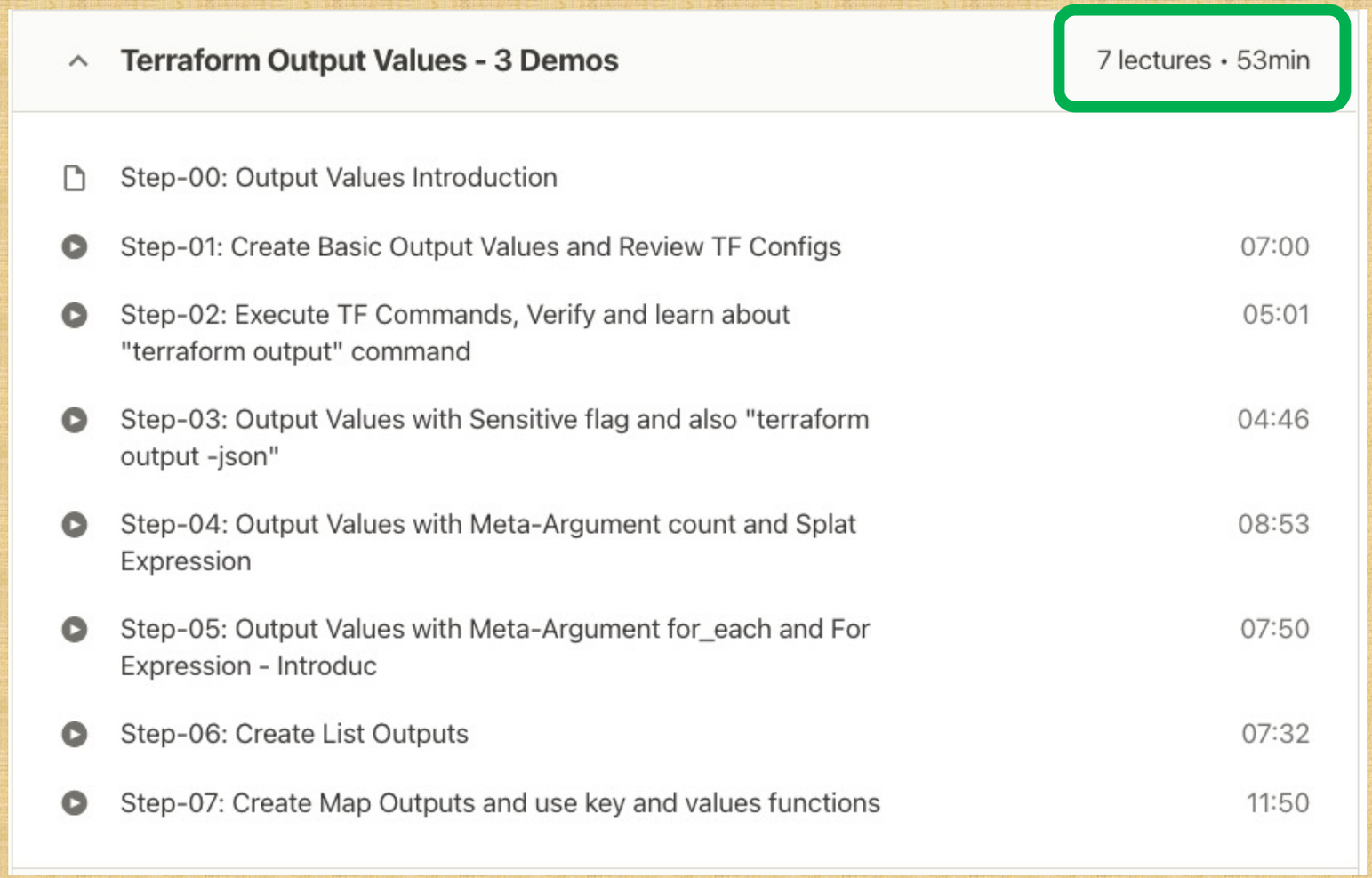
```
# Terraform keys() function: keys takes a map and returns a list containing the keys from
output "virtual_network_name_keys_function" {
  description = "Virtual Network - Terraform keys() function"
  value = keys({for env, vnet in azurerm_virtual_network.myvnet: env => vnet.name })
}

# Terraform values() function: values takes a map and returns a list containing the values
output "virtual_network_name_values_function" {
  description = "Virtual Network - Terraform values() function"
  value = values({for env, vnet in azurerm_virtual_network.myvnet: env => vnet.name })
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Practical Example with Step-by-Step Documentation on Github

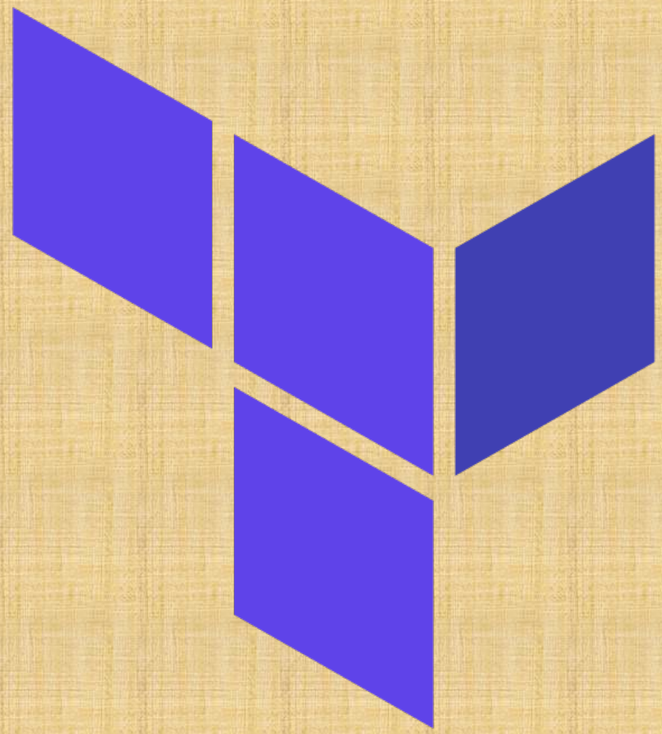




Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Variables
Local Values



Author: Nho Luong

Skill: DevOps Engineer Lead

DRY Principle

Don’t Repeat Yourself

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Variables – Local Values

A local value assigns a name to an expression, so you can use that name multiple times within a module without repeating it.

Local values are like a function's temporary local variables.

Once a local value is declared, you can reference it in expressions as local.<NAME>.

Local values can be helpful to avoid repeating the same values or expressions multiple times in a configuration

If overused they can also make a configuration hard to read by future maintainers by hiding the actual values used

The ability to easily change the value in a central place is the key advantage of local values.

In short, Use local values only in moderation

```
locals {
  service_name = "forum"
  owner        = "Community Team"
}
```

```
locals {
  # Common tags to be assigned to all resources
  common_tags = {
    Service = local.service_name
    Owner   = local.owner
  }
}
```

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

  tags = local.common_tags
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

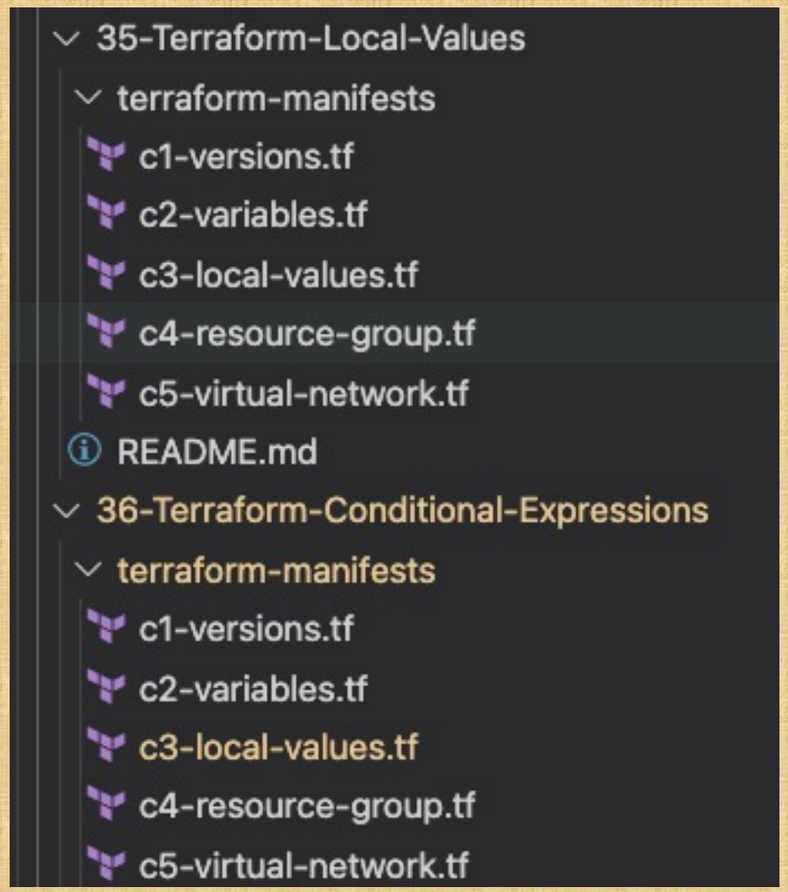
Terraform Variables – Local Values

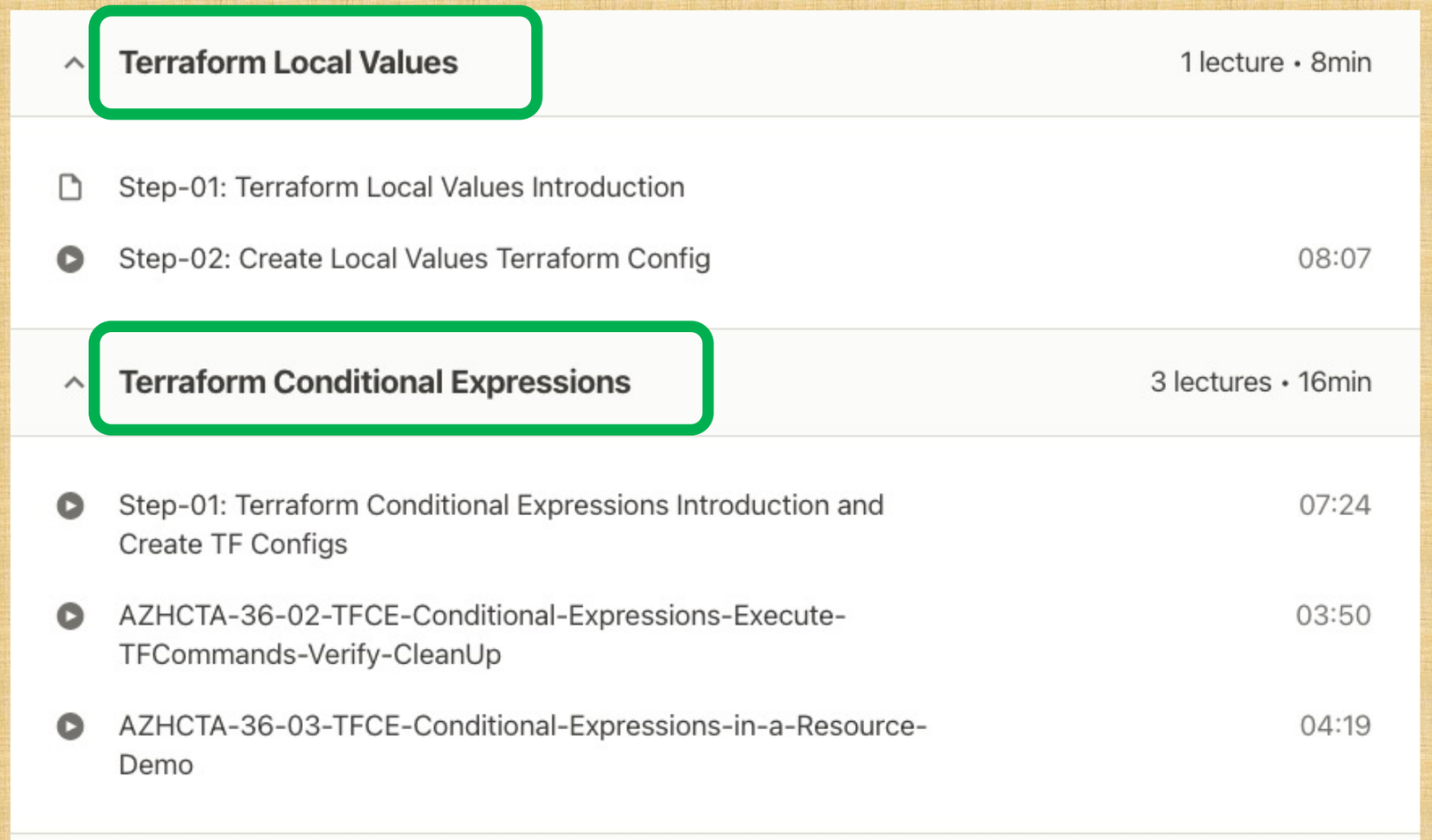
```
# Use-case-3: Terraform Conditional Expressions
# We will learn this when we are dealing with Conditional Expressions
# The expressions assigned to local value names can either be simple constants or can be mo
# Option-1: With Equals (==)
vnet_address_space = (var.environment == "dev" ? var.vnet_address_space_dev : var.vnet_addr
# Option-2: With Not Equals (!=)
#vnet_address_space = (var.environment != "dev" ? var.vnet_address_space_all : var.vnet_add
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Practical Example with Step-by-Step Documentation on Github

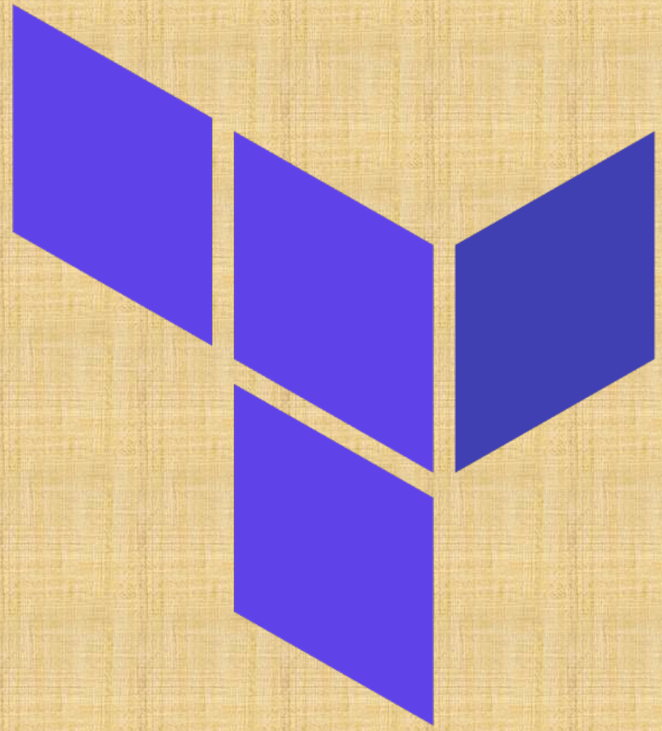




Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Datasources



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Datasources

Data sources allow data to be fetched or computed for use elsewhere in Terraform configuration.

Use of data sources allows a Terraform configuration to make use of information defined outside of Terraform, or defined by another separate Terraform configuration.

A data source is accessed via a special kind of resource known as a *data resource*, declared using a data block

Each data resource is associated with a single data source, which determines the kind of object (or objects) it reads and what query constraint arguments are available

