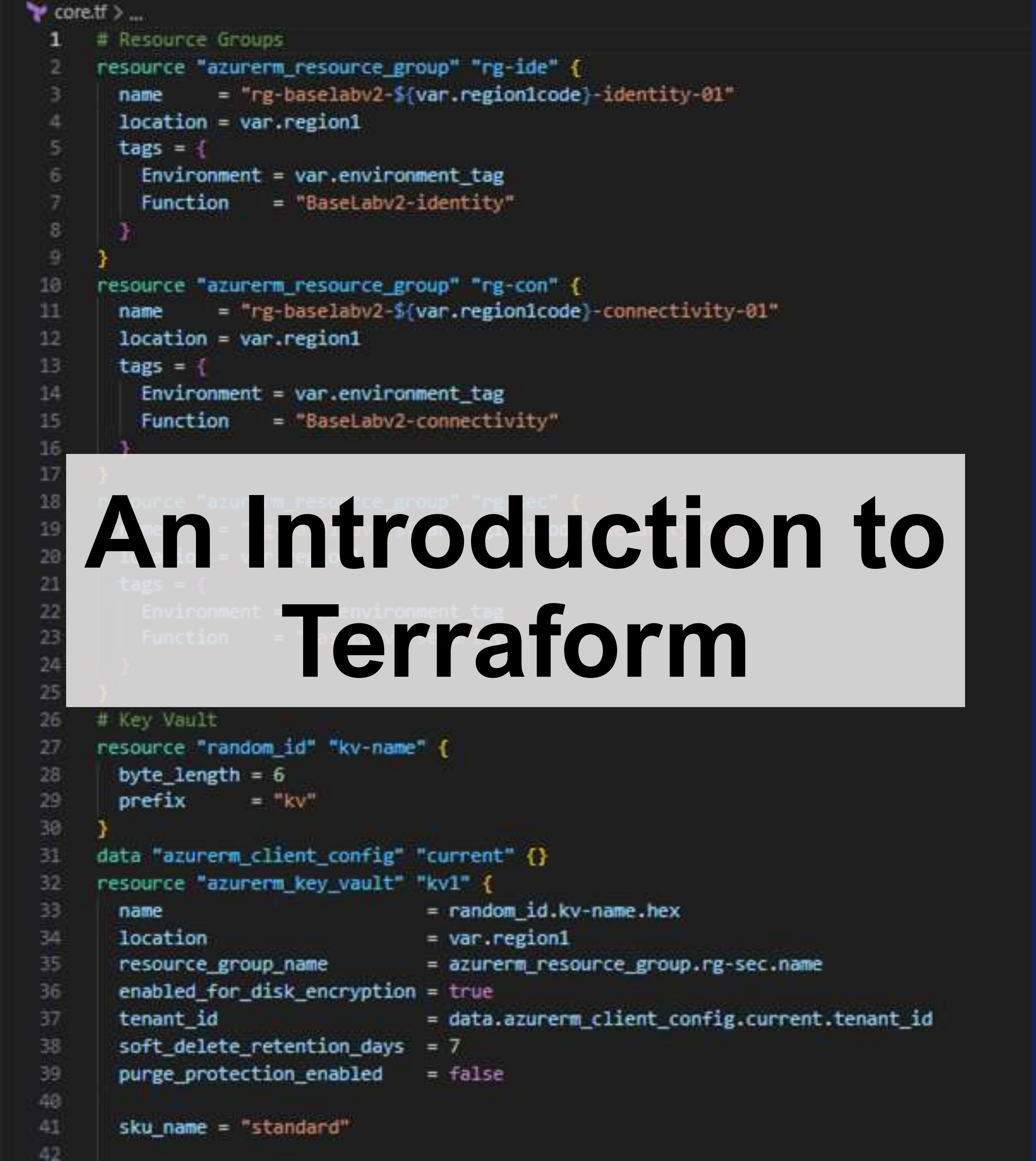
An Introduction to Terraform



core.tf > ...

```
# Resource Groups
resource "azurerm_resource_group" "rg-ide" {
  name     = "rg-baselabv2-${var.region1code}-identity-01"
  location = var.region1
  tags = {
    Environment = var.environment_tag
    Function    = "BaseLabv2-identity"
  }
}
resource "azurerm_resource_group" "rg-con" {
  name     = "rg-baselabv2-${var.region1code}-connectivity-01"
  location = var.region1
  tags = {
    Environment = var.environment_tag
    Function    = "BaseLabv2-connectivity"
  }
}
resource "azurerm_resource_group" "rg-sec" {
  name     = [illegible]
  location = [illegible]
  tags = {
    Environment = var.environment_tag
    Function    = [illegible]
  }
}
# Key Vault
resource "random_id" "kv-name" {
  byte_length = 6
  prefix      = "kv"
}
data "azurerm_client_config" "current" {}
resource "azurerm_key_vault" "kv1" {
  name                        = random_id.kv-name.hex
  location                    = var.region1
  resource_group_name         = azurerm_resource_group.rg-sec.name
  enabled_for_disk_encryption = true
  tenant_id                   = data.azurerm_client_config.current.tenant_id
  soft_delete_retention_days  = 7
  purge_protection_enabled    = false

  sku_name = "standard"
```

euc FORUM

```
core.tf > ...
# Resource Groups

  sku_name = "standard"
```

Hello!