Data resources have the same dependency resolution behavior as defined for managed resources. Setting the depends_on meta-argument within data blocks defers reading of the data source until after all changes to the dependencies have been applied.

```
# Datasources
# https://registry.terraform.io/providers/hashicorp/azurerm
data "azurerm_subscription" "current" {

}

## TEST DATASOURCES using OUTPUTS
# 1. My Current Subscription Display Name
output "current_subscription_display_name" {
  value = data.azurerm_subscription.current.display_name
}

# 2. My Current Subscription Id
output "current_subscription_id" {
  value = data.azurerm_subscription.current.subscription_id
}

# 3. My Current Subscription Spending Limit
output "current_subscription_spending_limit" {
  value = data.azurerm_subscription.current.spending_limit
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Datasources

Meta-Arguments for Datasources

Data resources support the provider meta-argument as defined for managed resources, with the same syntax and behavior.

Data resources do not currently have any customization settings available for their lifecycle, but the lifecycle nested block is reserved in case any are added in future versions.

Data resources support count and for_each meta-arguments as defined for managed resources, with the same syntax and behavior.
Each instance will separately read from its data source with its own variant of the constraint arguments, producing an indexed result.

Author: Nho Luong

Skill: DevOps Engineer Lead

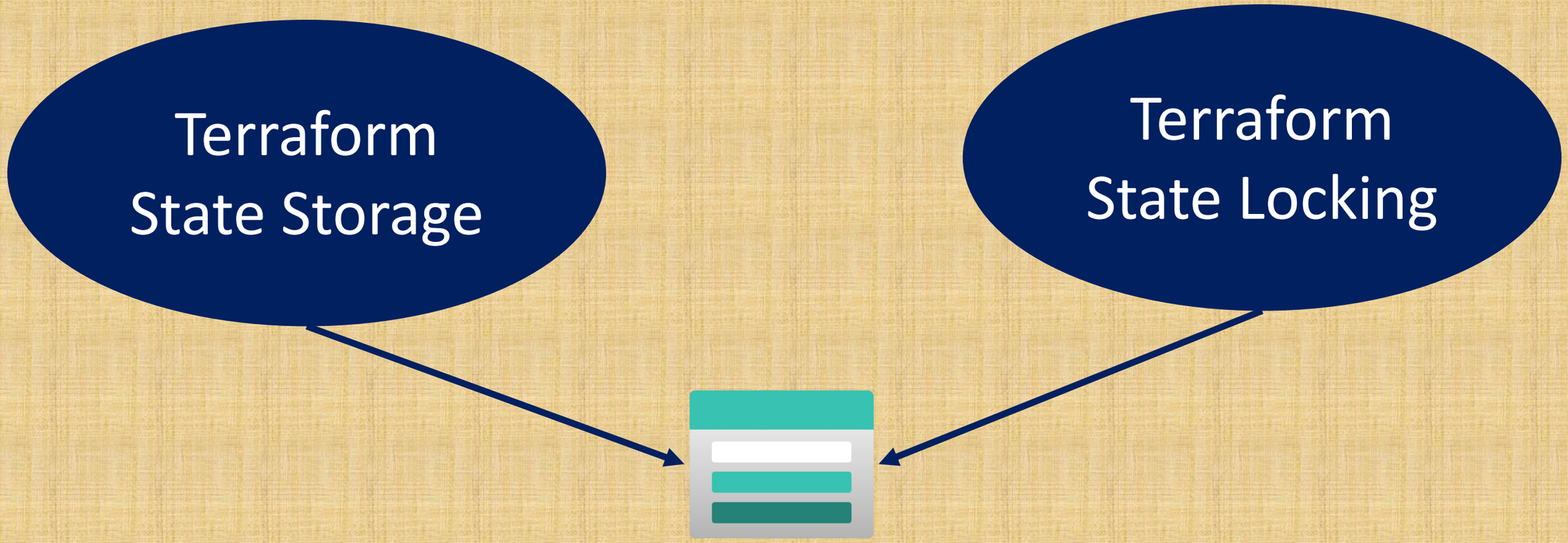
Terraform
Backend
Remote State Storage

Author: Nho Luong

Skill: DevOps Engineer Lead

What is Terraform Backend ?

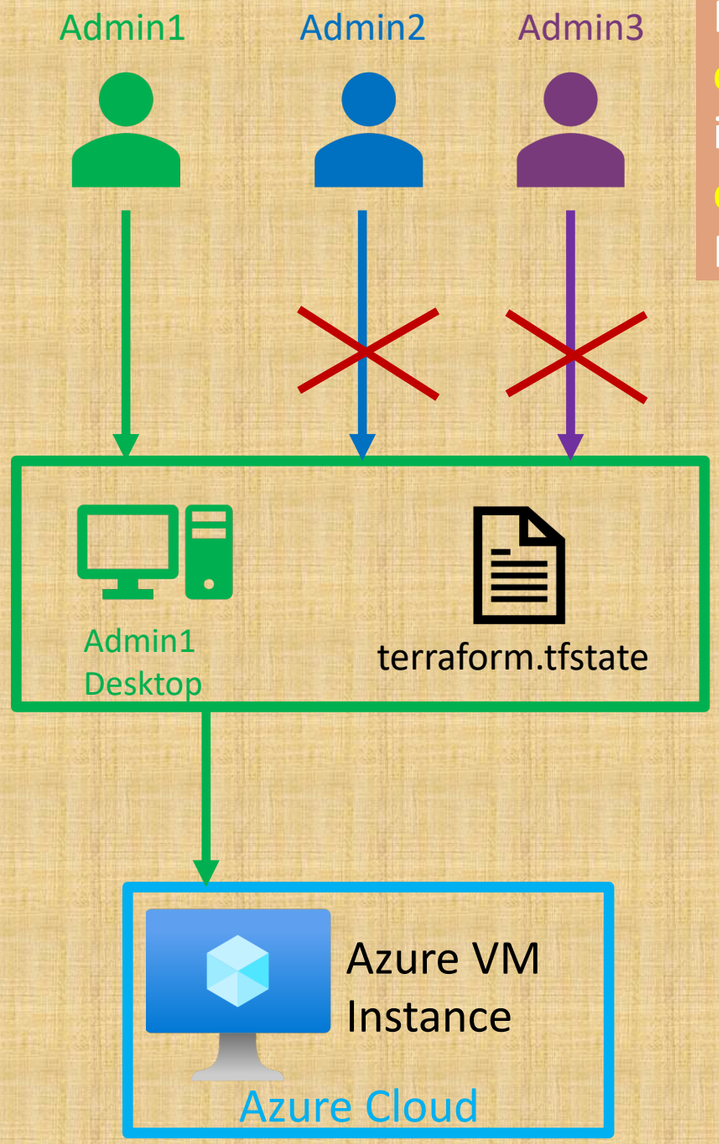
Backends are responsible for storing state and providing an API for state locking.



Author: Nho Luong

Skill: DevOps Engineer Lead

Local State File



Multiple Team members cannot update the infrastructure as they don't have access to State File

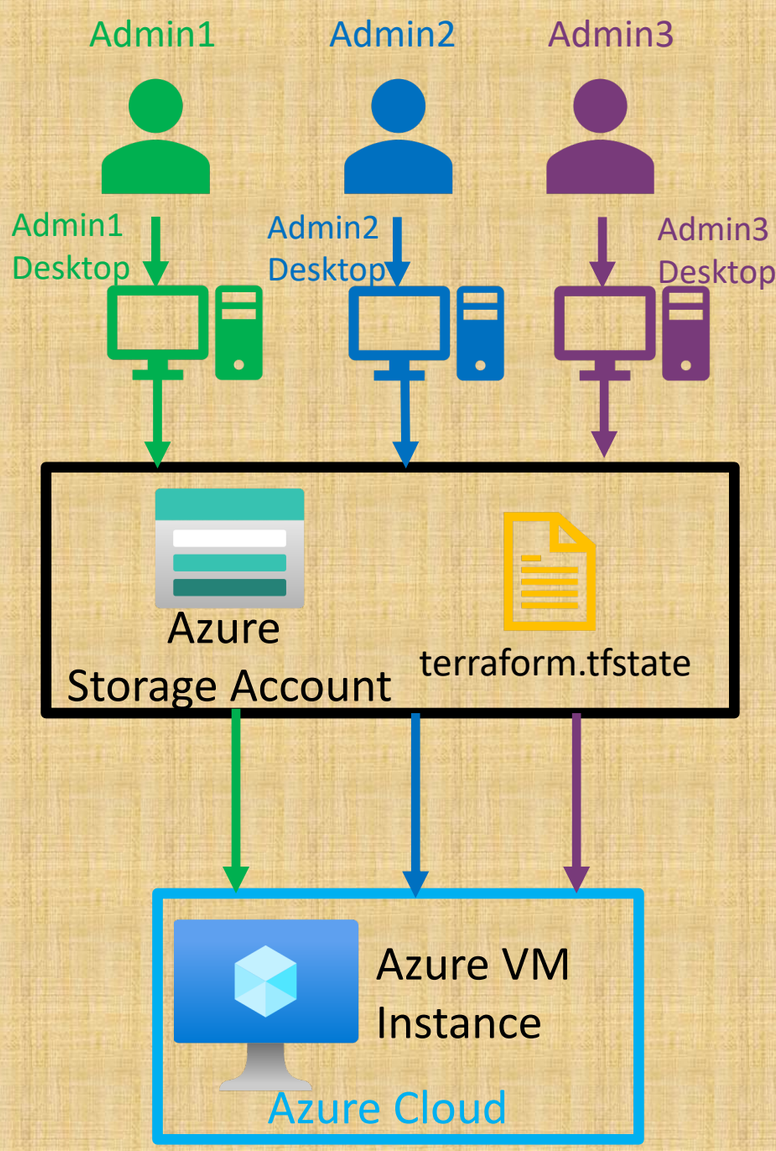
This means we need store the state file in a shared location.

Remote State File

Using Terraform Backend concept we can use Azure SA as the shared storage for State Files

If two team members are running Terraform at the same time, you may run into race conditions as multiple Terraform processes make concurrent updates to the state files, leading to conflicts, data loss, and state file corruption.

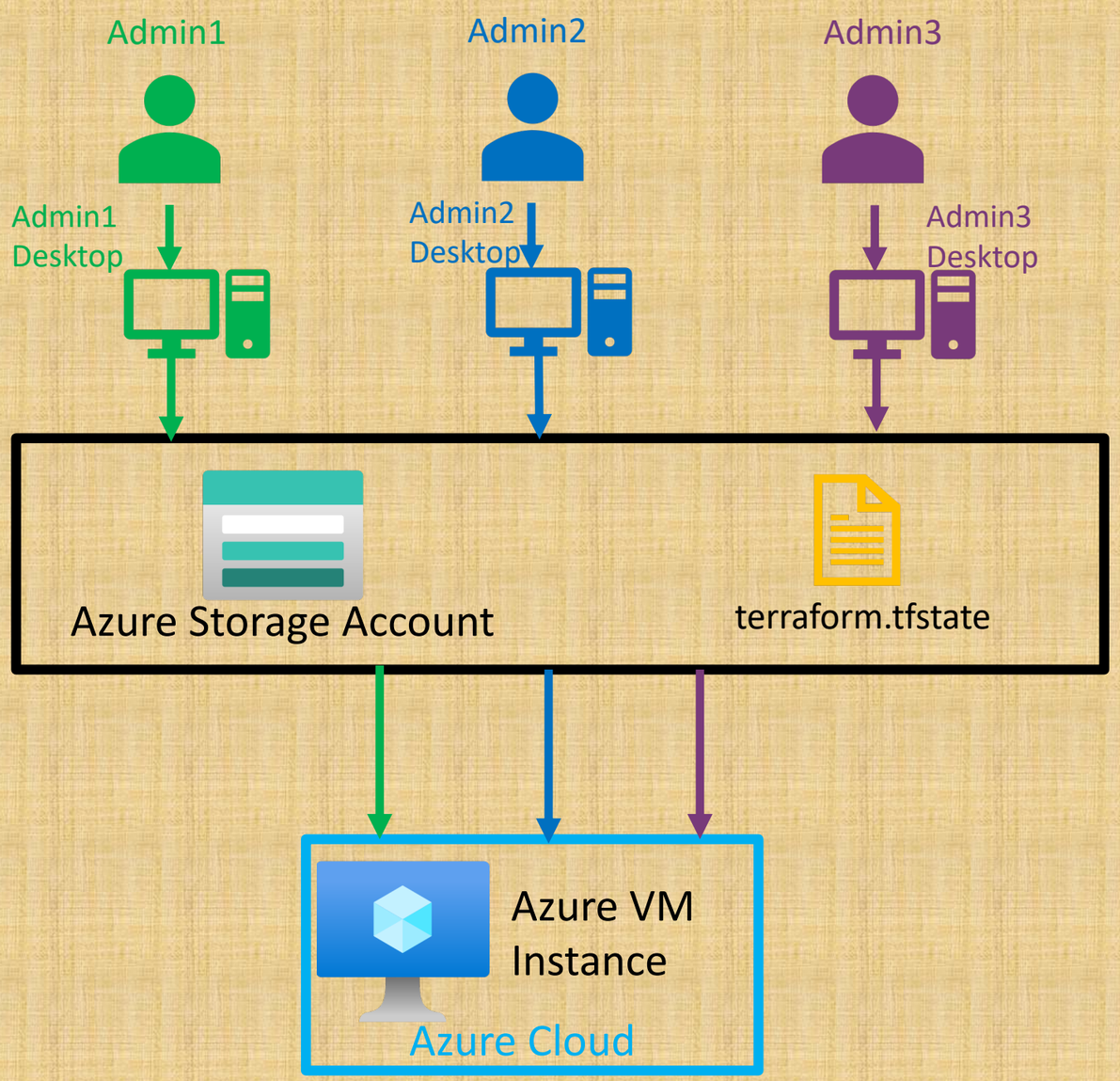
State Locking



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Remote State File with State Locking



Not all backends support State Locking. Azure Storage Account supports State Locking

State locking happens automatically on all operations that could write state.

If state locking fails, Terraform will not continue.

You can disable state locking for most commands with the -lock flag but it is not recommended.

If acquiring the lock is taking longer than expected, Terraform will output a status message.

If Terraform doesn't output a message, state locking is still occurring if your backend supports it.

Terraform has a force-unlock command to manually unlock the state if unlocking failed.

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Remote State File with State Locking

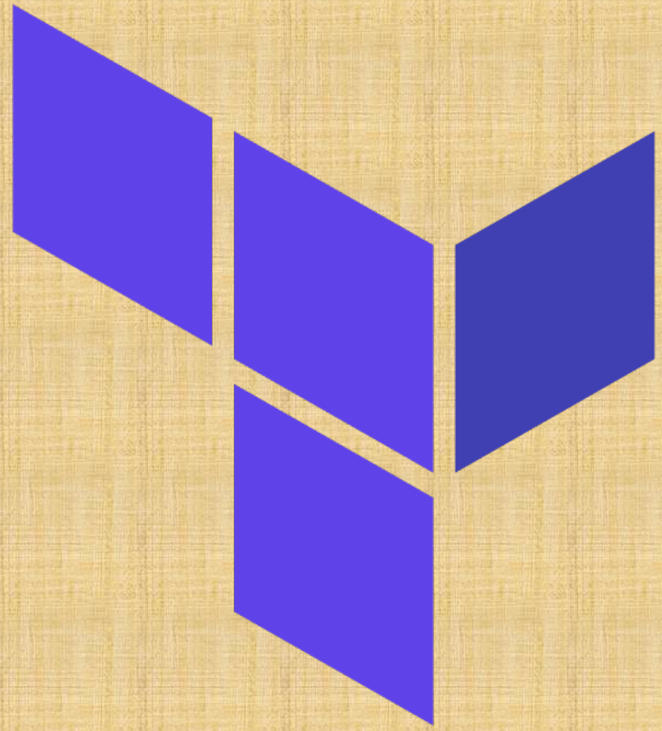
Terraform Remote State Storage

```
# Terraform Block
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    azurerm = {
      source = "hashicorp/azurerm"
      version = ">= 2.0"
    }
    random = {
      source = "hashicorp/random"
      version = ">= 3.0"
    }
  }
# Terraform State Storage to Azure Storage Container
  backend "azurerm" {
    resource_group_name   = "terraform-storage-rg"
    storage_account_name  = "terraformstate201"
    container_name        = "tfstatefiles"
    key                   = "terraform.tfstate"
  }
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Backends



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Backends

Each Terraform configuration can specify a backend, which defines where and how operations are performed, where state snapshots are stored, etc.

Where Backends are Used

Backend configuration is only used by Terraform CLI.

Terraform Cloud and Terraform Enterprise always use their own state storage when performing Terraform runs, so they ignore any backend block in the configuration.

For Terraform Cloud users also it is always recommended to use backend block in Terraform configuration for commands like terraform taint which can be executed only using Terraform CLI

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Backends

What Backends Do

There are two things backends will be used for

1. Where state is stored
2. Where operations are performed.

Store State

Terraform uses persistent state data to keep track of the resources it manages.

Everyone working with a given collection of infrastructure resources must be able to access the same state data (shared state storage).

State Locking

State Locking is to prevent conflicts and inconsistencies when the operations are being performed

What are Operations ?
terraform apply
terraform destroy

Operations

"Operations" refers to performing API requests against infrastructure services in order to create, read, update, or destroy resources.

Not every terraform subcommand performs API operations; many of them only operate on state data.

Only two backends actually perform operations: local and remote.

The remote backend can perform API operations remotely, using Terraform Cloud or Terraform Enterprise.

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Backends

Backend Types

Enhanced Backends

Enhanced backends can both store state and perform operations. There are only two enhanced backends: local and remote

Example for Remote Backend Performing Operations : Terraform Cloud, Terraform Enterprise

Standard Backends

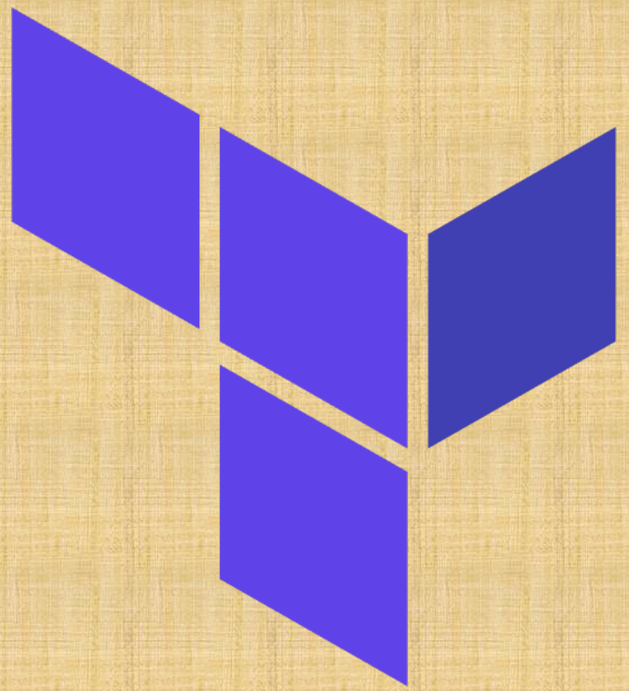
Standard backends only store state, and rely on the local backend for performing operations.

Example: AWS S3, Azure RM, Consul, etcd, gcs http and many more

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Remote State Datasource

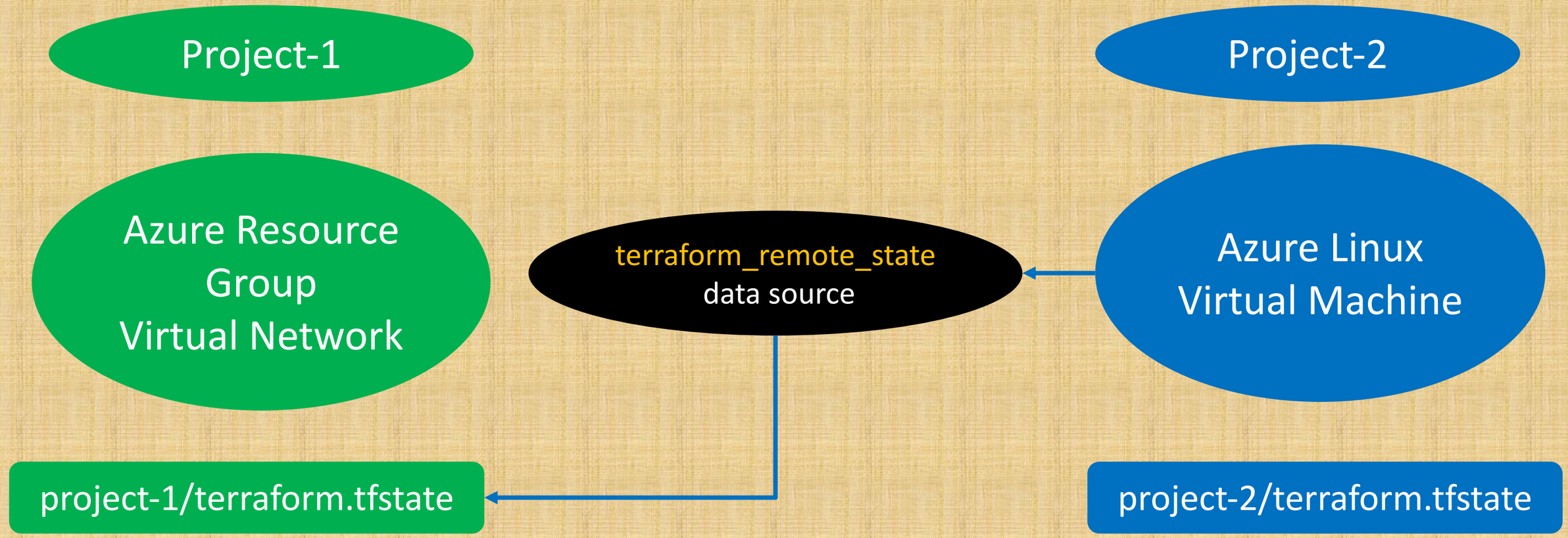


Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Remote State Datasource

The terraform_remote_state data source retrieves the root module output values from some other Terraform configuration, using the latest state snapshot from the remote backend.



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Backend – Both Projects

```
# Terraform State Storage to Azure Storage Container
  backend "azurerm" {
    resource_group_name   = "terraform-storage-rg"
    storage_account_name  = "terraformstate201"
    container_name        = "tfstatefiles"
    key                   = "network-terraform.tfstate"
  }
```

Project-1 Terraform State File

```
# Terraform State Storage to Azure Storage Container
  backend "azurerm" {
    resource_group_name   = "terraform-storage-rg"
    storage_account_name  = "terraformstate201"
    container_name        = "tfstatefiles"
    key                   = "app1-terraform.tfstate"
  }
```

Project-2 Terraform State File

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Remote State Datasource – Used in Project-2

```
# Terraform Remote State Datasource
data "terraform_remote_state" "project1" {
  backend = "azurerm"
  config = {
    resource_group_name   = "terraform-storage-rg"
    storage_account_name  = "terraformstate201"
    container_name        = "tfstatefiles"
    key                   = "network-terraform.tfstate"
  }
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Reference Project-1 Output Values in Project -2

c4-linux-virtual-machine.tf

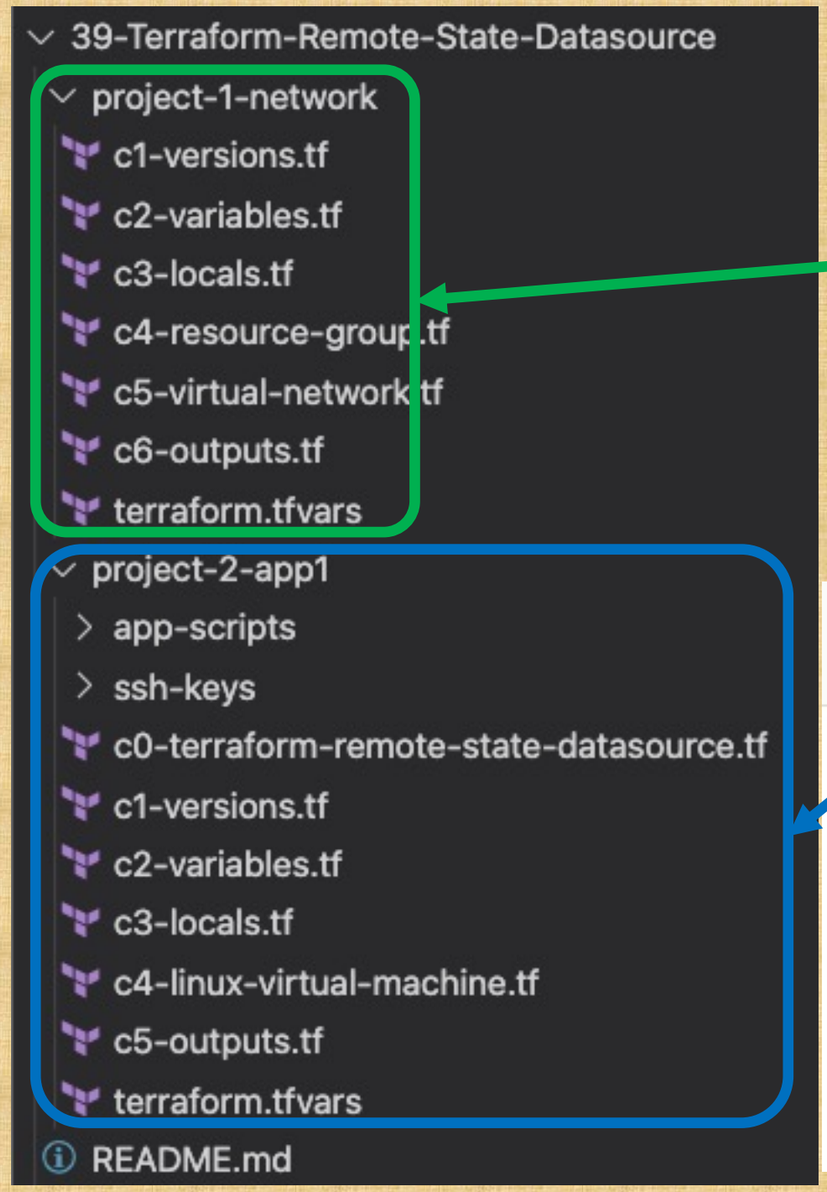
githubcontent > hashicorp-certified-terraform-associate-on-azure > 39-Terraform-Remote-State-Datasource > project-2-app1 > c4-linux-virtual-machine.tf > ...