Jake Walsh

Senior Solution Architect @ CDW UK

@jakewalsh90
jakewalsh.co.uk

Please note – the views/opinions in this presentation are entirely my own. If in any doubt, please check latest documentation etc!

EUC FORUM

Microsoft® MVP Most Valuable Professional

HashiCorp AMBASSADOR 2023

CTA Citrix Technology Advocate

Session Goals

- **What** is Infrastructure as Code – and **why** use it?
- What is **Terraform**?
- How Terraform Works & Why it's relevant for **EUC Environments**
- **Getting Started** – Installation
- Code Samples / Demo

What is Infrastructure as Code (IAC)?

✓ A method of managing and provisioning infrastructure resources via code.

✓ In most cases either uses **imperative** or **declarative** code.

$$\begin{tikzcd} \text{imperative} \arrow[d] & \text{declarative} \arrow[d] \\ \text{“do this”} & \text{“build this”} \end{tikzcd}$$

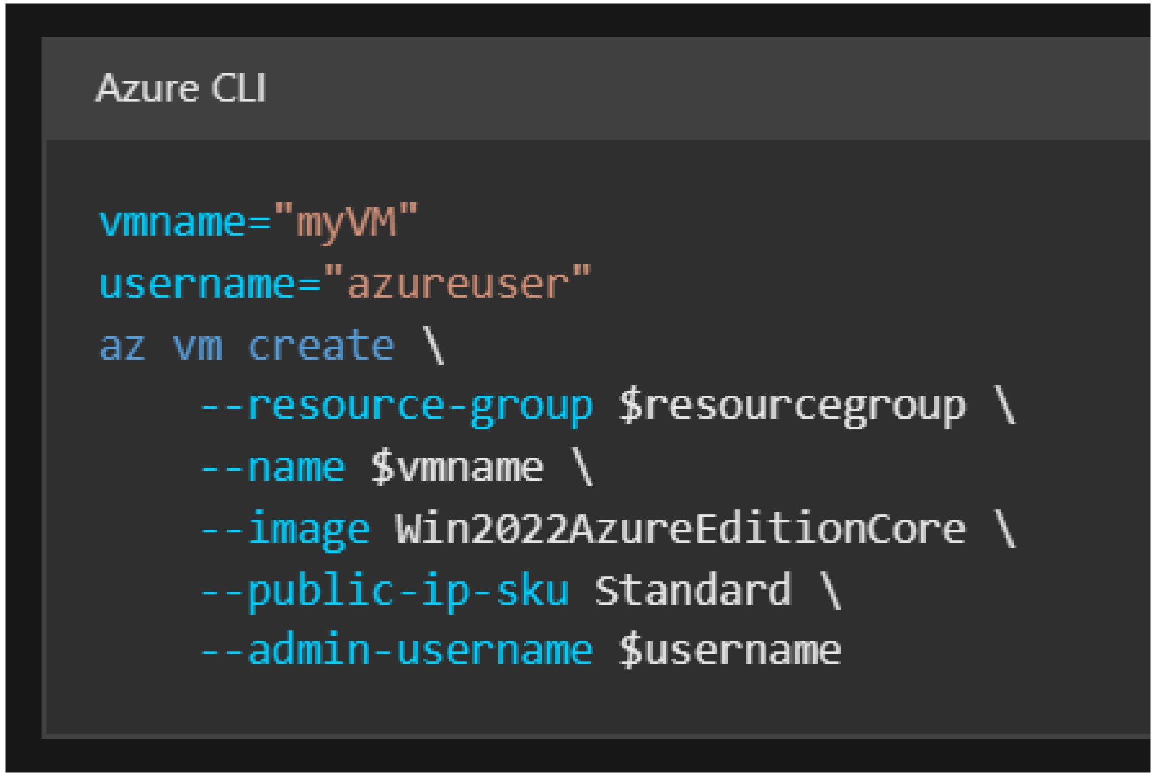
✓ Often integrated into version control systems – e.g. Git.

✓ Can be edited and managed in most common tools and platforms – e.g. GitHub, Visual Studio Code, Azure DevOps etc.

✓ Usually adopted as part of a wider DevOps Strategy.