```
11  resource_group_name = data.terraform_remote_state.project1.outputs.resource_group_name
12  location = data.terraform_remote_state.project1.outputs.resource_group_location
13  network_interface_ids = [data.terraform_remote_state.project1.outputs.network_interface_id]
14  admin_ssh_key {
15    username   = "azureuser"
16    public_key = file("${path.module}/ssh-keys/terraform-azure.pub")
17  }
18  os_disk {
19    name = "osdisk${random_string.myrandom.id}"
20    caching              = "ReadWrite"
21    storage_account_type = "Standard_LRS"
```

Author: Nho Luong

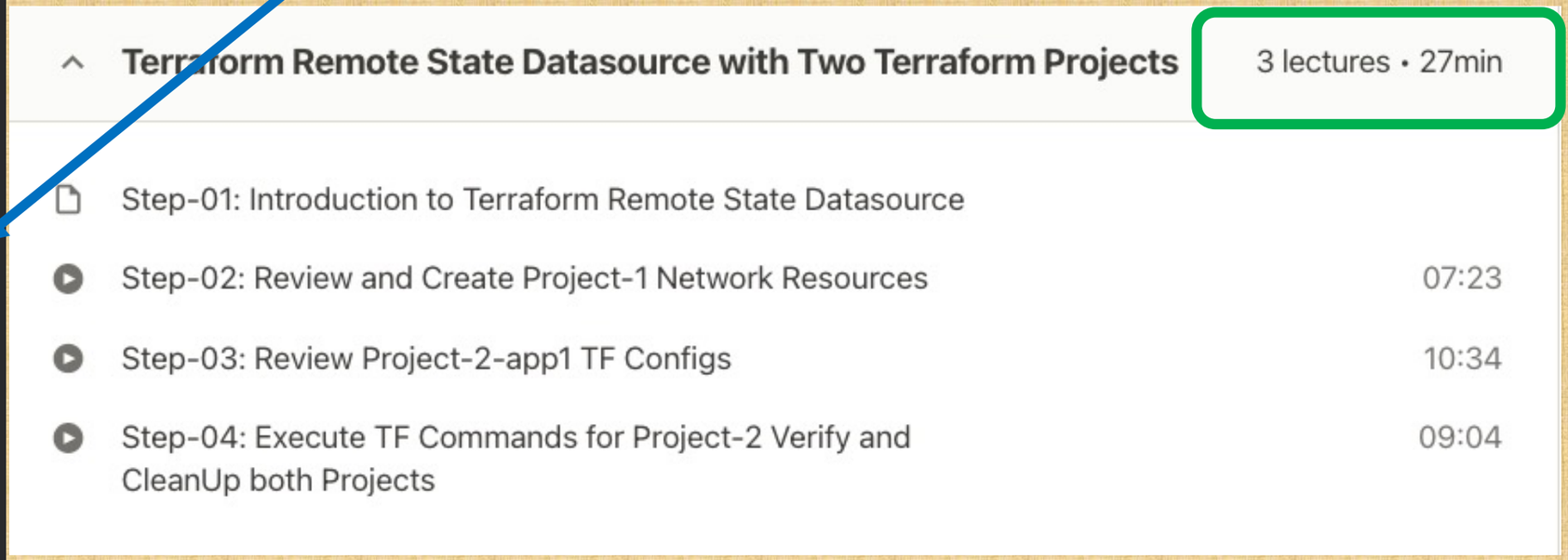
Skill: DevOps Engineer Lead

Practical Example with Step-by-Step Documentation on Github



Project-1 Terraform Configuration Files

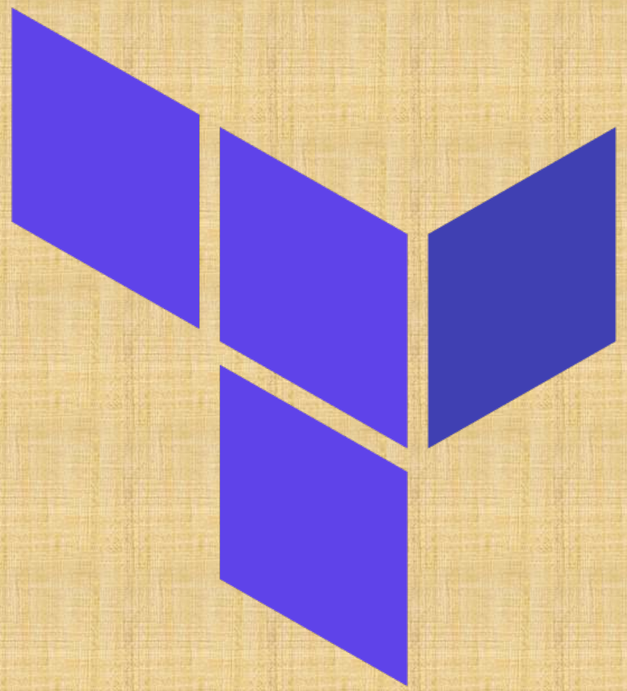
Project-2 Terraform Configuration Files



Author: Nho Luong

Skill: DevOps Engineer Lead

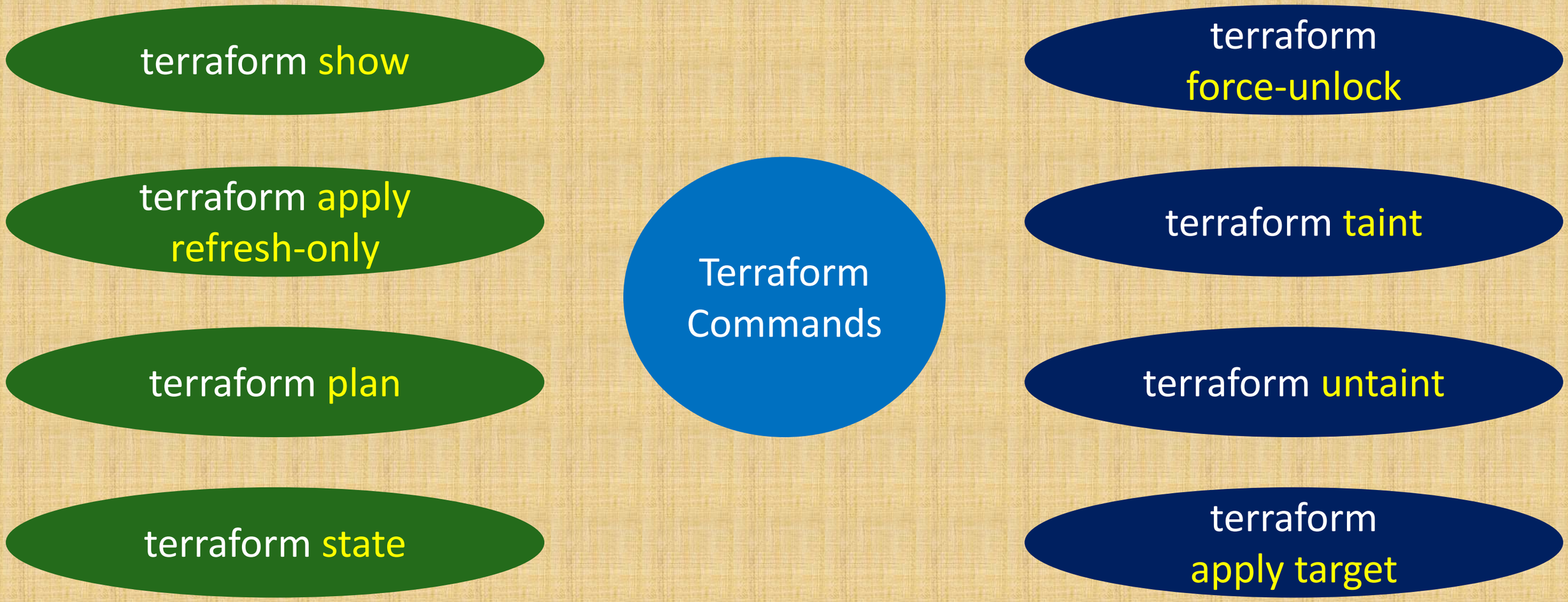
Terraform
State Commands



Author: Nho Luong

Skill: DevOps Engineer Lead

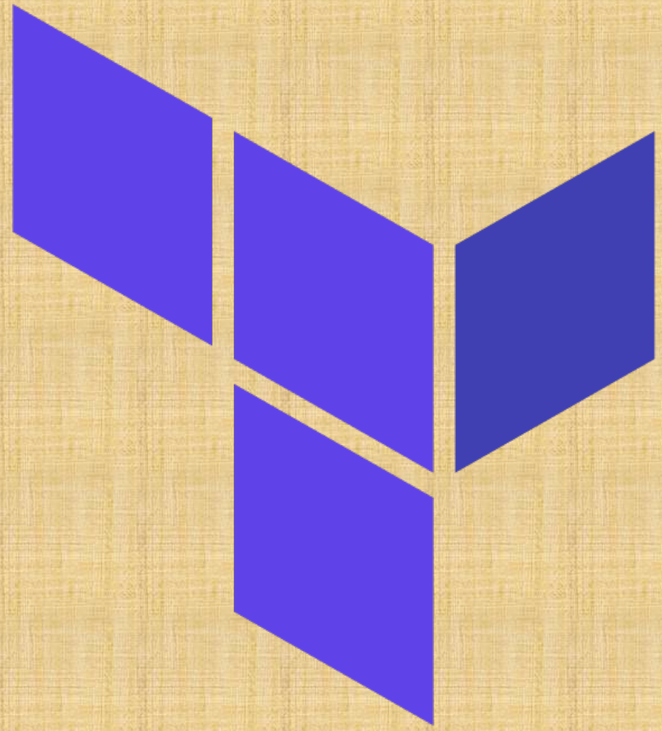
Terraform Commands – State Perspective



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Workspaces



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Workspaces – CLI based

Terraform starts with a single workspace named "default"

This workspace is special both because it is the default and also because it cannot ever be deleted.

By default, we are working in default workspace

Named workspaces allow conveniently switching between multiple instances of a *single* configuration within its *single* backend.

They are convenient in a number of situations, but cannot solve all problems.

A common use for multiple workspaces is to create a parallel, distinct copy of a set of infrastructure in order to test a set of changes before modifying the main production infrastructure.

For example, a developer working on a complex set of infrastructure changes might create a new temporary workspace in order to freely experiment with changes without affecting the default workspace

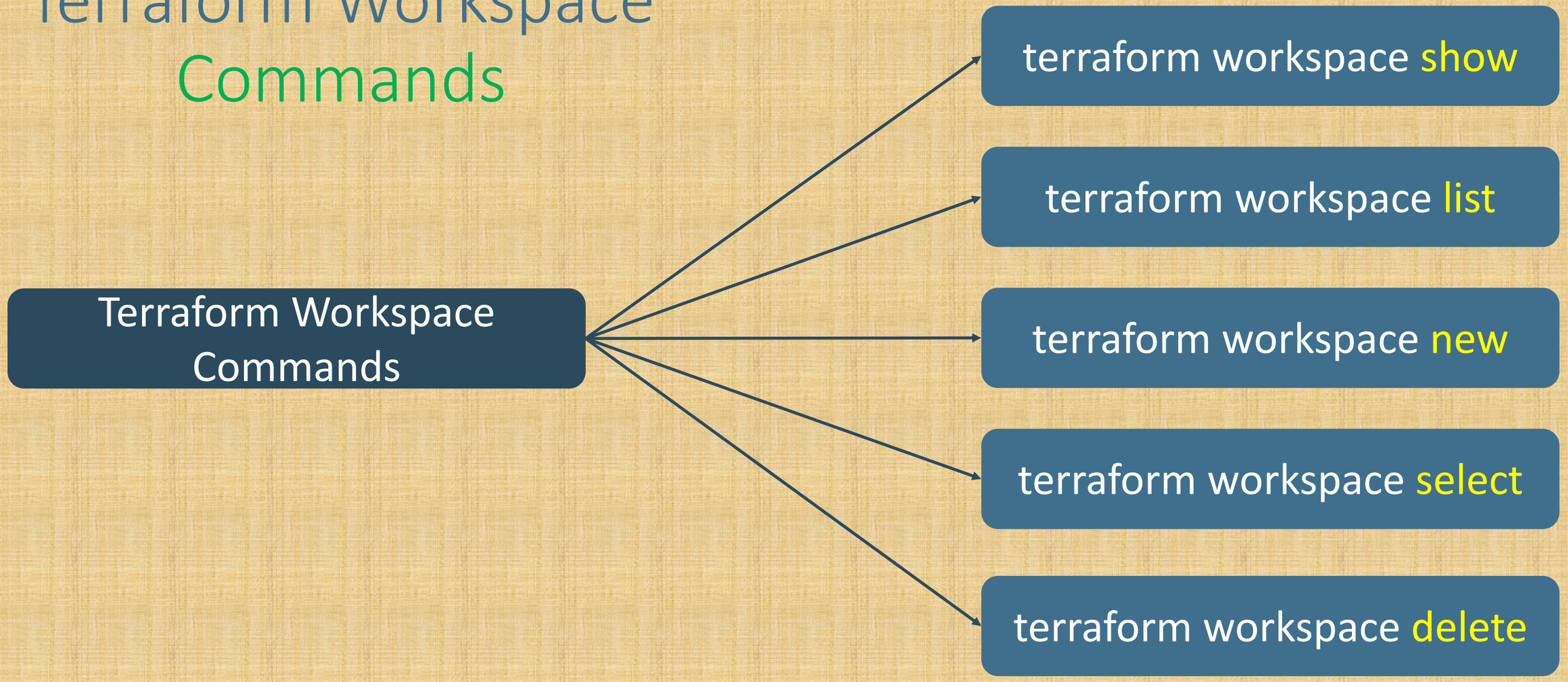
Terraform will not recommend using workspaces for larger infrastructures inline with environments pattern like dev, qa, staging. Recommended to use separate configuration directories

Terraform CLI workspaces are completed different from Terraform Cloud Workspaces

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Workspace Commands



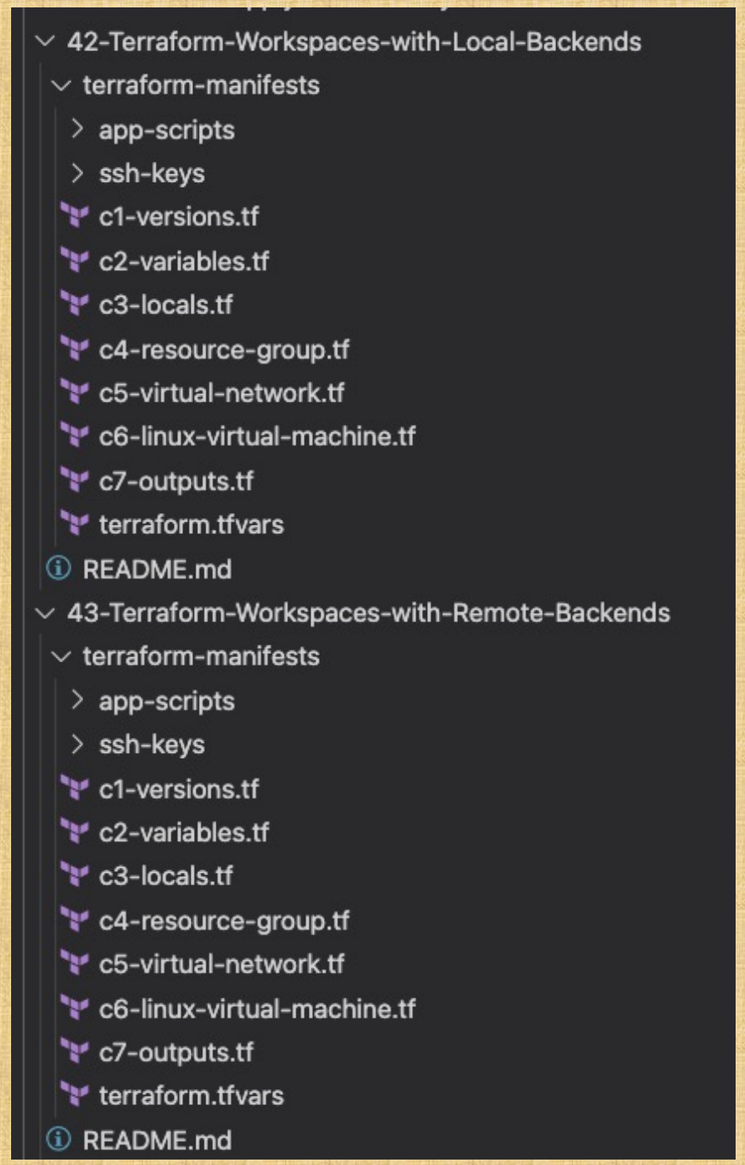
Usecase-1: Local Backend

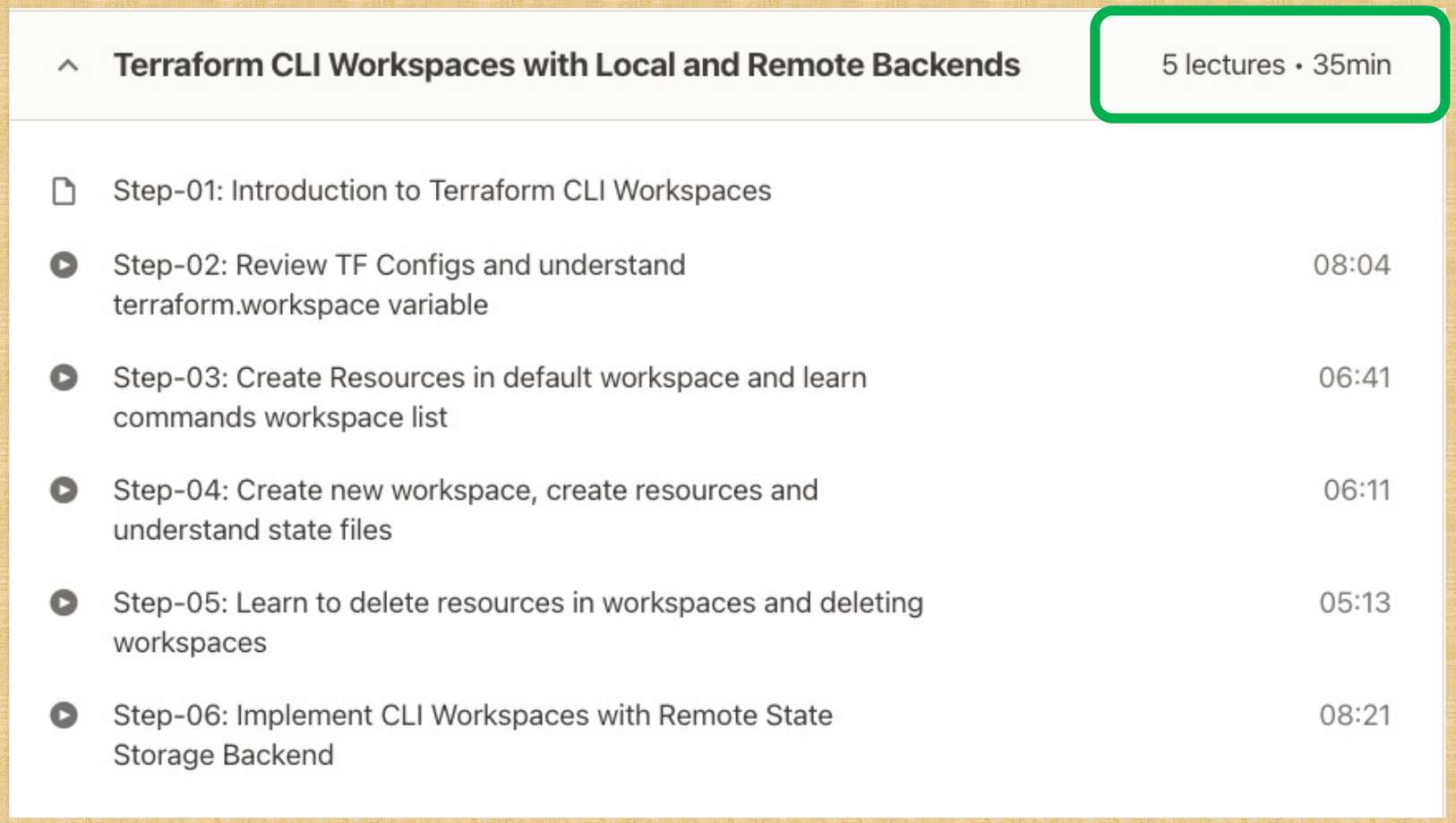
Usecase-2: Remote Backend

Author: Nho Luong

Skill: DevOps Engineer Lead

Practical Example with Step-by-Step Documentation on Github

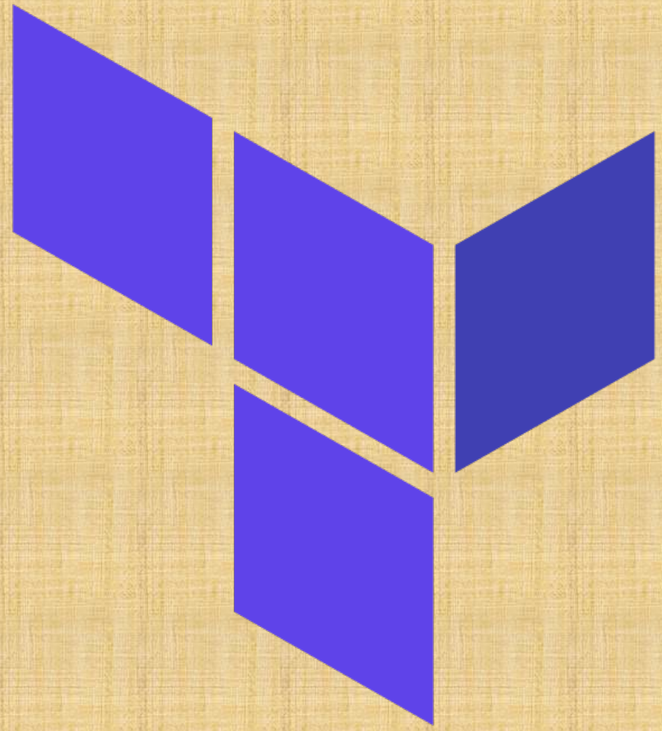




Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Provisioners



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Provisioners

Provisioners can be used to model specific actions on the local machine or on a remote machine in order to prepare servers

Passing data into virtual machines and other compute resources

Running configuration management software (packer, chef, ansible)

Creation-Time Provisioners

Failure Behaviour: Continue: Ignore the error and continue with creation or destruction.

Provisioners are a Last Resort

First-class Terraform provider functionality may be available

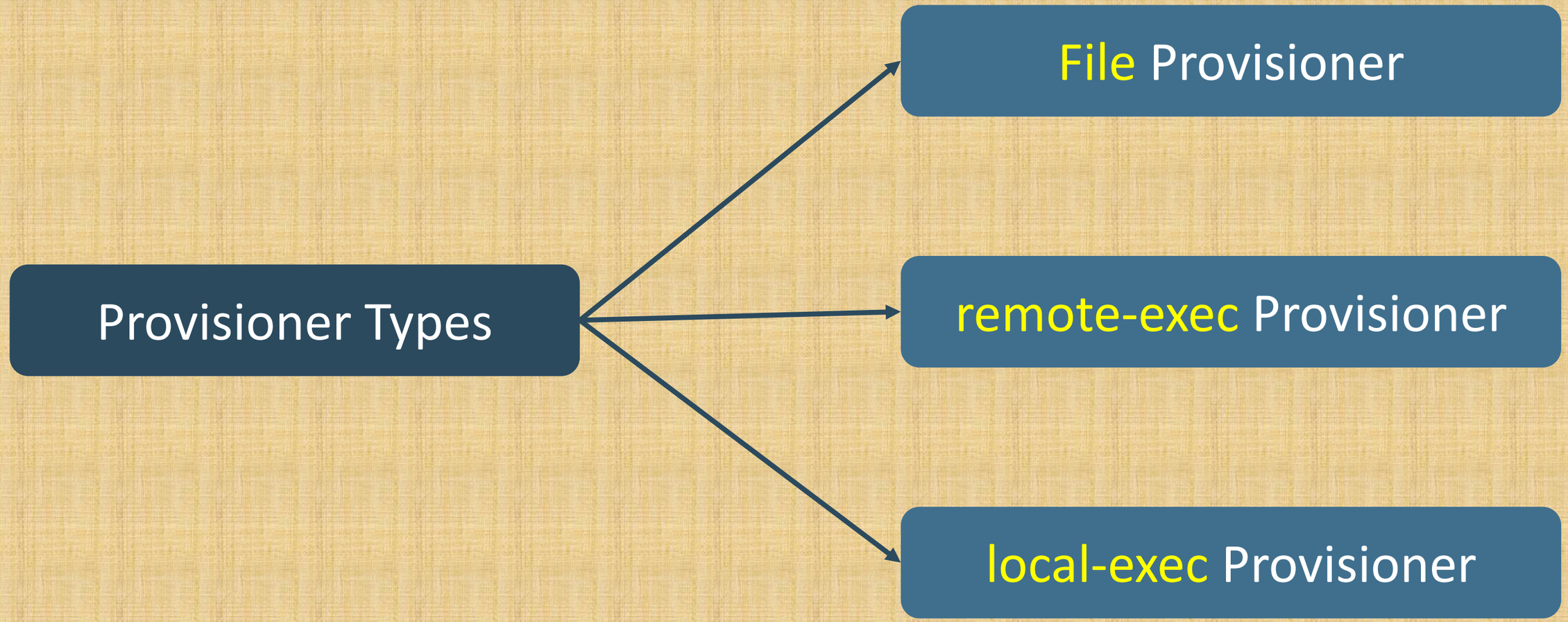
Destroy-Time Provisioners

Failure Behaviour: Fail: Raise an error and stop applying (the default behavior). If creation provisioner, taint resource

Author: Nho Luong

Skill: DevOps Engineer Lead

Types of Provisioners



Author: Nho Luong

Skill: DevOps Engineer Lead

Connection Block

Most provisioners require access to the remote resource via SSH or WinRM, and expect a nested connection block with details about how to connect.

Expressions in connection blocks cannot refer to their parent resource by name. Instead, they can use the special self object.

```
# Connection Block for Provisioners to connect to Azure Virtual Machine
connection {
  type = "ssh"
  host = self.public_ip_address
  user = self.admin_username
  private_key = file("${path.module}/ssh-keys/terraform-azure.pem")
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

File Provisioner

File Provisioner

- File Provisioner is used to copy files or directories from the machine executing Terraform to the newly created resource.
- The file provisioner supports both ssh and winrm type of connections

```
# File Provisioner-1: Copies the file-copy.html file t
provisioner "file" {
  source = "apps/file-copy.html"
  destination = "/tmp/file-copy.html"
}
# File Provisioner-2: Copies the string in content int
provisioner "file" {
  content = "VM Host name: ${self.computer_name}"
  destination = "/tmp/file.log"
}
# File Provisioner-3: Copies the app1 folder to /tmp -
provisioner "file" {
  source = "apps/app1"
  destination = "/tmp"
}
# File Provisioner-4: Copies all files and folders in
provisioner "file" {
  source = "apps/app2/"
  destination = "/tmp"
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

local-exec Provisioner

local-exec Provisioner

- The local-exec provisioner invokes a local executable after a resource is created.
- This invokes a process on the machine running Terraform, not on the resource.

```
# local-exec provisioner-1 (Creation-Time Provisioner - Triggered during Create Resource)
provisioner "local-exec" {
  command = "echo ${azurerm_linux_virtual_machine.mylinuxvm.public_ip_address} >> creation-
  working_dir = "local-exec-output-files/"
}
# local-exec provisioner-2 - (Destroy-Time Provisioner - Triggered during Destroy Resource)
provisioner "local-exec" {
  when    = destroy
  command = "echo Destroy-time provisioner Instanace Destroyed at `date` >> destroy-time.tx
  working_dir = "local-exec-output-files/"
}
```

Creation Time Provisioner (by default)

Destroy Time Provisioner (when = destroy)

Author: Nho Luong

Skill: DevOps Engineer Lead

remote-exec Provisioner

remote-exec Provisioner

- The remote-exec provisioner invokes a script on a remote resource after it is created.
- This can be used to run a configuration management tool, bootstrap into a cluster, etc.

```
# File-Provisioner-1: Copies the file-copy.html file to /tmp/file-copy.html
  provisioner "file" {
    source      = "apps/file-copy.html"
    destination = "/tmp/file-copy.html"
  }

# Remote-exec Provisioner-1: Copies the file to Apache Webserver /var/www/html directory
  provisioner "remote-exec" {
    inline = [
      "sleep 120", # Will sleep for 120 seconds to ensure Apache webserver is provisioned us
      "sudo cp /tmp/file-copy.html /var/www/html"
    ]
  }
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Null-Resource & Provisioners

null_resource

- If you need to run provisioners that aren't directly associated with a specific resource, you can associate them with a null_resource.
- Instances of null_resource are treated like normal resources, but they don't do anything.
- Same as other resource, you can configure provisioners and connection details on a null_resource.

```
# Time Resource
# Wait for 90 seconds after creating the above Azure Virtu
resource "time_sleep" "wait_90_seconds" {
  depends_on = [azurerm_linux_virtual_machine.mylinuxvm]
  create_duration = "90s"
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Null-Resource & Provisioners

```
# Terraform NULL RESOURCE
# Sync App1 Static Content to Webserver using Provisioners
resource "null_resource" "sync_app1_static" {
  depends_on = [time_sleep.wait_90_seconds]
  triggers = {
    always-update = timestamp()
  }

# Connection Block for Provisioners to connect to Azure VM Instance
  connection {
    type = "ssh"
    host = azurerm_linux_virtual_machine.mylinuxvm.public_ip_address
    user = azurerm_linux_virtual_machine.mylinuxvm.admin_username
    private_key = file("${path.module}/ssh-keys/terraform-azure.pem")
  }
# File Provisioner: Copies the app1 folder to /tmp
  provisioner "file" {
    source = "apps/app1"
    destination = "/tmp"
  }
```

Author: Nho Luong

Skill: DevOps Engineer Lead

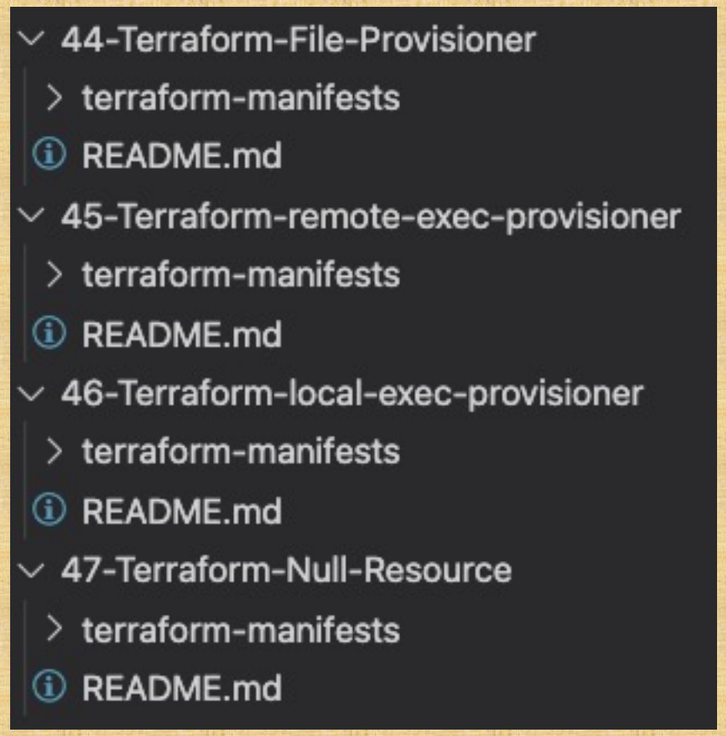
Null-Resource & Provisioners

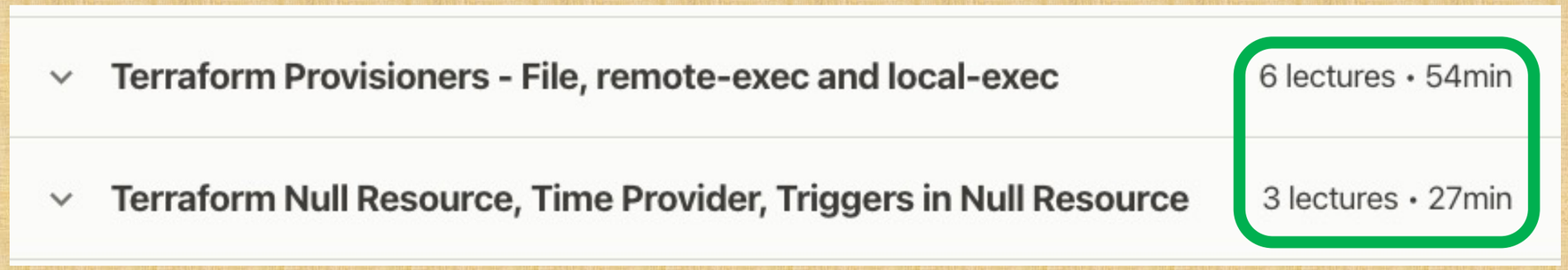
```
# Remote-Exec Provisioner: Copies the /tmp/app1 folder to Apache Webserver /var/www/html dir
  provisioner "remote-exec" {
    inline = [
      "sudo cp -r /tmp/app1 /var/www/html"
    ]
  }
}
```

Author: Nho Luong

Skill: DevOps Engineer Lead

Practical Examples & Step-by-Step Documentation on Github

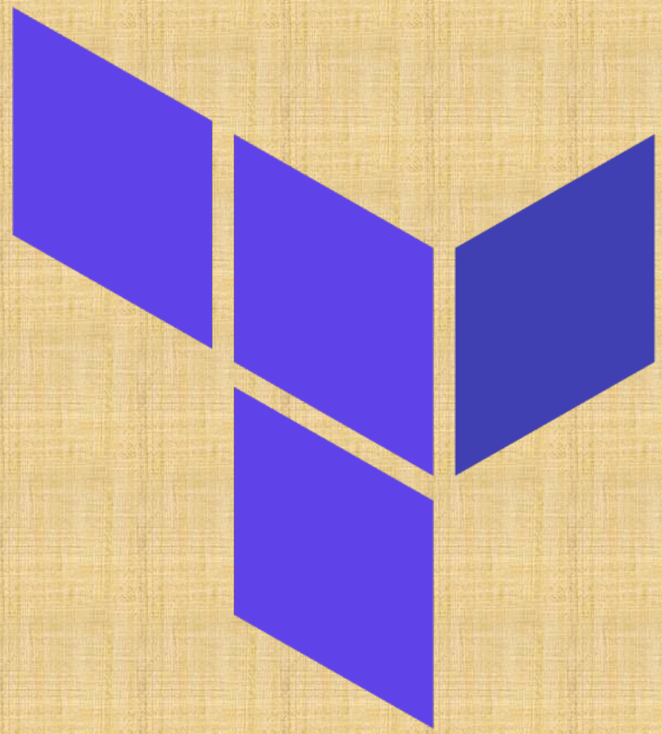




Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Modules



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Modules

Modules are containers for multiple resources that are used together. A module consists of a collection of .tf files kept together in a directory.

Modules are the main way to package and reuse resource configurations with Terraform.

A module that has been called by another module is often referred to as a child module.

Every Terraform configuration has at least one module, known as its root module, which consists of the resources defined in the .tf files in the main working directory

Child modules can be called multiple times within the same configuration, and multiple configurations can use the same child module.

A Terraform module (usually the root module of a configuration) can call other modules to include their resources into the configuration.

In addition to modules from the local filesystem, Terraform can load modules from a public or private registry.

This makes it possible to publish modules for others to use, and to use modules that others have published.

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Registry – Use Publicly Available Modules

The Terraform Registry hosts a broad collection of publicly available Terraform modules for configuring many kinds of common infrastructure.

Demo

These modules are free to use, and Terraform can download them automatically if you specify the appropriate source and version in a module call block.

```
# Create Virtual Network and Subnets using Terraform Public Registry Mo
module "vnet" {
  source  = "Azure/vnet/azurerm"
  version = "2.5.0" # No versions constraints for production grade imp
  vnet_name = local.vnet_name
  resource_group_name = azurerm_resource_group.myrg.name
  address_space       = ["10.0.0.0/16"]
  subnet_prefixes     = ["10.0.1.0/24", "10.0.2.0/24", "10.0.3.0/24"]
  subnet_names        = ["subnet1", "subnet2", "subnet3"]
  subnet_service_endpoints = {
    subnet2 = ["Microsoft.Storage", "Microsoft.Sql"],
    subnet3 = ["Microsoft.AzureActiveDirectory"]
  }
  tags = {
    environment = "dev"
    costcenter  = "it"
  }
  depends_on = [azurerm_resource_group.myrg]
}
```

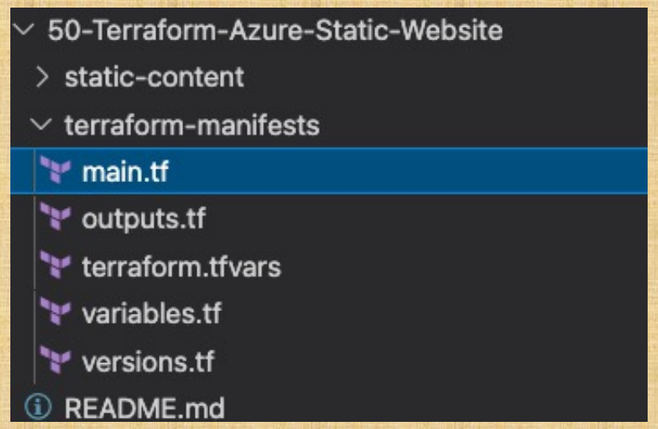
Author: Nho Luong

Skill: DevOps Engineer Lead

Build manually and then automate using Terraform

1. Build a Static Website using Azure manually using Azure Portal.
2. Automate it using Terraform Resources

Demo




```
# Create Azure Storage account
resource "azurerm_storage_account" "storage_account" {
  name                     = "${var.storage_account_name}${random_string.myrandom.id}"
  resource_group_name = azurerm_resource_group.resource_group.name

  location                 = var.location
  account_tier             = var.storage_account_tier
  account_replication_type = var.storage_account_replication_type
  account_kind             = var.storage_account_kind

  static_website {
    index_document     = var.static_website_index_document
    error_404_document = var.static_website_error_404_document
  }
}
```

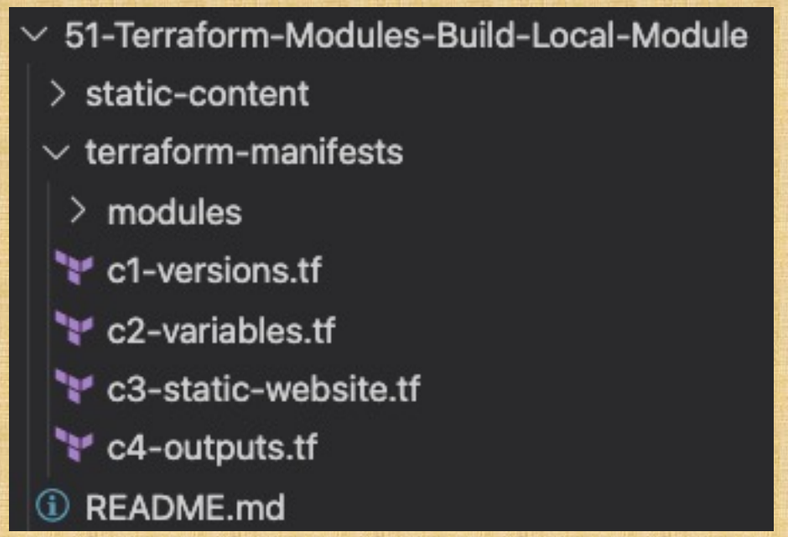
Author: Nho Luong

Skill: DevOps Engineer Lead

Build a Local Terraform Module

Demo

Build a Local Terraform Module and call it from Root Module and Test




```hcl
# Call our Custom Terraform Module which we built earlier
module "azure_static_website" {
  source = "./modules/azure-static-website" # Mandatory

  # Resource Group
  location                          = "eastus"
  resource_group_name               = "myrg1"

  # Storage Account
  storage_account_name              = "staticwebsite"
  storage_account_tier              = "Standard"
  storage_account_replication_type  = "LRS"
  storage_account_kind              = "StorageV2"
  static_website_index_document     = "index.html"
  static_website_error_404_document = "error.html"
}
```

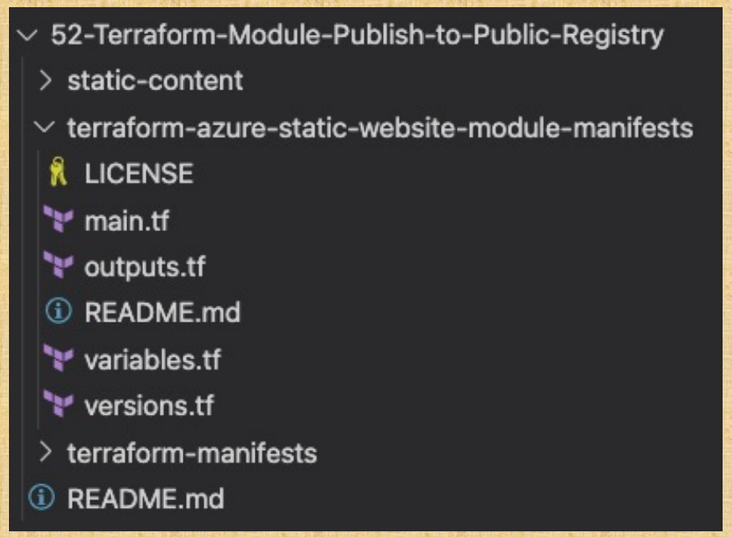
Author: Nho Luong

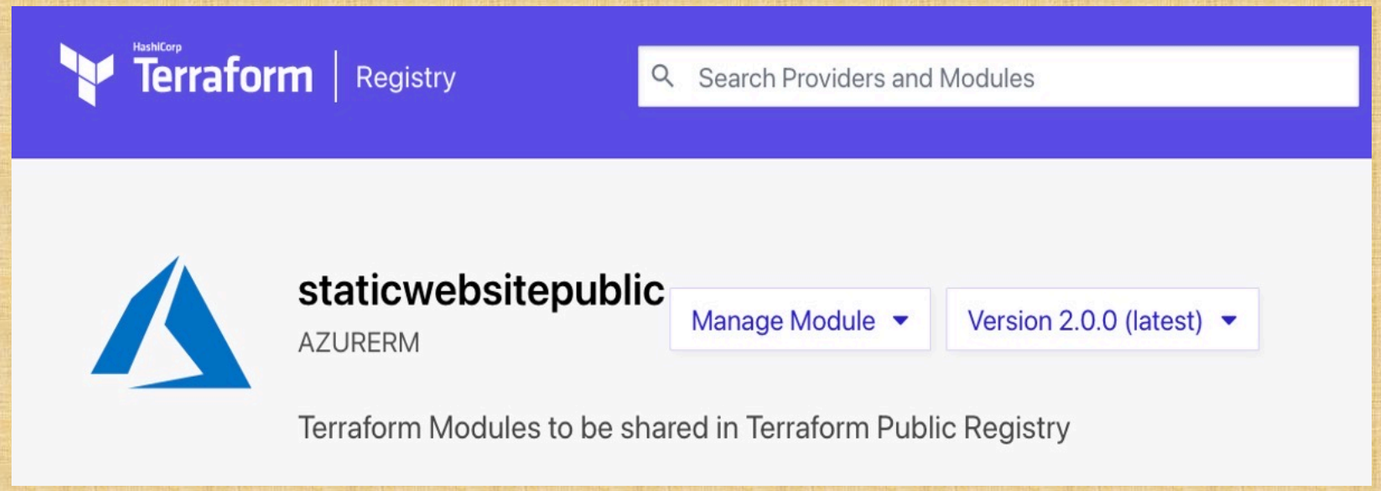
Skill: DevOps Engineer Lead

Publish to Public Terraform Registry

Demo

Publish to Public Terraform Registry and Access it from Public Registry and Test.





Author: Nho Luong

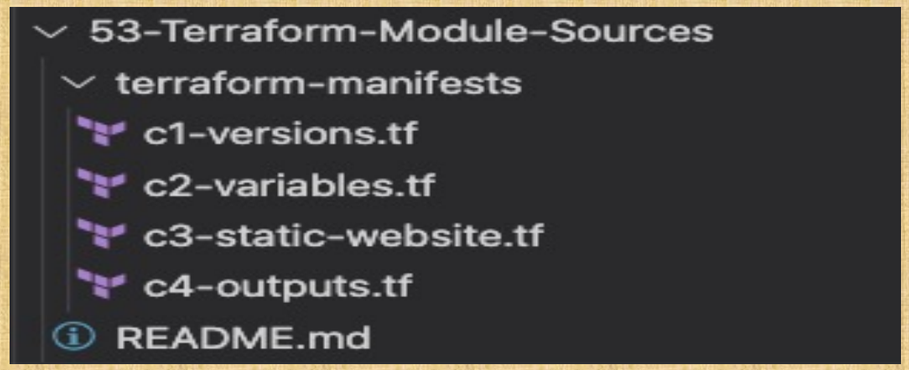
Skill: DevOps Engineer Lead

Use various Module Sources

Demo

In addition to Terraform Public and Private Registry use various module sources.

```
# Call our Custom Terraform Module which we built earlier
module "azure_static_website" {