✓ Allows a move away from ClickOps and provides options to version control infrastructure resources.

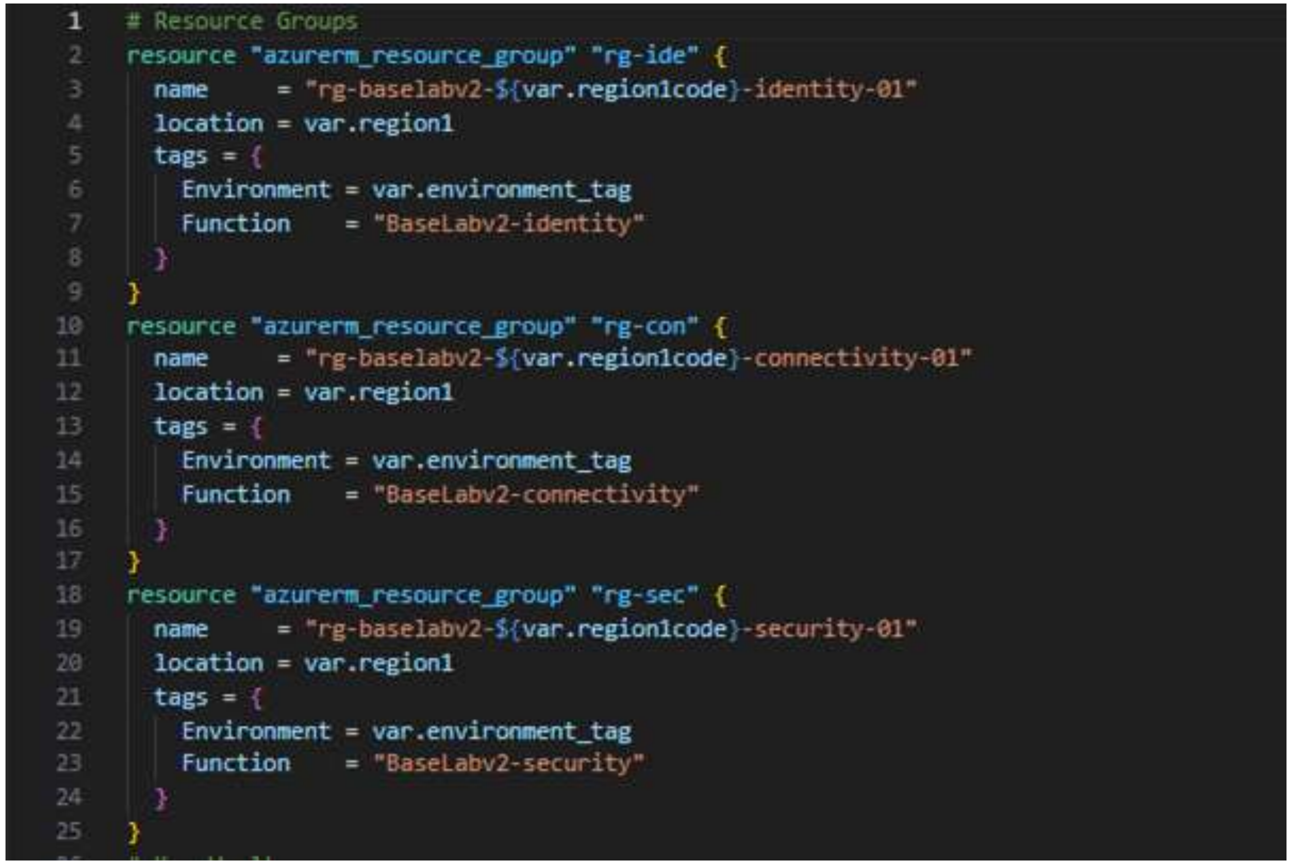
Imperative


Azure CLI

```
vmname="myVM"
username="azureuser"
az vm create \
    --resource-group $resourcegroup \
    --name $vmname \
    --image Win2022AzureEditionCore \
    --public-ip-sku Standard \
    --admin-username $username
```

- Defines a task to be carried out
- In this example repeated executions would error – as the VM already exists after 1 run

Declarative



```
# Resource Groups
resource "azurerm_resource_group" "rg-ide" {
  name     = "rg-baselabv2-${var.region1code}-identity-01"
  location = var.region1
  tags = {
    Environment = var.environment_tag
    Function    = "BaseLabv2-identity"
  }
}
resource "azurerm_resource_group" "rg-con" {
  name     = "rg-baselabv2-${var.region1code}-connectivity-01"
  location = var.region1
  tags = {
    Environment = var.environment_tag
    Function    = "BaseLabv2-connectivity"
  }
}
resource "azurerm_resource_group" "rg-sec" {
  name     = "rg-baselabv2-${var.region1code}-security-01"
  location = var.region1
  tags = {
    Environment = var.environment_tag
    Function    = "BaseLabv2-security"
  }
}
```

- Defines infrastructure components to be created
- In this example repeated executions would result in a message informing us that “no changes” are required (as the VM is already built).