  # Terraform Local Module (Local Paths)
  #source = "./modules/azure-static-website"
```



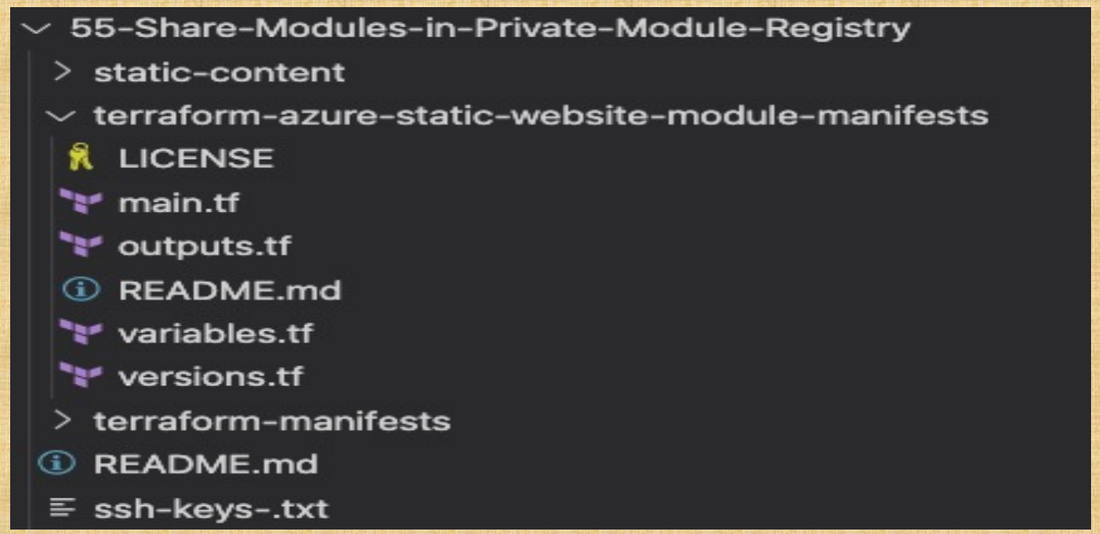
Author: Nho Luong

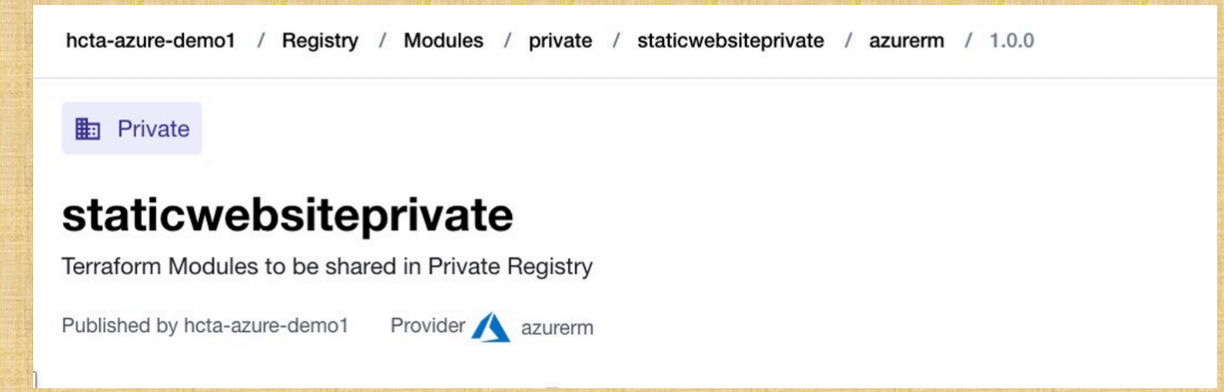
Skill: DevOps Engineer Lead

Private Module Registry in Terraform Cloud & Enterprise

Members of your organization might produce modules specifically crafted for your own infrastructure needs.

Terraform Cloud and Terraform Enterprise both include a private module registry for sharing modules internally within your organization.



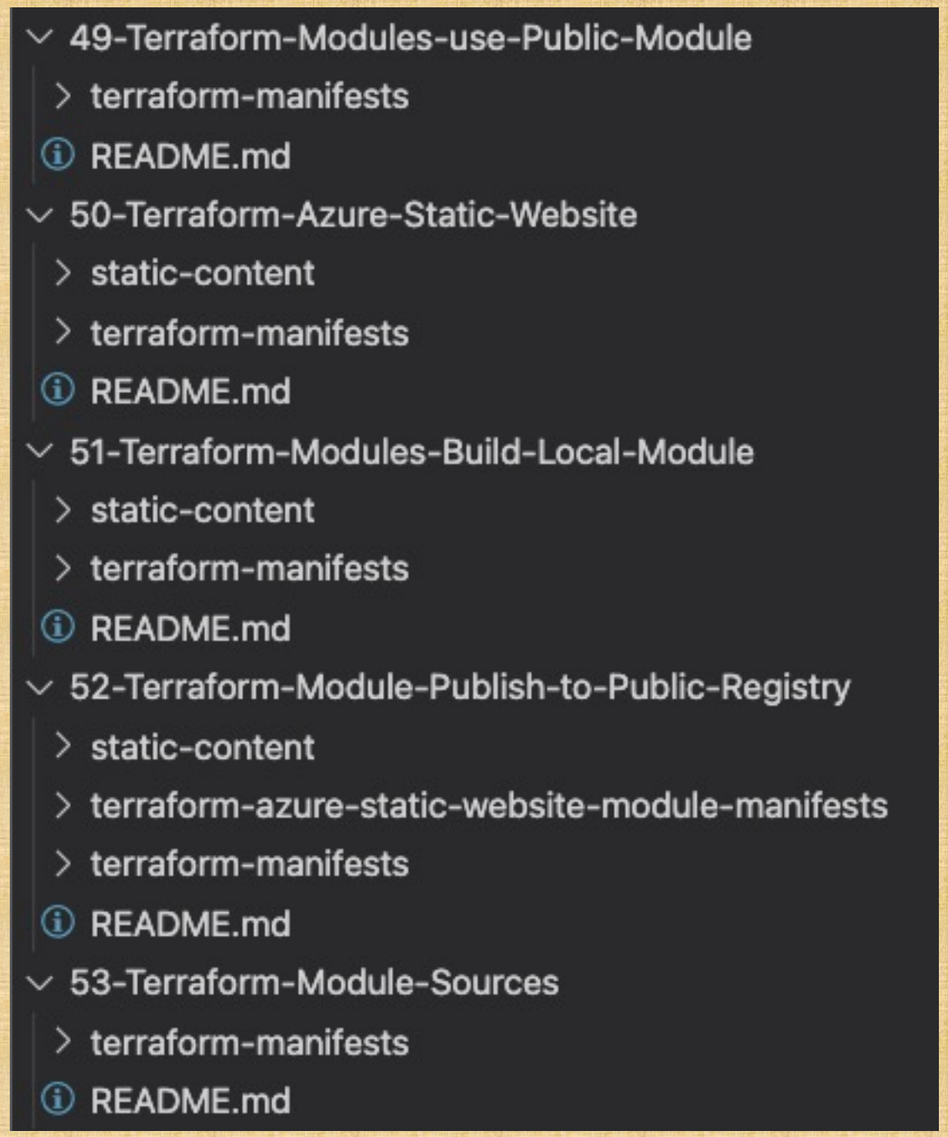


Demo

Author: Nho Luong

Skill: DevOps Engineer Lead

Practical Examples & Step-by-Step Documentation on Github

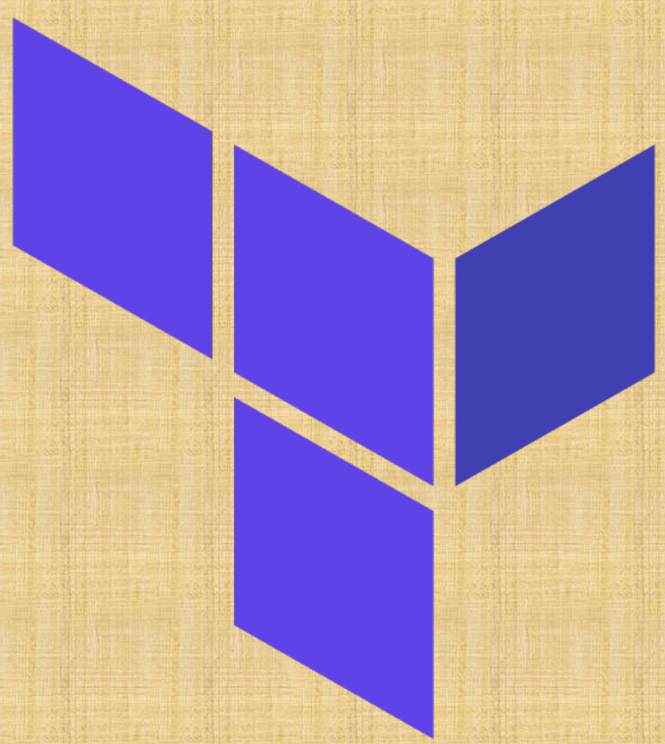


Terraform Modules - Use Public Registry Module	4 lectures • 42min
Terraform Azure Static Website	3 lectures • 21min
Terraform Modules - Build Local Terraform Module	4 lectures • 26min
Terraform Modules - Publish to Terraform Public Registry	4 lectures • 25min
Terraform Module Sources	1 lecture • 12min
Terraform Cloud - Private Module Registry	4 lectures • 34min

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Cloud



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Cloud
Or
Terraform Enterprise

Terraform Cloud and Terraform Enterprise are different distributions of the same application

It manages Terraform runs in a consistent and reliable environment, and includes easy access to shared state and secret data, access controls for approving changes to infrastructure, a private registry for sharing Terraform modules, detailed policy controls for governing the contents of Terraform configurations

Terraform Cloud is available as a hosted service at https://app.terraform.io.

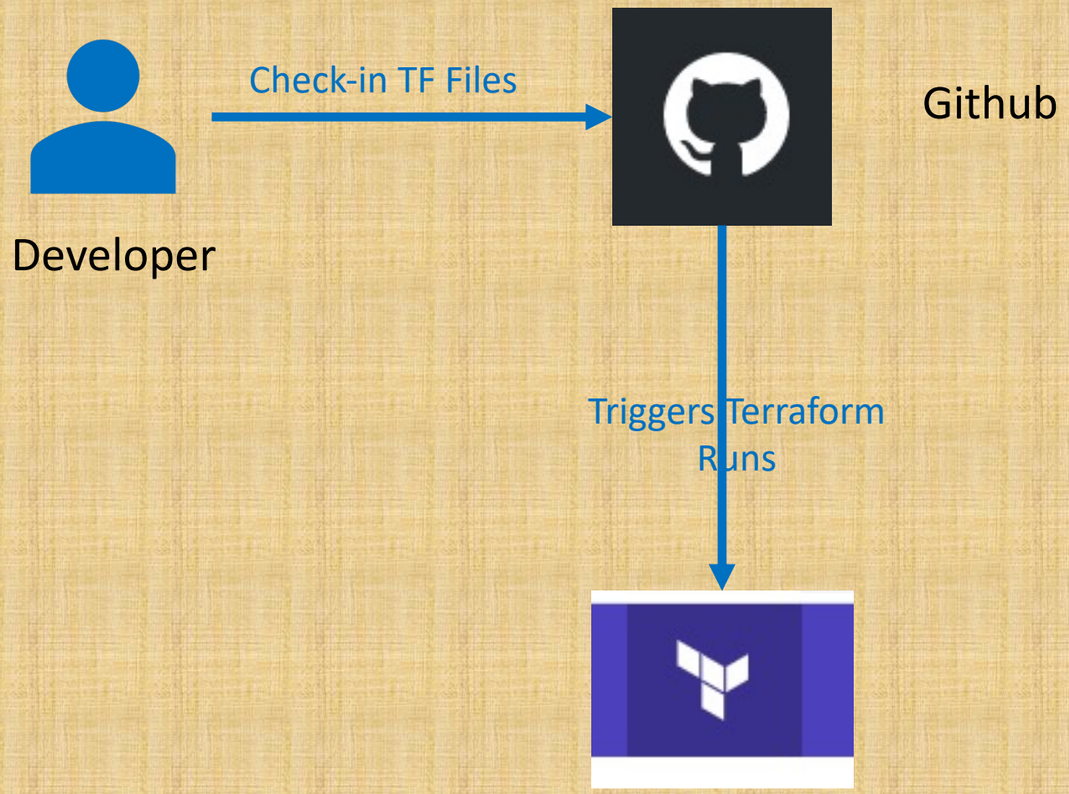
They offer free accounts for small teams, and paid plans with additional feature sets for medium-sized businesses.

Large enterprises can purchase Terraform Enterprise, their self-hosted distribution of Terraform Cloud.

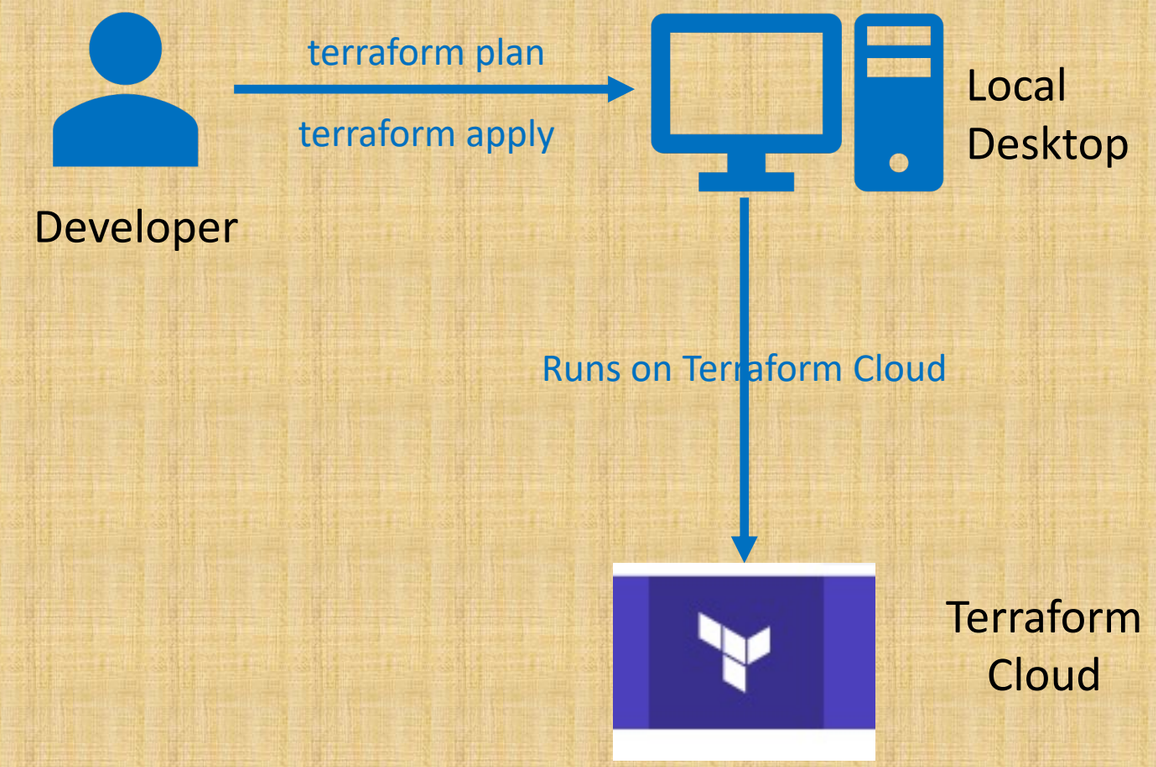
Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform VCS-Driven Workflow



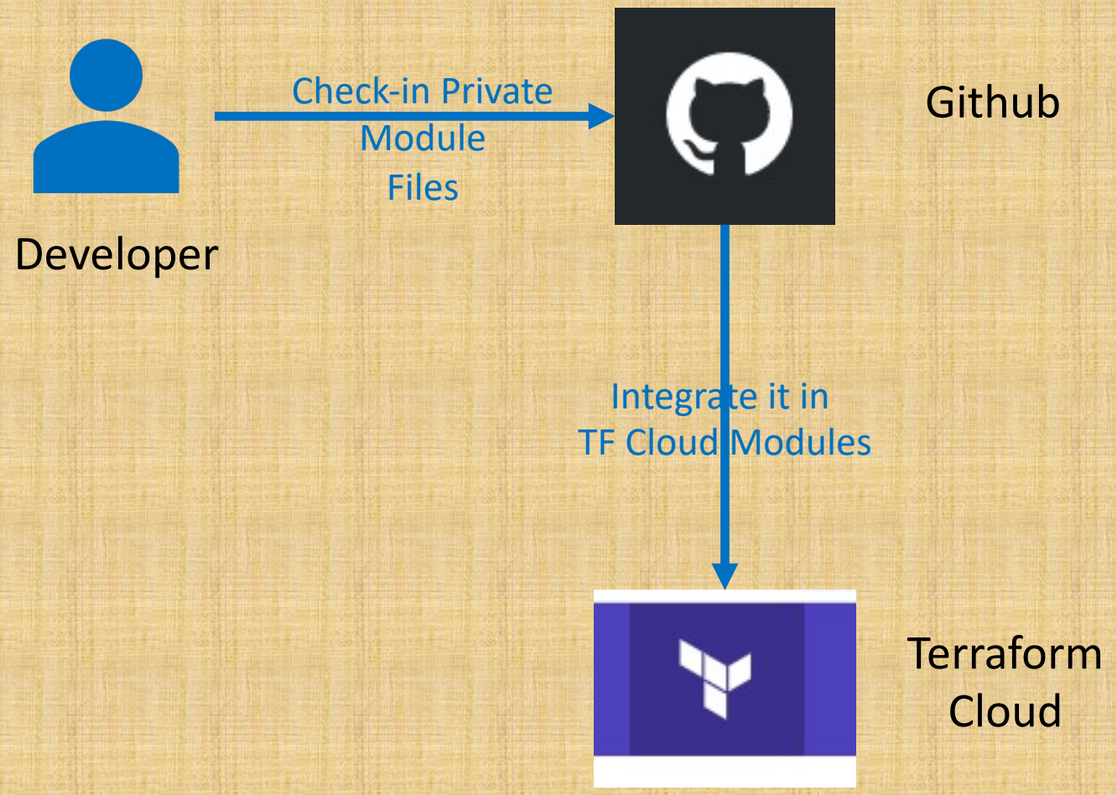
Terraform CLI-Driven Workflow



Author: Nho Luong

Skill: DevOps Engineer Lead

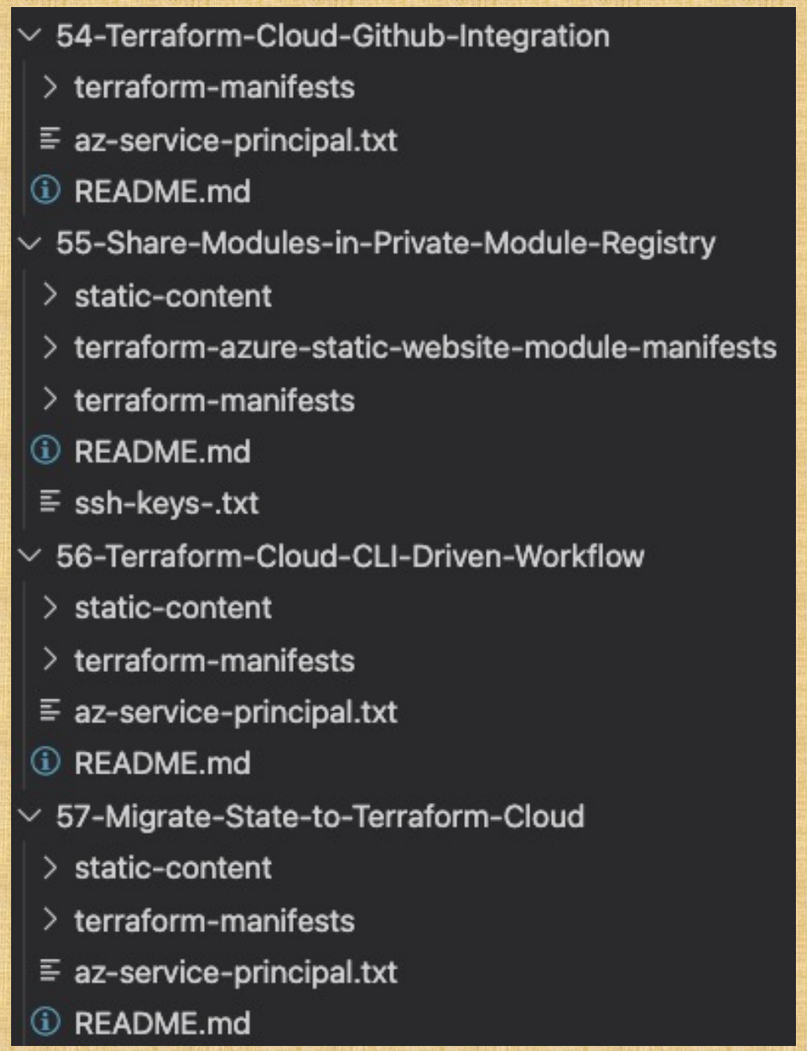
Publish Private Module Registry in Terraform Cloud