Why use Infrastructure as Code?

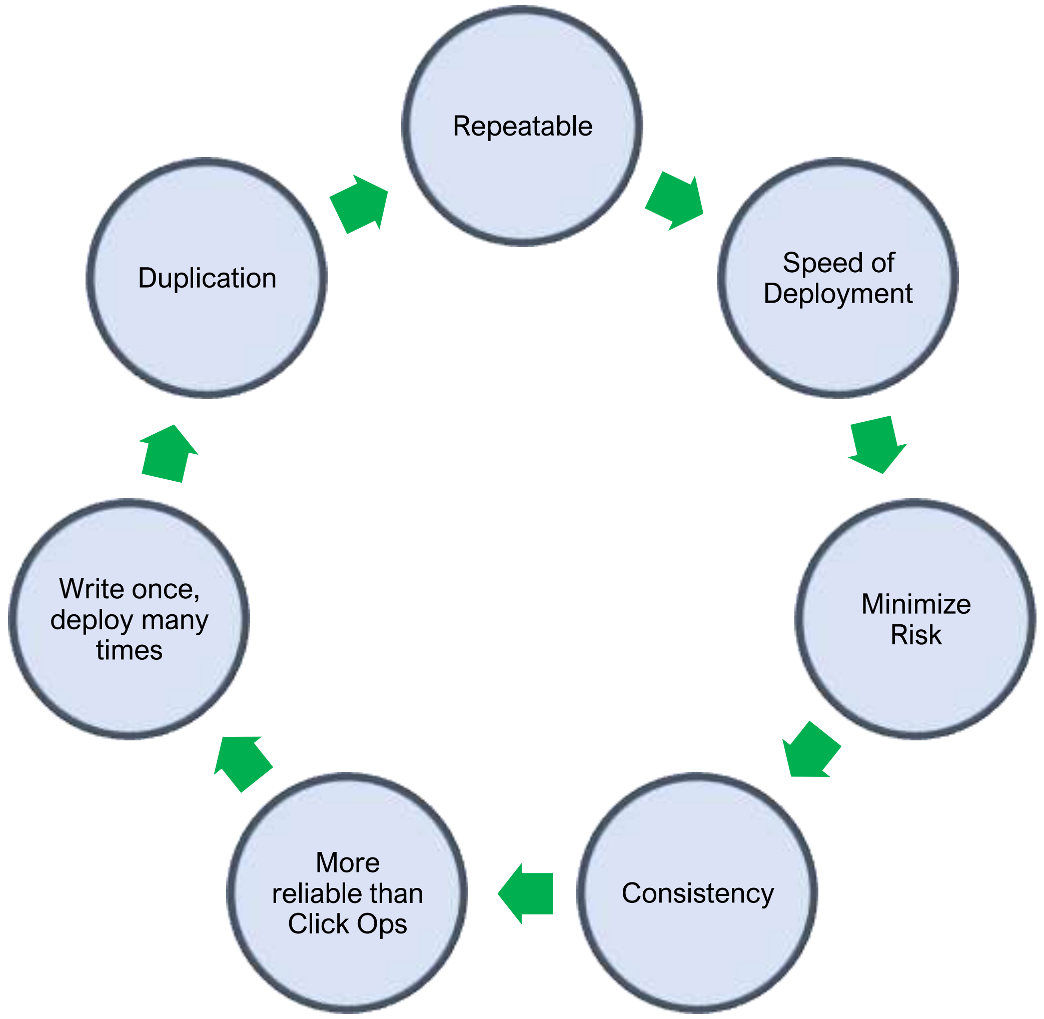
Cost – enables more rapid deployment, changes, test environments etc.

Speed – faster deployment due to less manual intervention (no ClickOps), easy testing, less human error etc. Enables DevOps methods/practices.

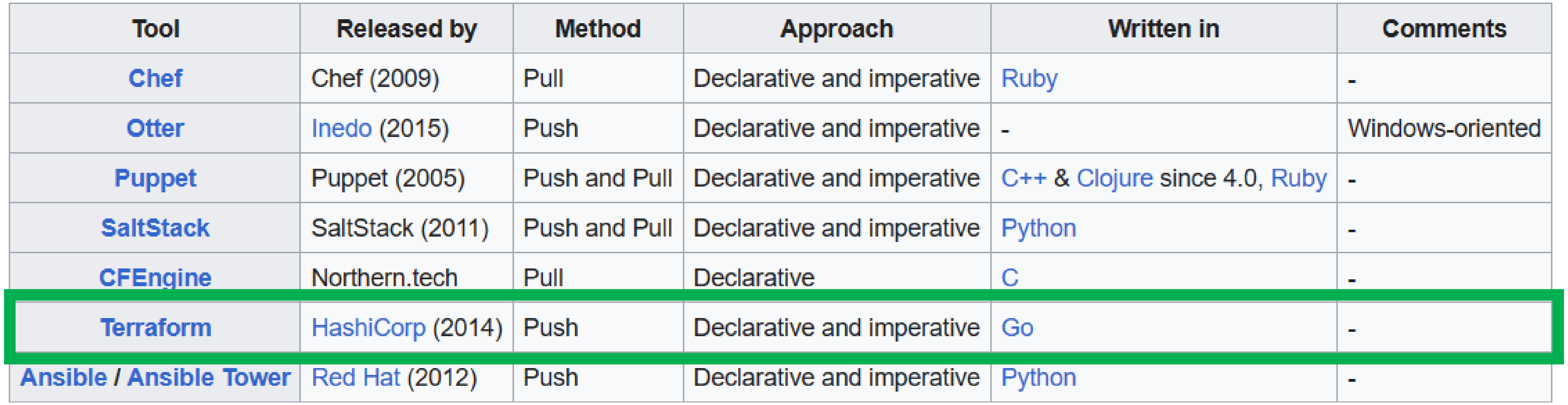
Risk – reduced through testing, consistency of deployments, version control etc.

Why use Infrastructure as Code?

Benefits Cycle



Other platforms are available…

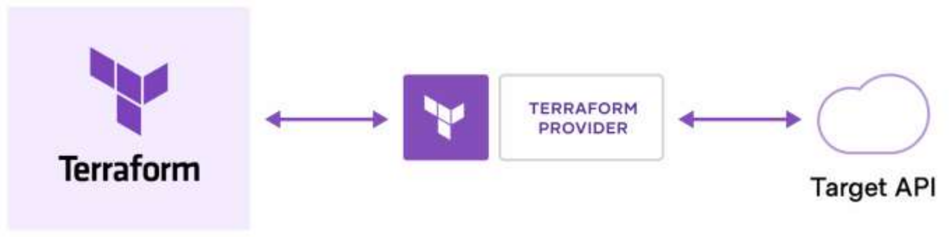


Tool	Released by	Method	Approach	Written in	Comments
Chef	Chef (2009)	Pull	Declarative and imperative	Ruby	-
Otter	Inedo (2015)	Push	Declarative and imperative	-	Windows-oriented
Puppet	Puppet (2005)	Push and Pull	Declarative and imperative	C++ & Clojure since 4.0, Ruby	-
SaltStack	SaltStack (2011)	Push and Pull	Declarative and imperative	Python	-
CFEngine	Northern.tech	Pull	Declarative	C	-
Terraform	HashiCorp (2014)	Push	Declarative and imperative	Go	-
Ansible / Ansible Tower	Red Hat (2012)	Push	Declarative and imperative	Python	-

https://en.wikipedia.org/wiki/Infrastructure_as_code

What is Terraform?

- Terraform is an **<u>Infrastructure as Code</u>** Software tool, that can interact with a wide range of Platforms and Environments, using Providers.



AWS, VMware, Azure, etc

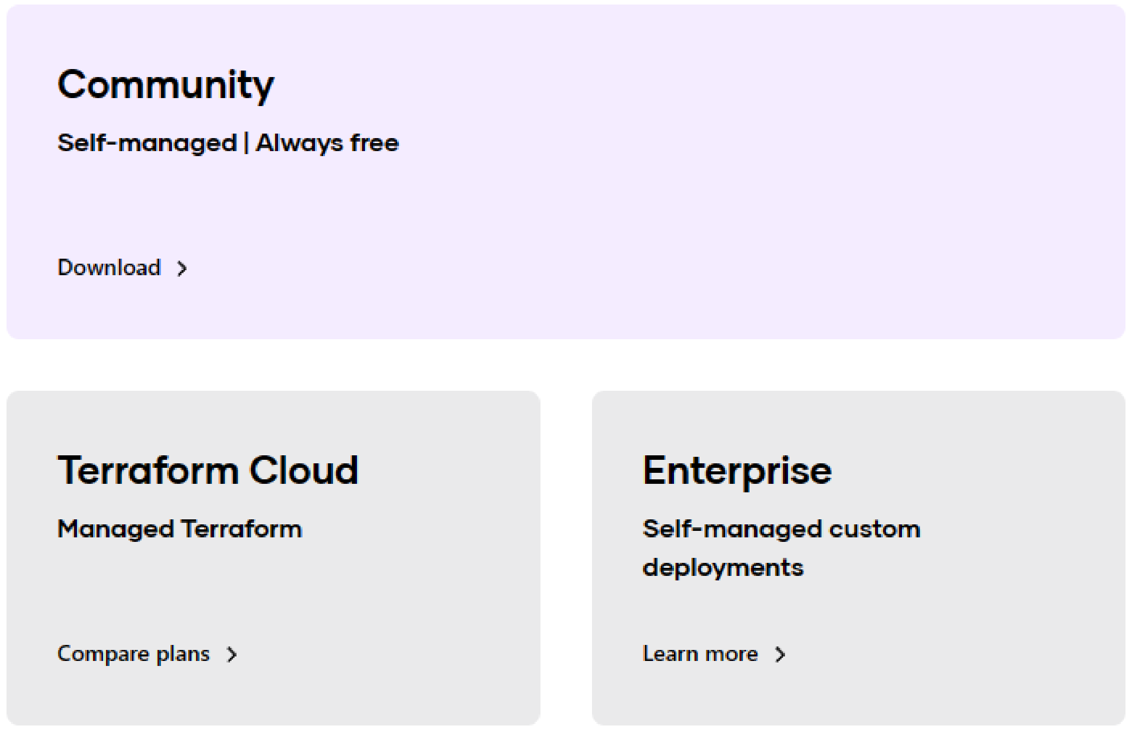
- Can be used in both Cloud and On-Premises environments. Can be used to combine on-premises and Cloud, or Cloud and Cloud for example.