Author: Nho Luong

Skill: DevOps Engineer Lead

Practical Examples with Step-by-Step Github Documentation



Terraform Cloud - Version Control Workflow	8 lectures • 1hr 18min
Terraform Cloud - Private Module Registry	4 lectures • 34min
Terraform Cloud - CLI Driven Workflow	3 lectures • 27min
Terraform Cloud - Migrate State to Terraform Cloud	3 lectures • 22min
Terraform Cloud - Sentinel Policies	5 lectures • 51min
Terraform Cloud - Sentinel Foundational Policies	2 lectures • 11min

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Pricing

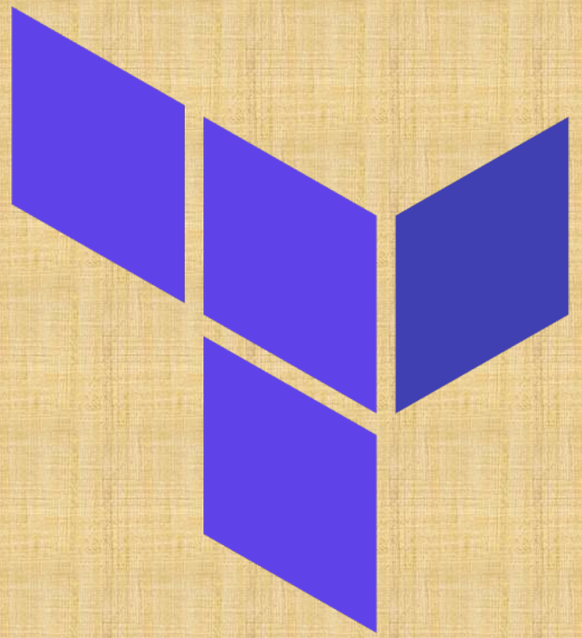
Terraform Cloud	Free	Team & Governance	Business
Get started with our Cloud Infrastructure Automation as a Service	Collaborate on infrastructure as code for Terraform configurations.	Collaborate on infrastructure as code, manage users, and enforce provisioning policies.	Everything organizations need to use Terraform at scale.
Cloud Pricing	$0 up to 5 users	Starting at $20 user/month	Contact Sales

https://www.hashicorp.com/products/terraform/pricing

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Cloud Version Control Workflow



Author: Nho Luong

Skill: DevOps Engineer Lead

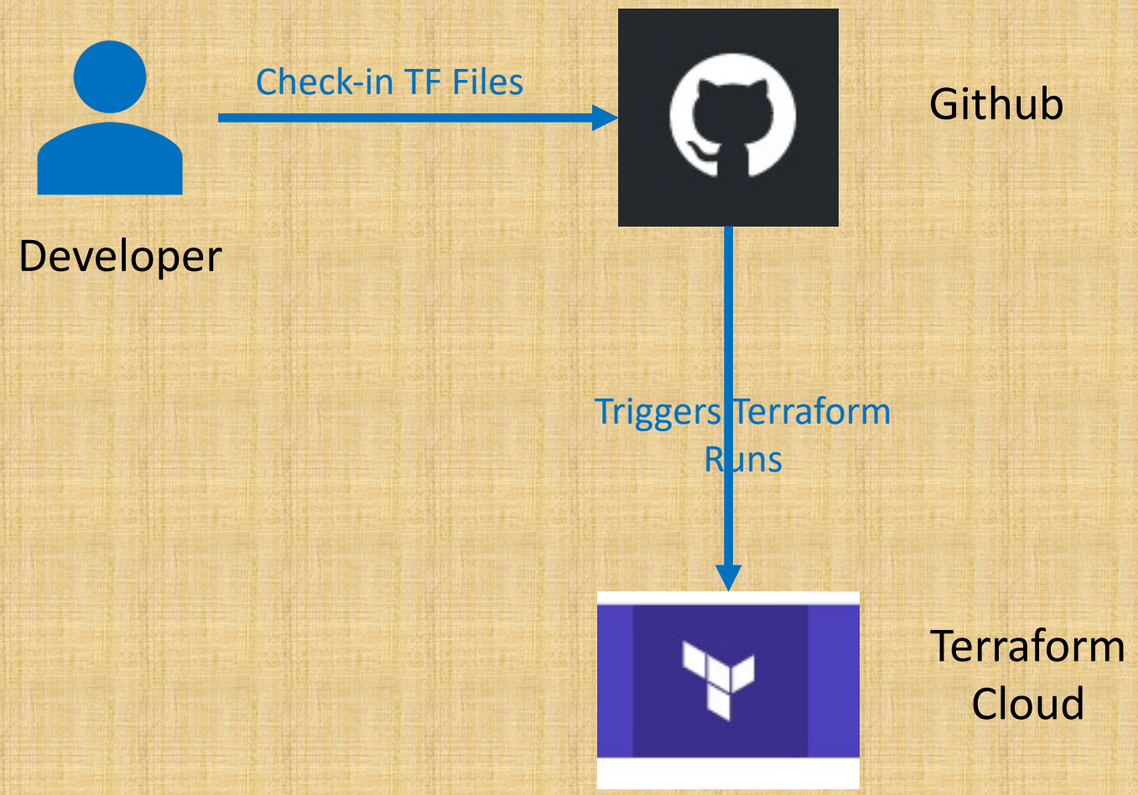
Terraform VCS Integration

Terraform Cloud is more powerful when you integrate it with your version control system (VCS) provider (Github, Bitbucket, Gitlab).

Terraform Cloud can automatically initiate Terraform runs

Terraform Cloud makes code review easier by automatically predicting how pull requests will affect infrastructure.

Publishing new versions of a private Terraform module is as easy as pushing a tag to the module's repository.



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Cloud – Supported VCS Providers

Supported VCS Providers

Terraform Cloud supports the following VCS providers:

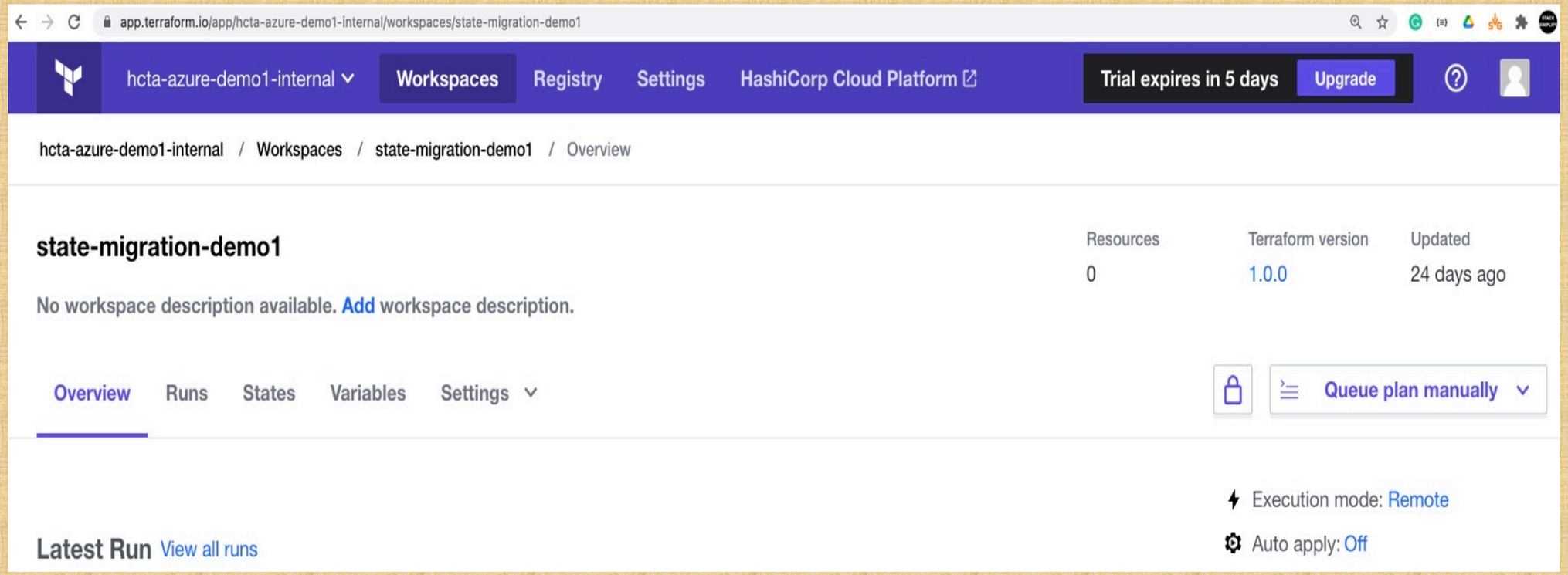
- GitHub.com
- GitHub.com (OAuth)
- GitHub Enterprise
- GitLab.com
- GitLab EE and CE
- Bitbucket Cloud
- Bitbucket Server
- Azure DevOps Server
- Azure DevOps Services

https://www.terraform.io/docs/cloud/vcs/index.html

Author: Nho Luong

Skill: DevOps Engineer Lead

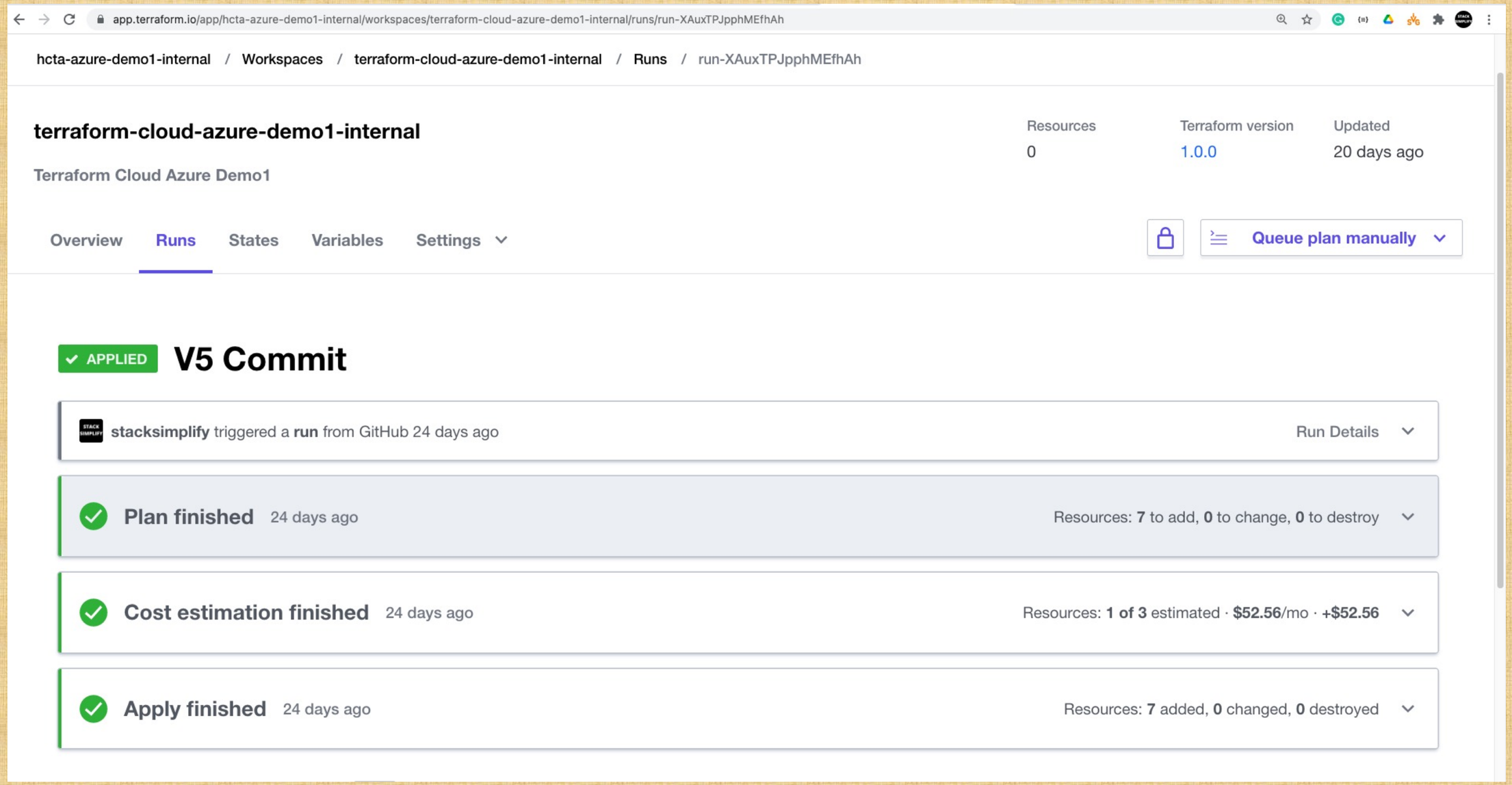
Terraform Cloud – Overview Tab



Author: Nho Luong

Skill: DevOps Engineer Lead

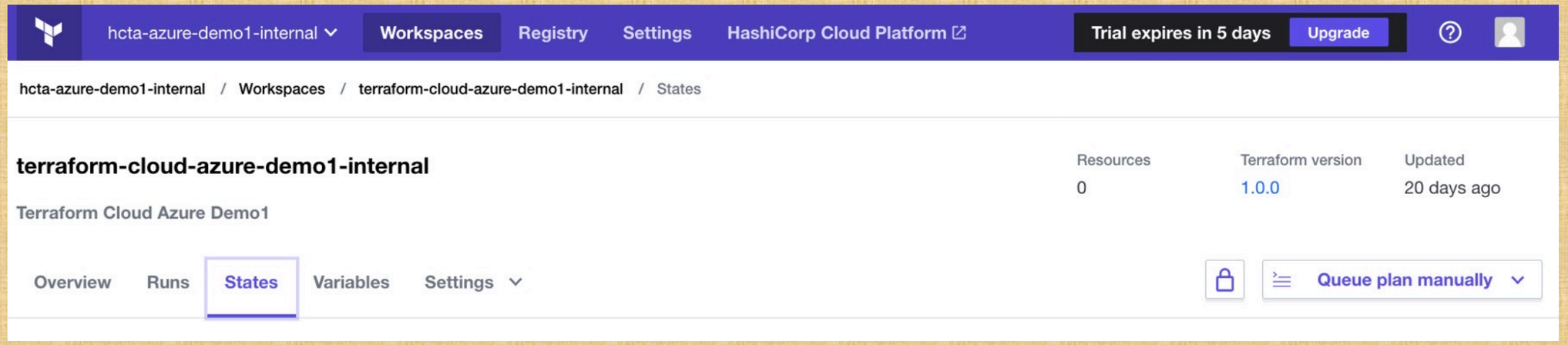
Terraform Cloud – Runs Tab