- Terraform comes in 3 main varieties:
    - Community Edition – I will be using this to demo today!
    - Terraform Cloud
    - Terraform Enterprise

https://developer.hashicorp.com/terraform/intro

What is Terraform?

- Terraform comes in 3 main varieties:
    - Community Edition – I will be using this to demo today!
    - Terraform Cloud
    - Terraform Enterprise



Community

Self-managed | Always free

Download >

Terraform Cloud

Managed Terraform

Compare plans >

Enterprise

Self-managed custom deployments

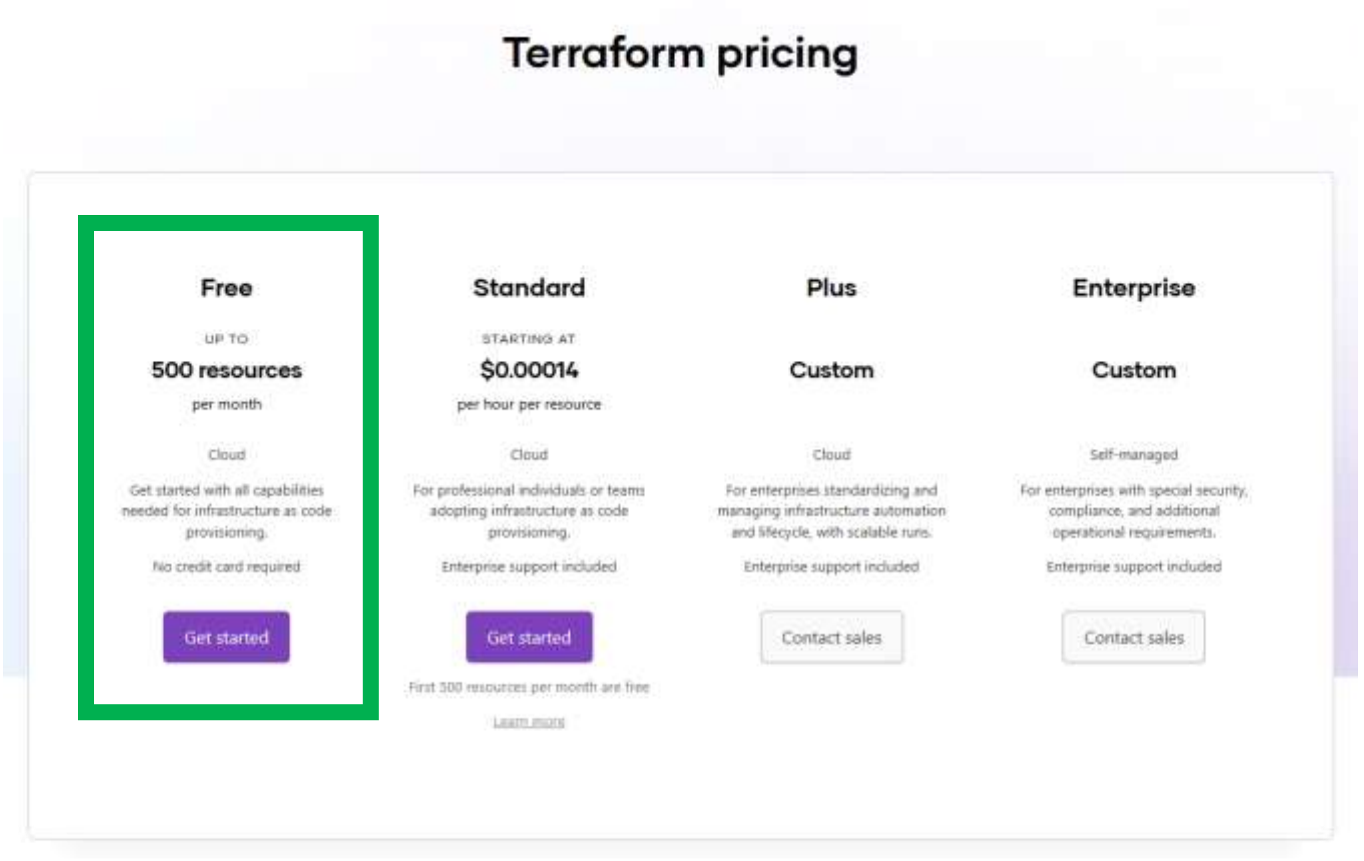
Learn more >

https://www.terraform.io/

What is Terraform?

HashiCorp Terraform

- Terraform Cloud
- Terraform Enterprise



Terraform pricing

Free	Standard	Plus	Enterprise
UP TO	STARTING AT		
500 resources	$0.00014	Custom	Custom
per month	per hour per resource		
Cloud	Cloud	Cloud	Self-managed
Get started with all capabilities needed for infrastructure as code provisioning.	For professional individuals or teams adopting infrastructure as code provisioning.	For enterprises standardizing and managing infrastructure automation and lifecycle, with scalable runs.	For enterprises with special security, compliance, and additional operational requirements.
No credit card required	Enterprise support included	Enterprise support included	Enterprise support included
Get started	Get started	Contact sales	Contact sales
	First 500 resources per month are free		
	Learn more		

Authentication

- **Demo / Lab Environments**

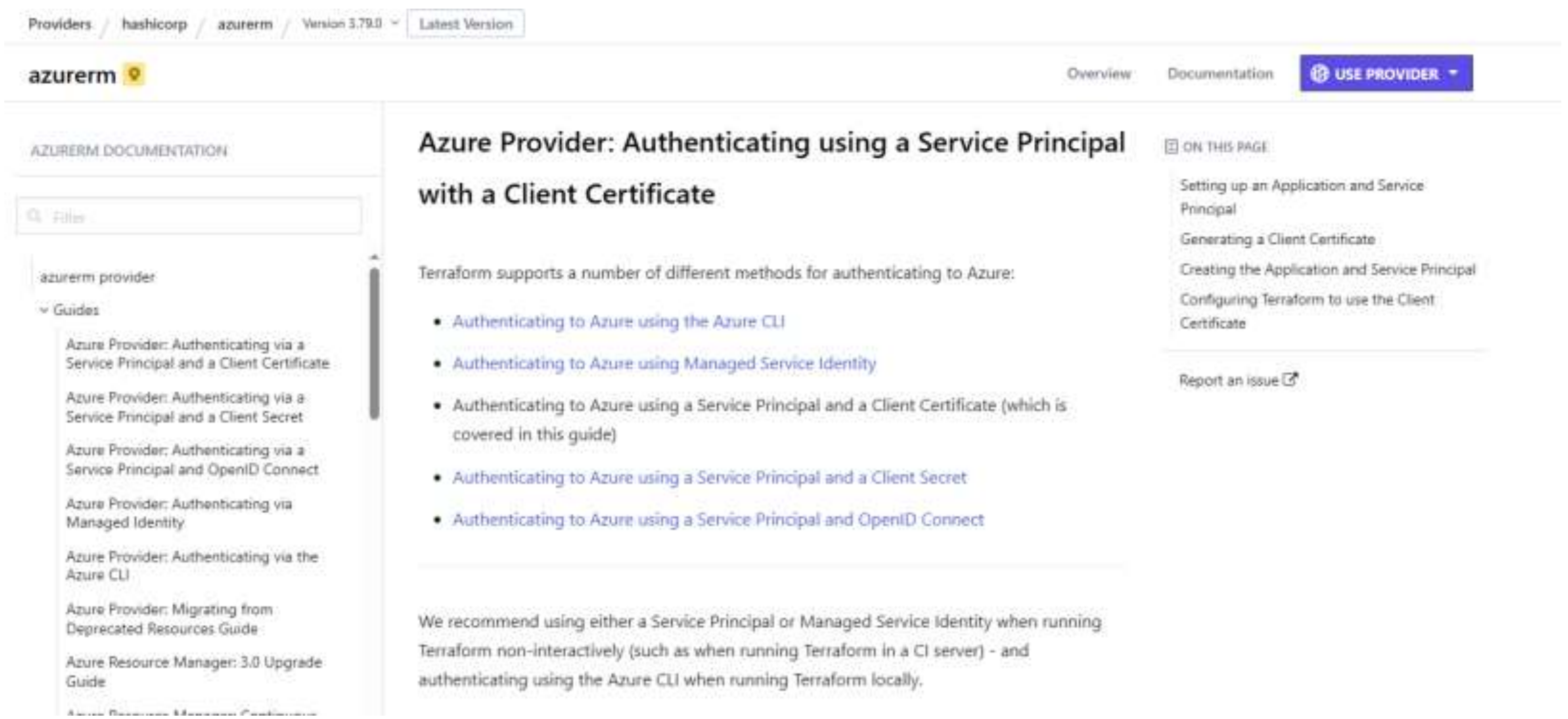
  Usually authenticate at the CLI or use a Service Principal