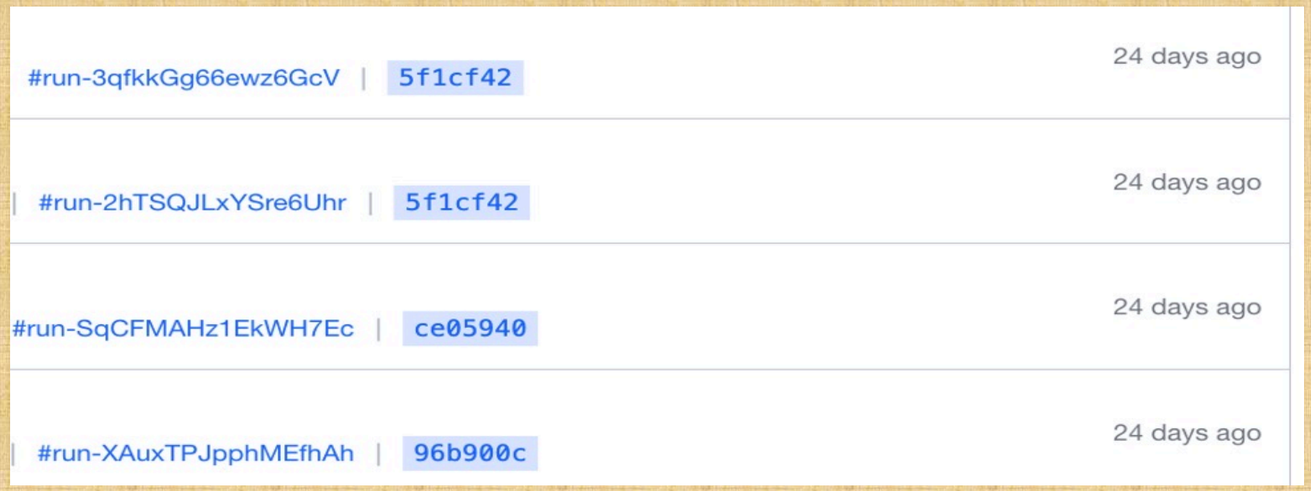


Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Cloud – States Tab



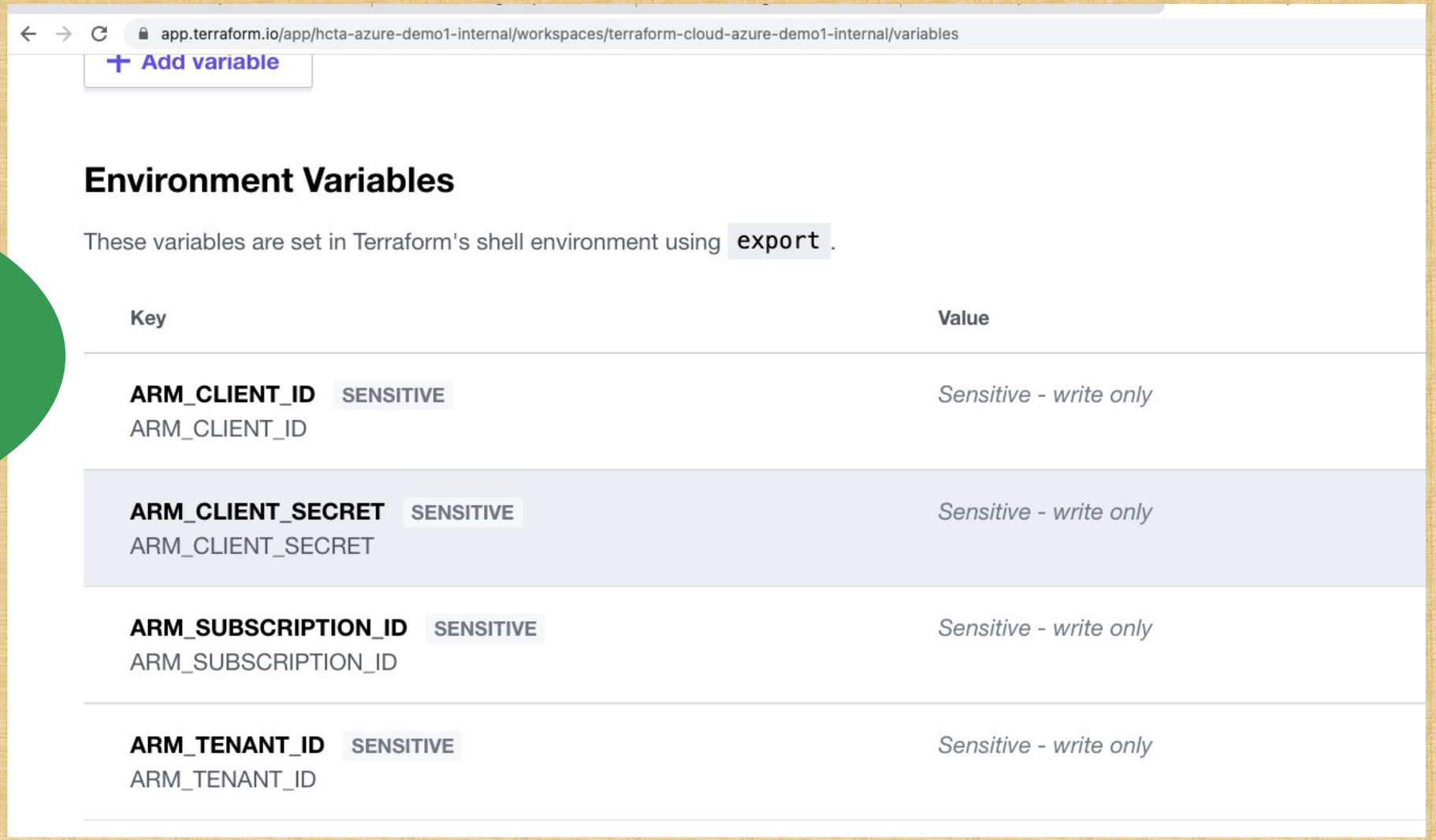


Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Cloud – Variables Tab

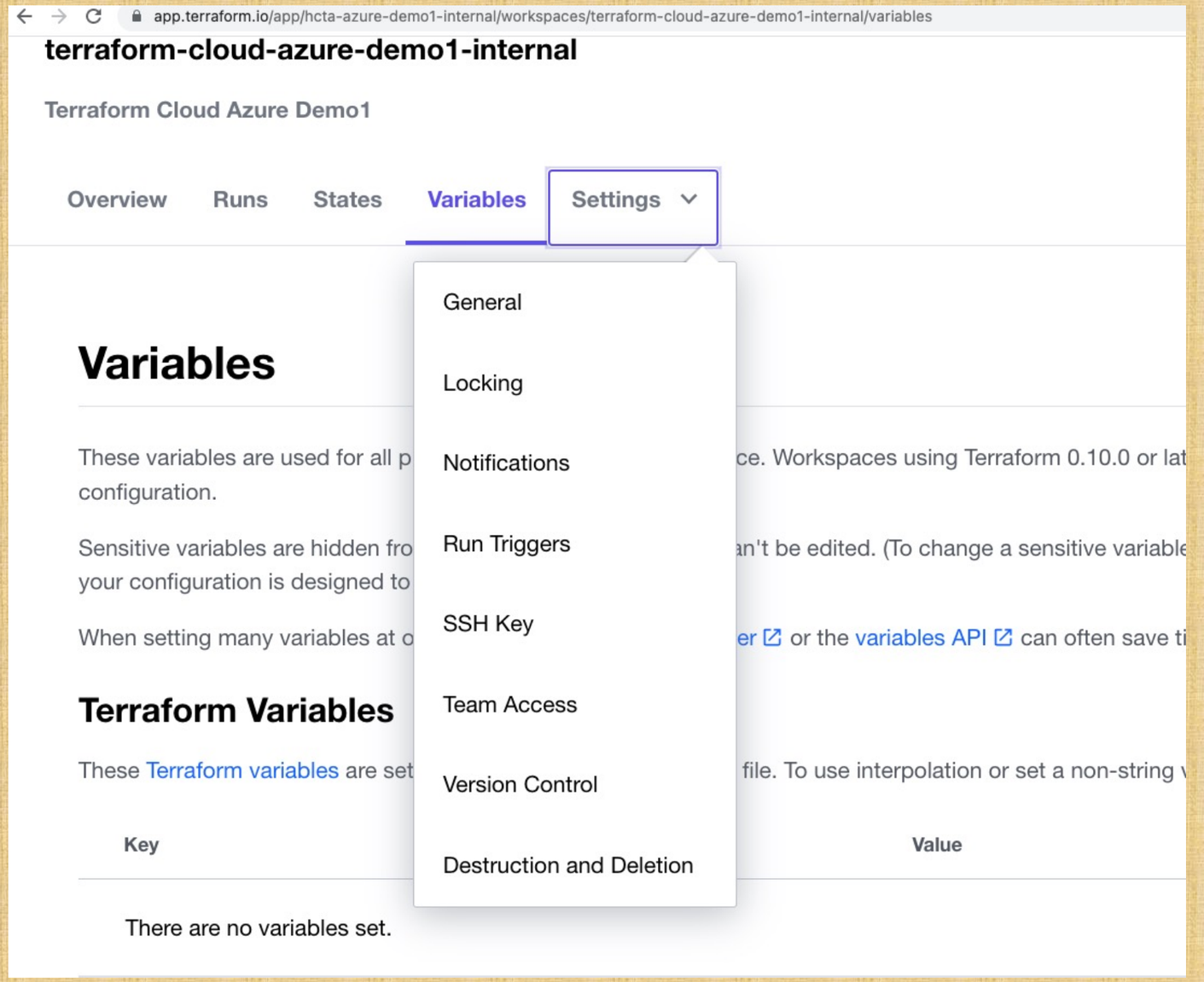
Environment Variables
defined with
Azure Client ID and Secret
to Connect to Azure Cloud



Author: Nho Luong

Skill: DevOps Engineer Lead

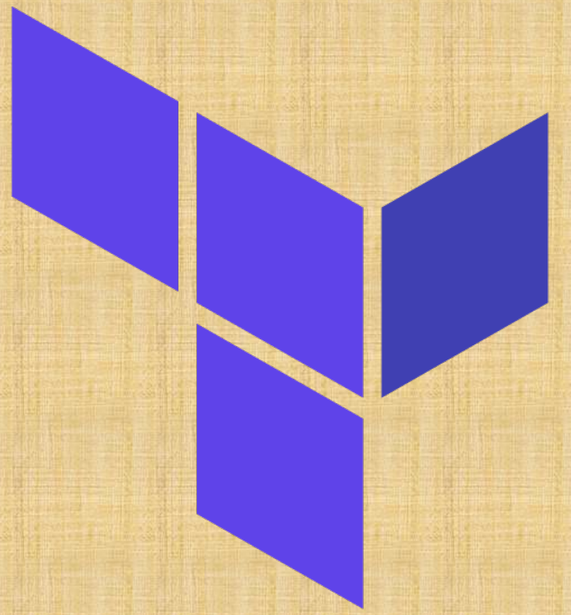
Terraform Cloud Settings Tab



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Cloud Private Module Registry



Author: Nho Luong

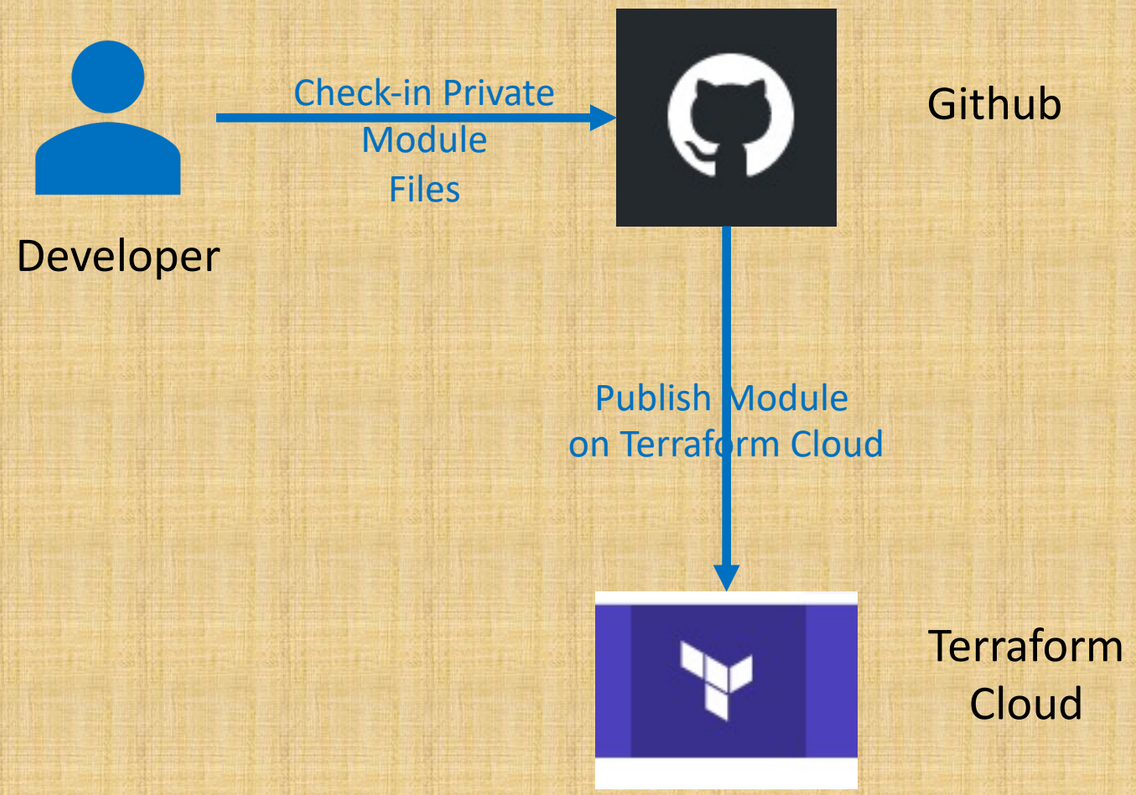
Skill: DevOps Engineer Lead

Terraform Private Module Registry

Terraform Cloud's private module registry helps you share Terraform modules across your organization.

It includes support for module versioning, a searchable and filterable list of available modules, and a configuration designer to help you build new workspaces faster.

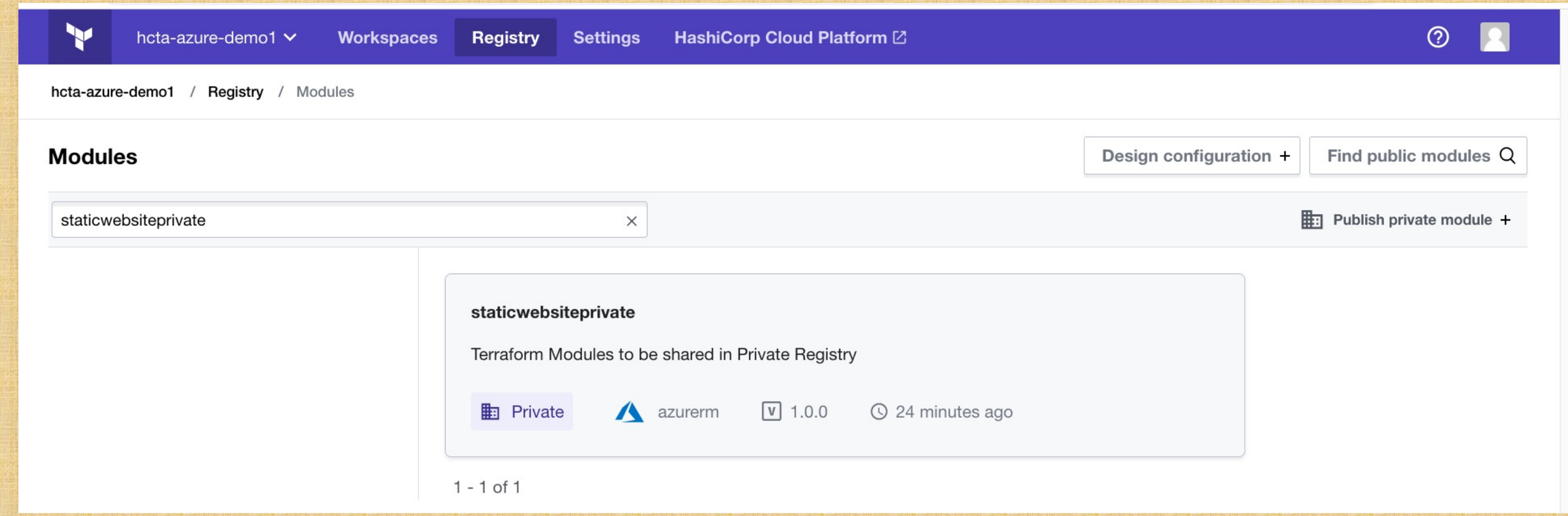
By design, the private module registry works much like the public Terraform Registry. If you're already used the public registry, Terraform Cloud's registry will feel familiar.



Author: Nho Luong

Skill: DevOps Engineer Lead

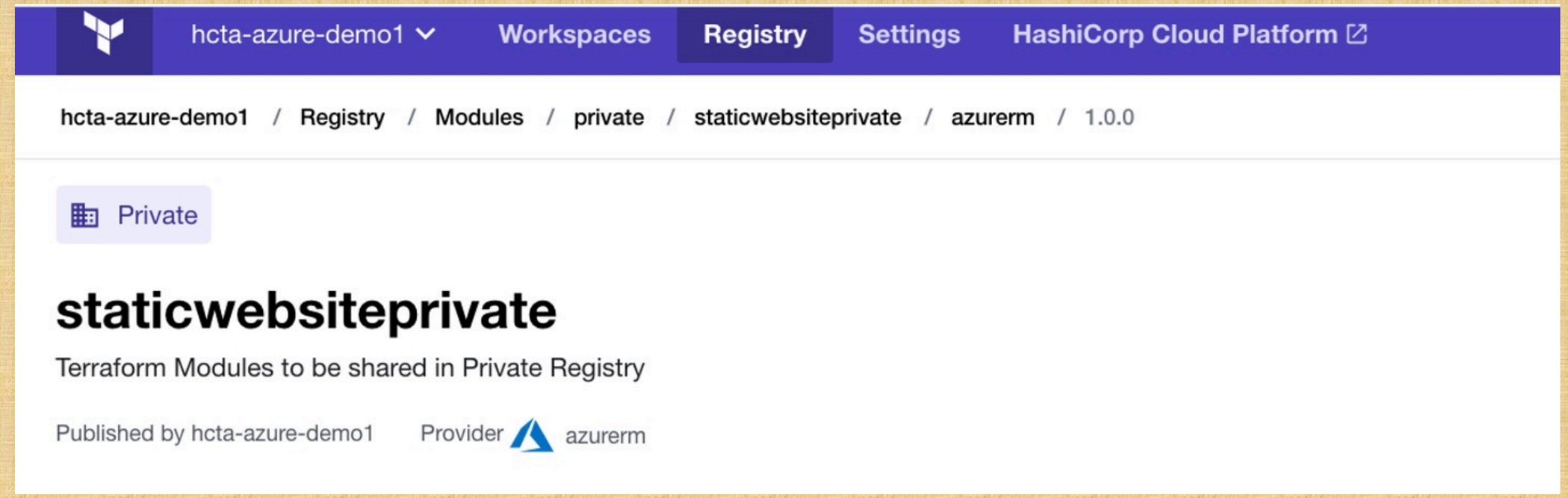
Terraform Cloud – Private Module Registry



Author: Nho Luong

Skill: DevOps Engineer Lead

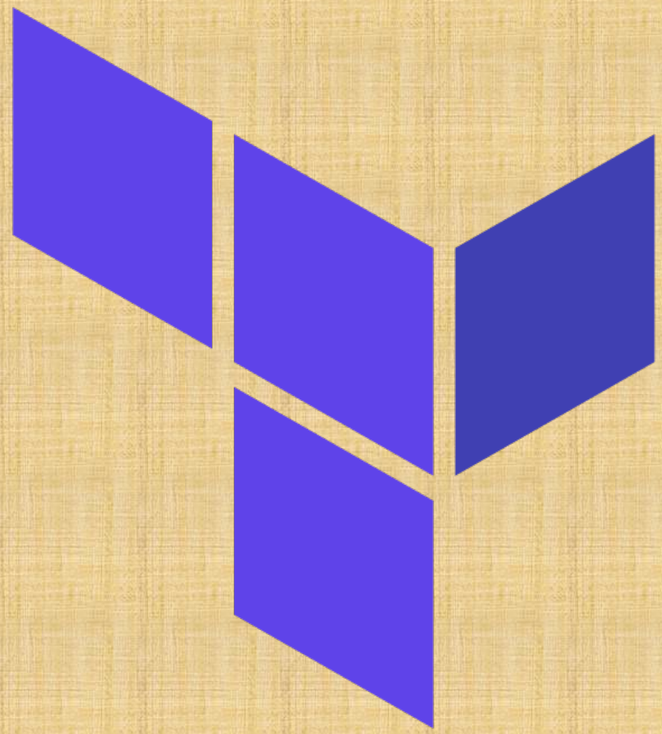
Terraform Cloud – Private Module Registry



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform
Cloud
CLI Driven Workflow



Author: Nho Luong

Skill: DevOps Engineer Lead

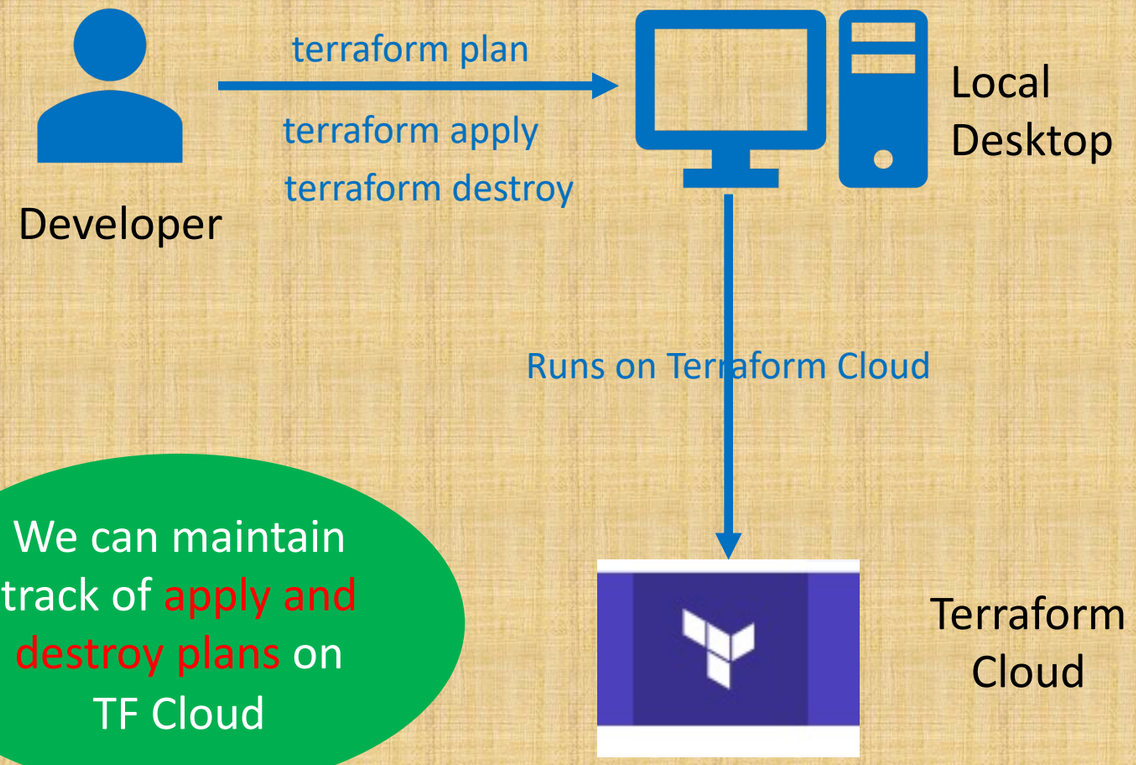
Terraform Cloud - CLI Driven Workflow

The CLI-driven run workflow uses Terraform's standard CLI tools to execute runs in Terraform Cloud.

The Terraform remote backend brings Terraform Cloud's collaboration features into the familiar Terraform CLI workflow

Users can start runs with the standard terraform plan and terraform apply commands, and can watch the progress of the run without leaving their terminal.

These runs execute remotely in Terraform Cloud; they use variables from the appropriate workspace, enforce any applicable Sentinel policies, and can access Terraform Cloud's private module registry and remote state inputs.



We can maintain track of apply and destroy plans on TF Cloud

Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Cloud - CLI Driven Workflow

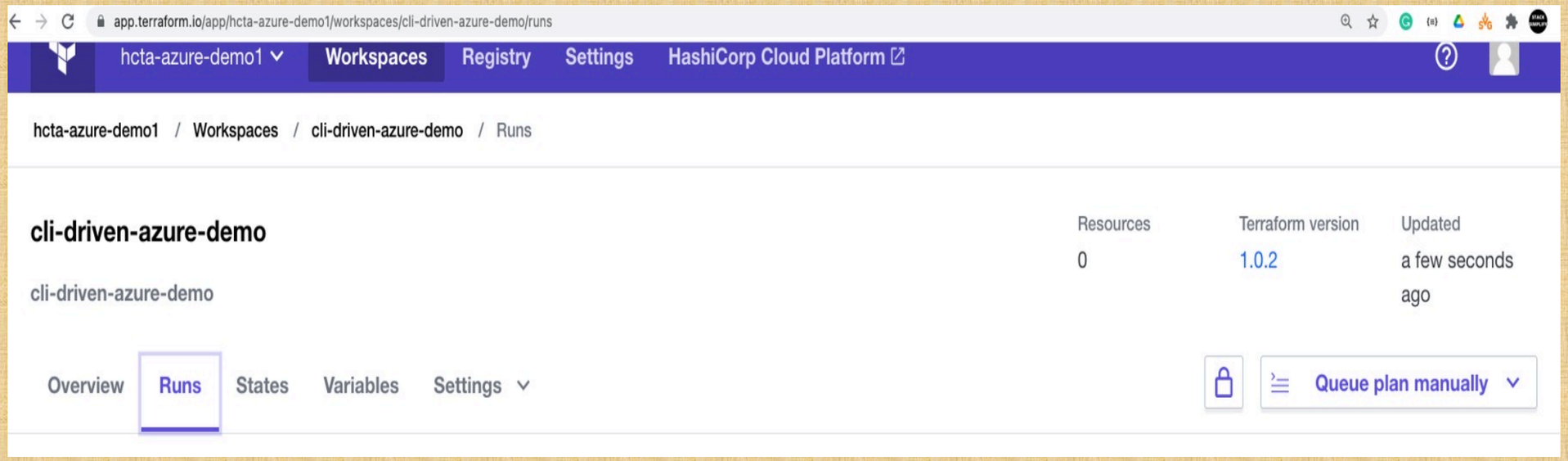
```
# Terraform Block
terraform {
  required_version = ">= 1.0.0"
  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = ">= 2.0"
    }
  }
  # Update Terraform Cloud Backend Block
  backend "remote" {
    organization = "hcta-azure-demo1"
    workspaces {
      name = "cli-driven-azure-demo"
    }
    #hostname = "value"  # defaults to ap
    #token = "value" # Hard Code TF Cloud
  }
}
```

Terraform Cloud as Backend configured in Terraform Block

Author: Nho Luong

Skill: DevOps Engineer Lead

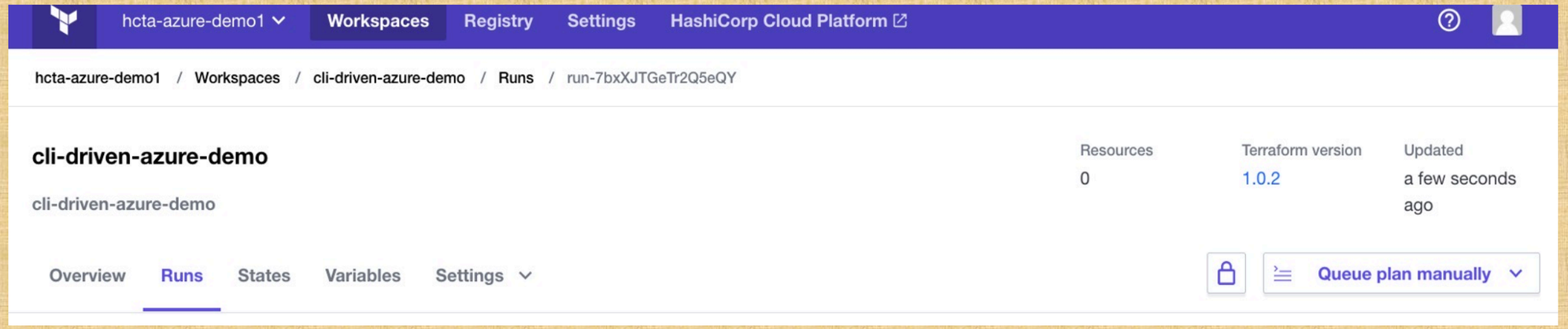
Terraform Cloud - CLI Driven Workflow



Author: Nho Luong

Skill: DevOps Engineer Lead

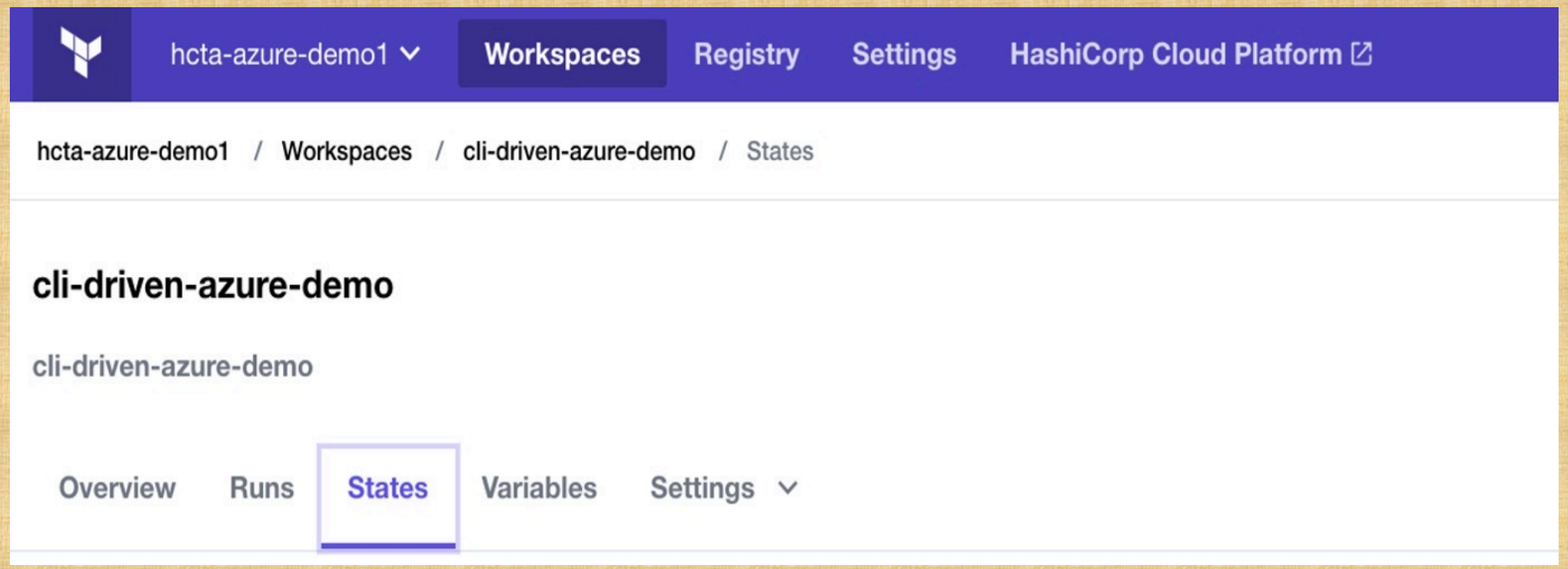
Terraform Cloud - CLI Driven Workflow



Author: Nho Luong

Skill: DevOps Engineer Lead

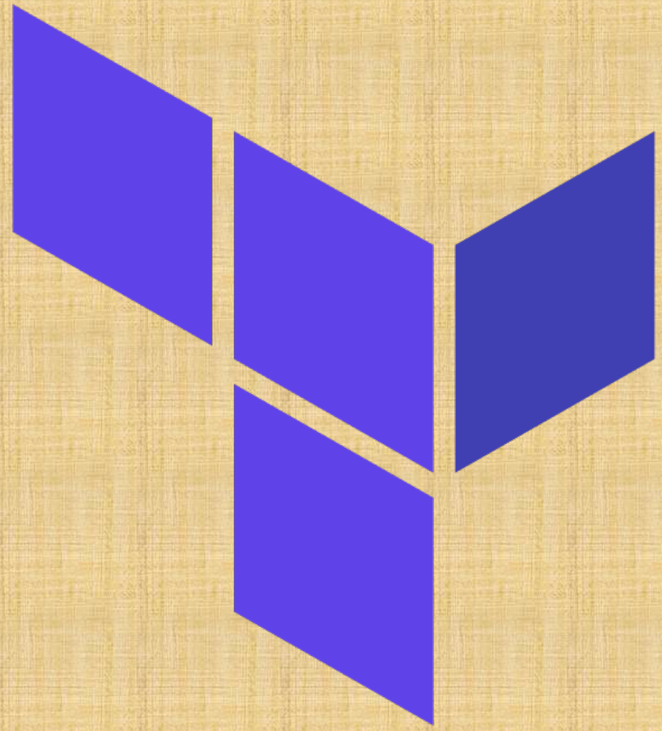
Terraform Cloud - CLI Driven Workflow – States Tab



Author: Nho Luong

Skill: DevOps Engineer Lead

Terraform Cloud Sentinel



Author: Nho Luong

Skill: DevOps Engineer Lead

What is Sentinel ?

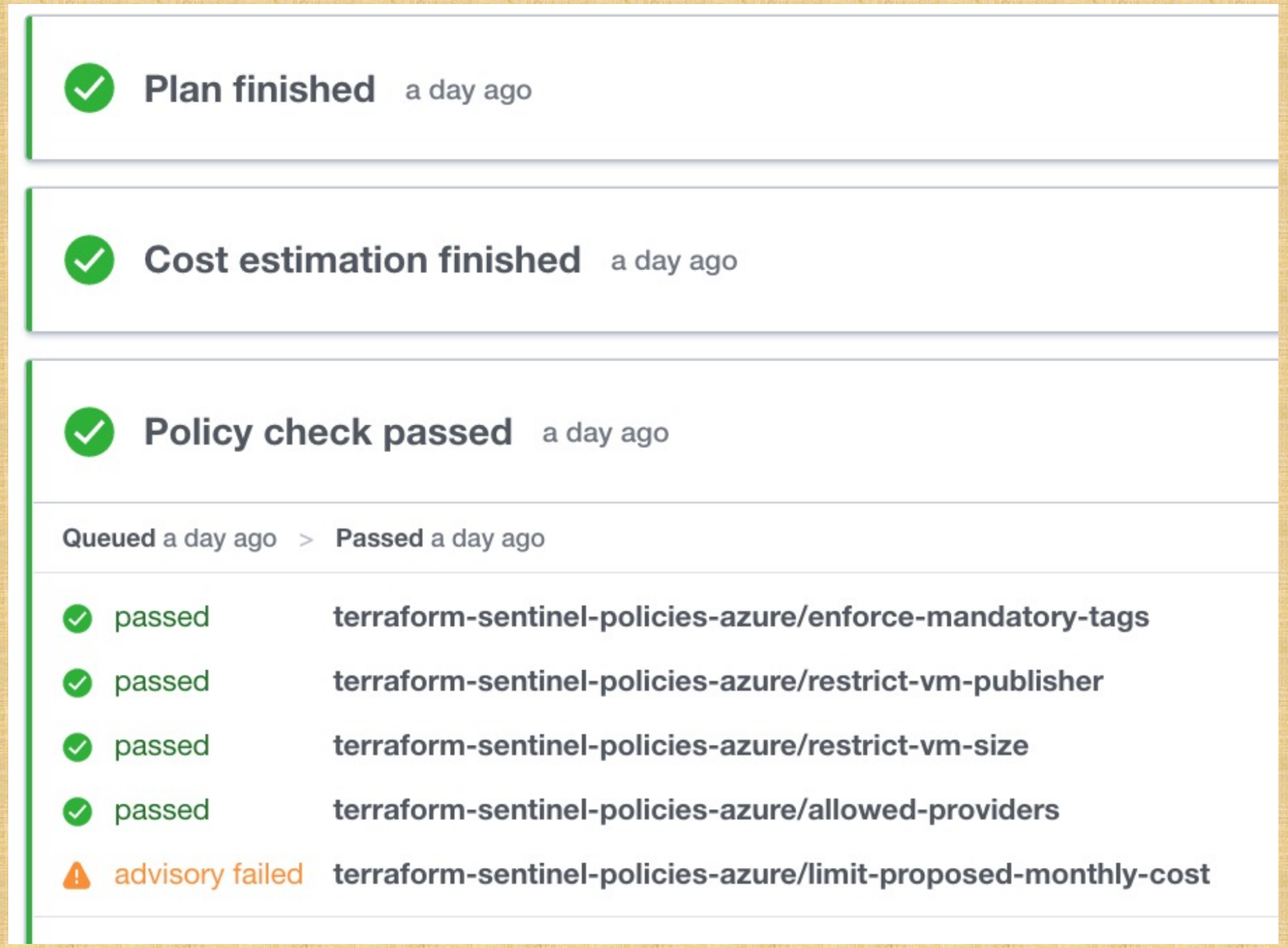
Sentinel is an embedded policy-as-code framework integrated with the HashiCorp Enterprise products.

It enables fine-grained, logic-based policy decisions, and can be extended to use information from external sources.

Author: Nho Luong

Skill: DevOps Engineer Lead

Sentinel Policies

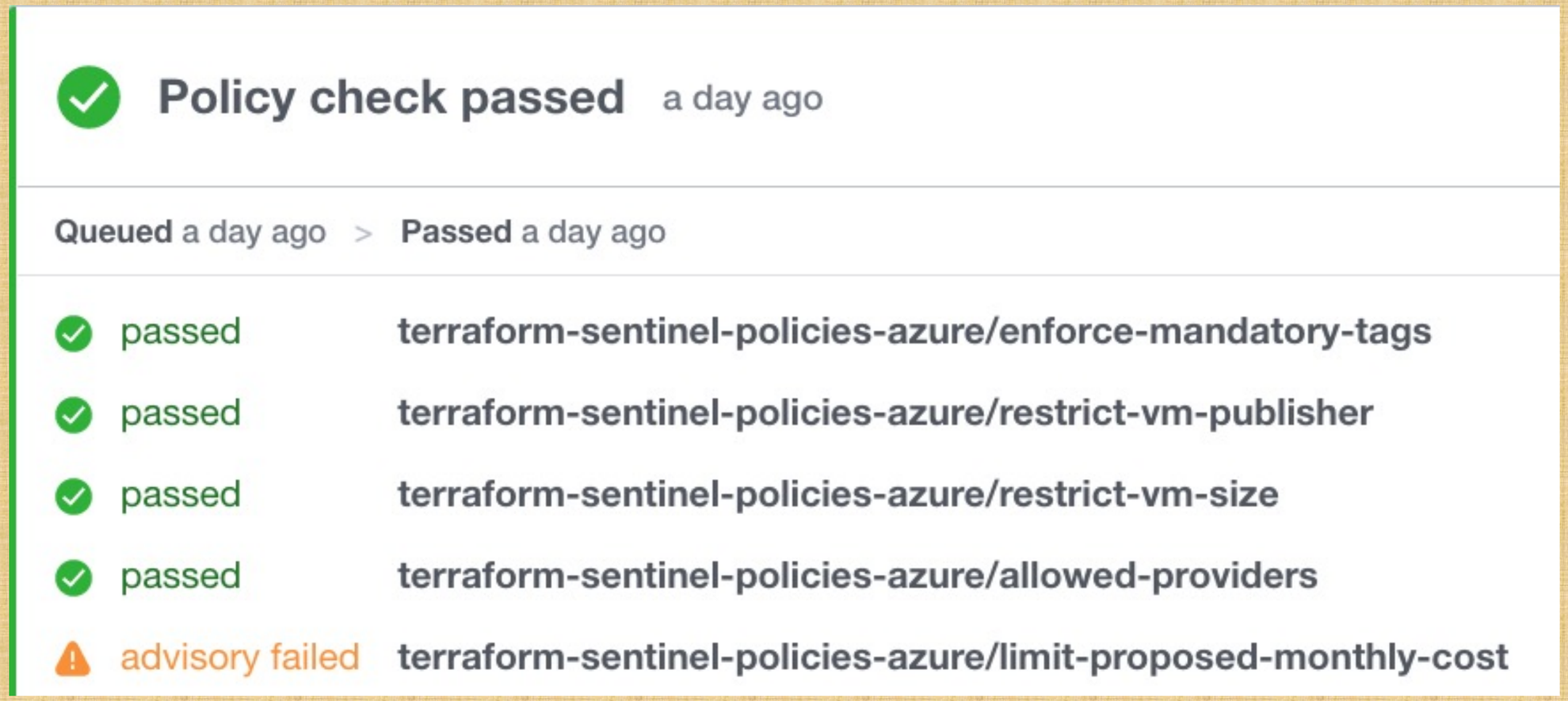


Sentinel Policies

Author: Nho Luong

Skill: DevOps Engineer Lead

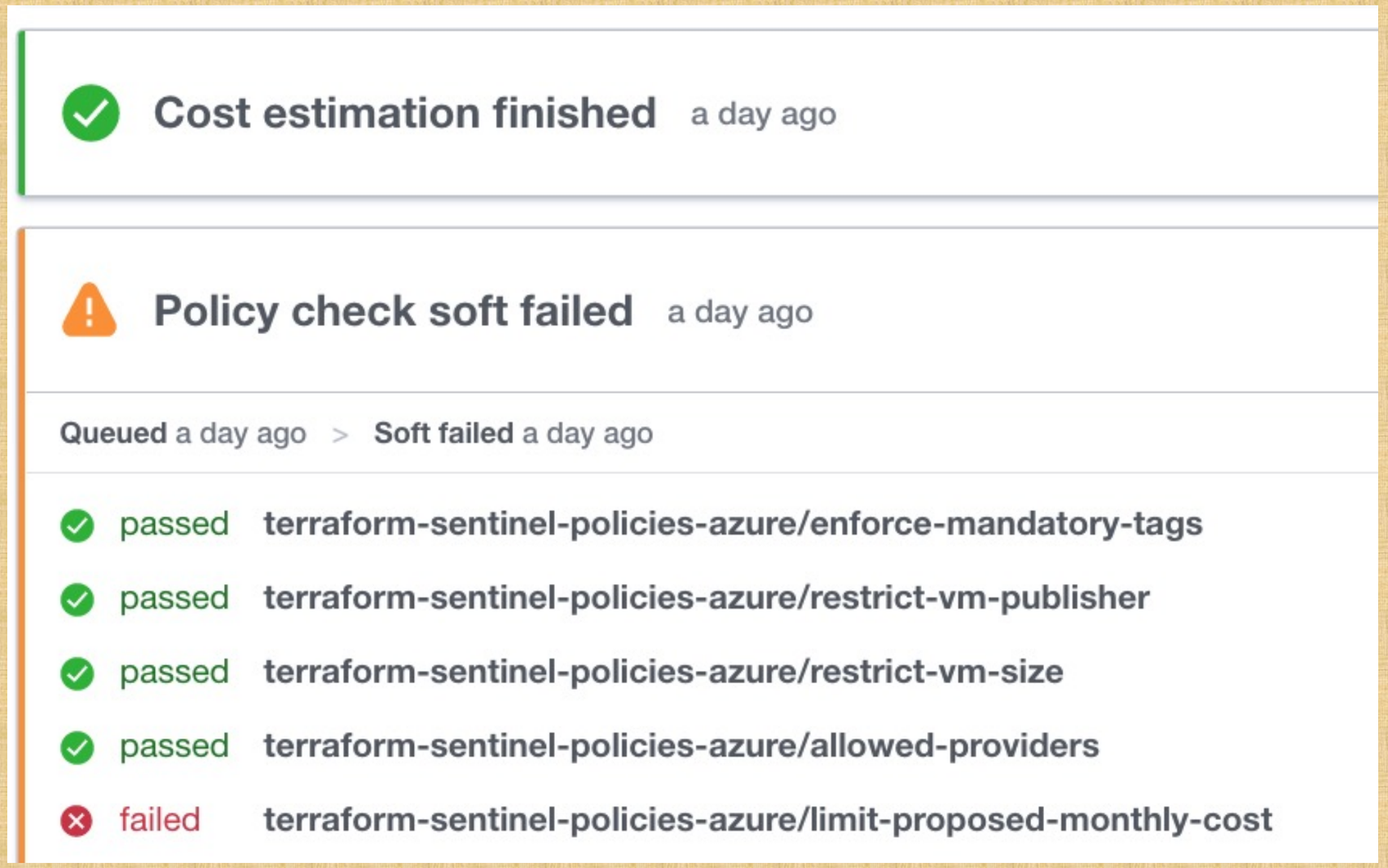
Sentinel Enforcement Mode - Advisory



Author: Nho Luong

Skill: DevOps Engineer Lead

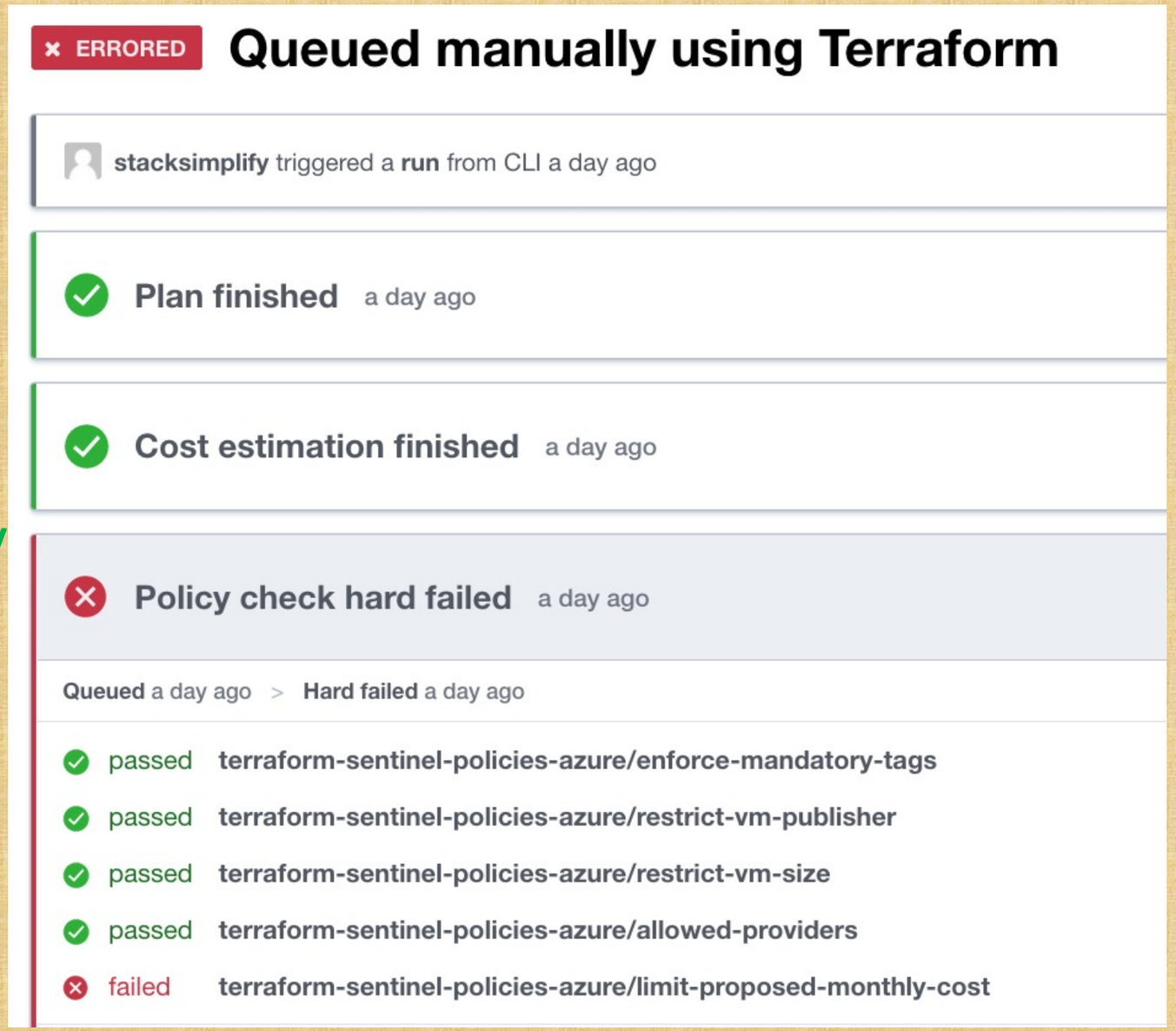
Sentinel Enforcement Mode – Soft-Mandatory



Author: Nho Luong

Skill: DevOps Engineer Lead

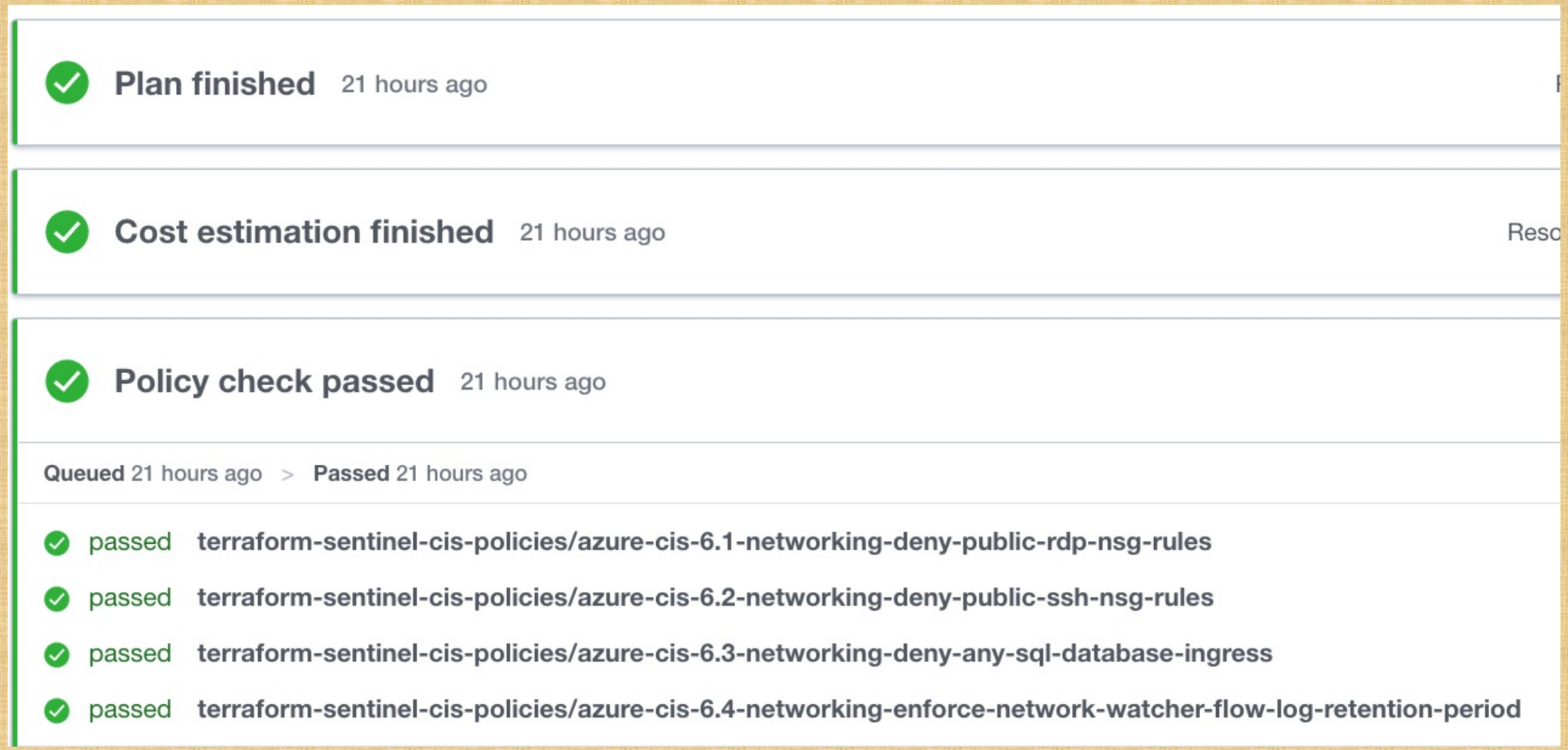
Sentinel Enforcement Mode – Hard-Mandatory



Author: Nho Luong

Skill: DevOps Engineer Lead

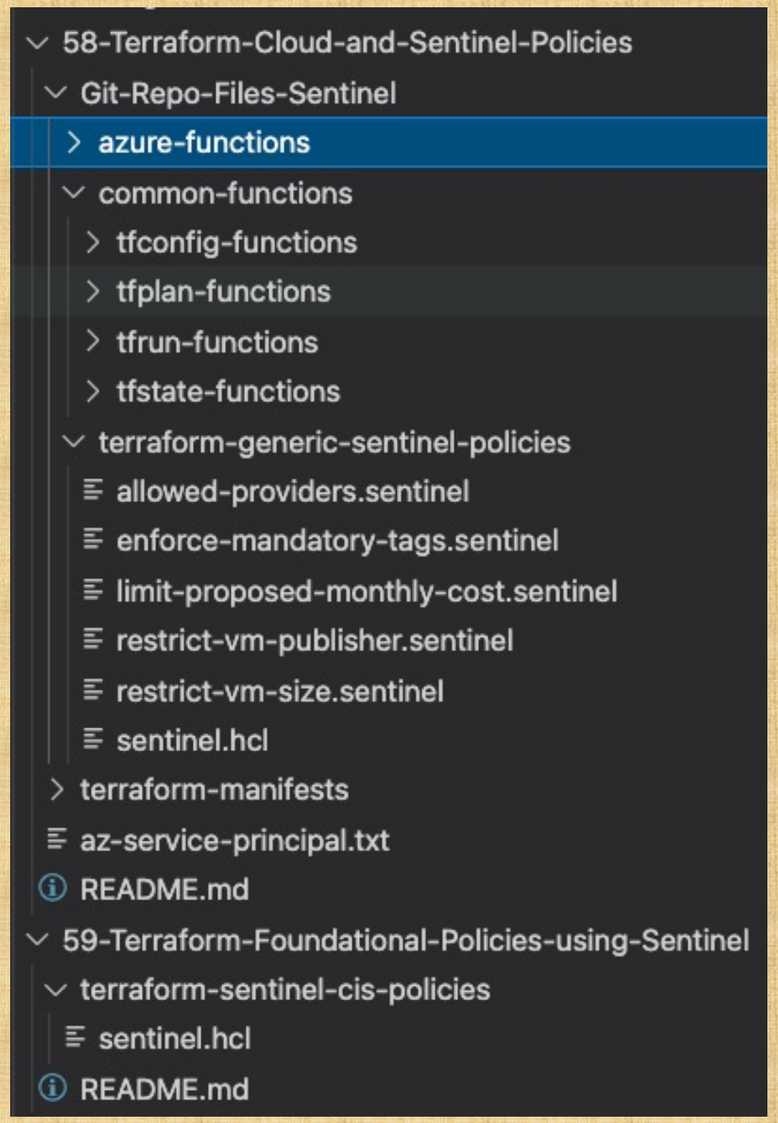
Sentinel – CIS Policies

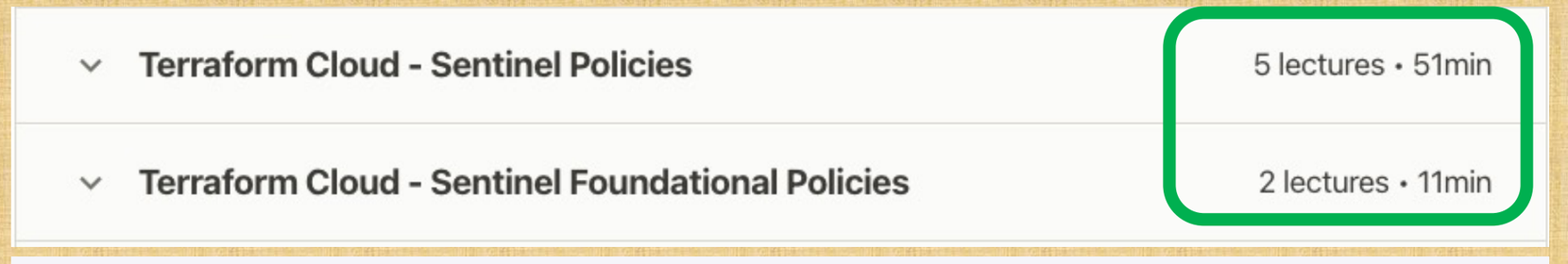


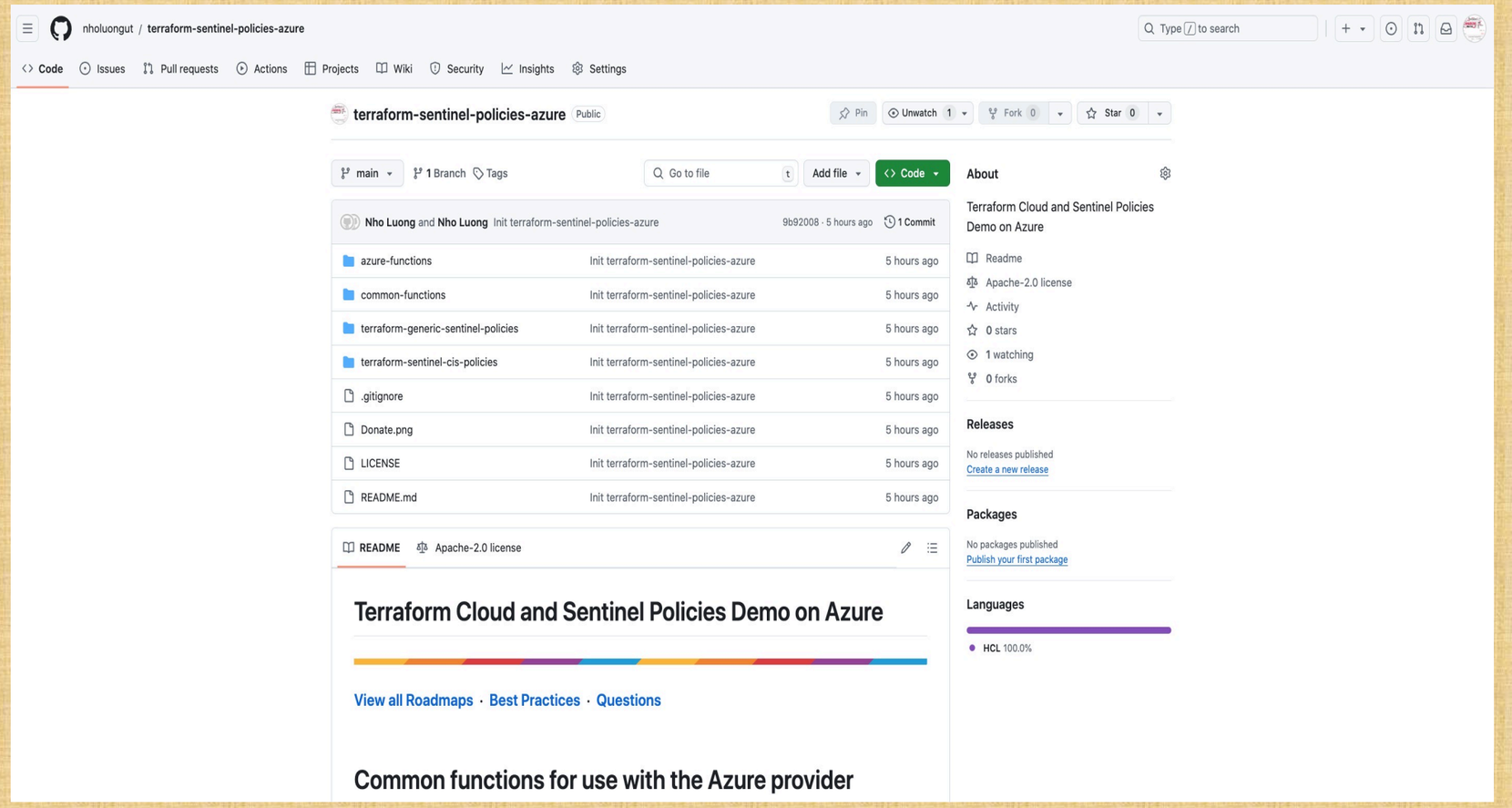
Author: Nho Luong

Skill: DevOps Engineer Lead

Practical Examples with Step-by-Step Github Documentation







Author: Nho Luong

Skill: DevOps Engineer Lead

Thank You

Author: Nho Luong

Skill: DevOps Engineer Lead