- **Production Environments**

  Service Principal or a Managed Service Identity

https://learn.microsoft.com/en-us/azure/developer/terraform/authenticate-to-azure?tabs=bash

Authentication



Providers / hashicorp / azurerm / Version 3.79.0 Latest Version

azurerm

Overview Documentation USE PROVIDER

AZURERM DOCUMENTATION

Filter

azurerm provider

Guides

Azure Provider: Authenticating via a Service Principal and a Client Certificate

Azure Provider: Authenticating via a Service Principal and a Client Secret

Azure Provider: Authenticating via a Service Principal and OpenID Connect

Azure Provider: Authenticating via Managed Identity

Azure Provider: Authenticating via the Azure CLI

Azure Provider: Migrating from Deprecated Resources Guide

Azure Resource Manager: 3.0 Upgrade Guide

Azure Provider: Authenticating using a Service Principal with a Client Certificate

Terraform supports a number of different methods for authenticating to Azure:

- Authenticating to Azure using the Azure CLI
- Authenticating to Azure using Managed Service Identity
- Authenticating to Azure using a Service Principal and a Client Certificate (which is covered in this guide)
- Authenticating to Azure using a Service Principal and a Client Secret
- Authenticating to Azure using a Service Principal and OpenID Connect

We recommend using either a Service Principal or Managed Service Identity when running Terraform non-interactively (such as when running Terraform in a CI server) - and authenticating using the Azure CLI when running Terraform locally.

ON THIS PAGE

Setting up an Application and Service Principal

Generating a Client Certificate

Creating the Application and Service Principal

Configuring Terraform to use the Client Certificate

Report an issue

Providers

- Before we can run Terraform, we need to add a “Provider” to our Code. Providers are plugins for Terraform that allow Terraform to interact with an external API.

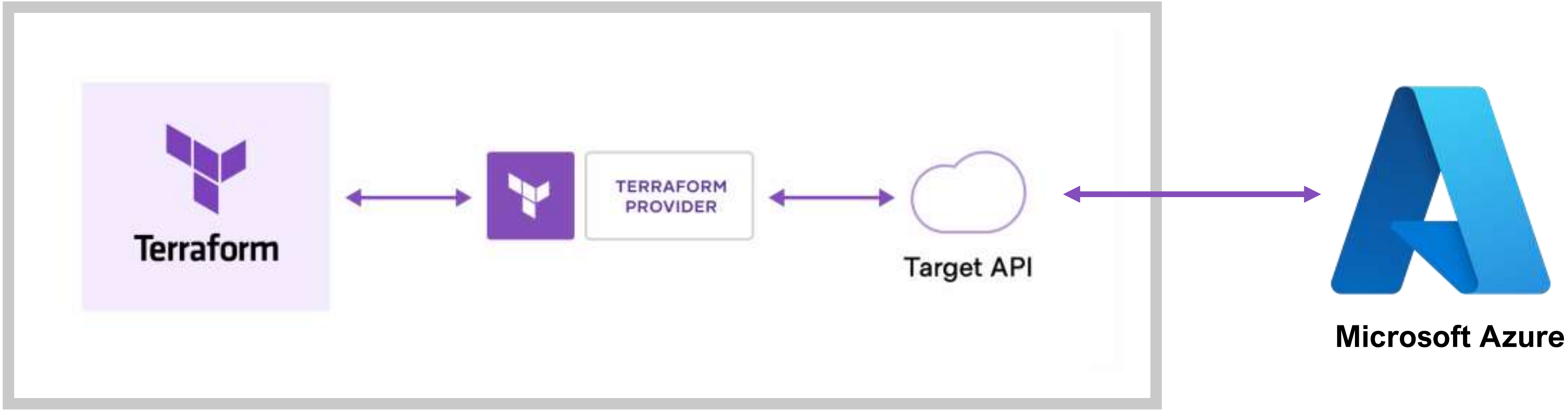
  https://registry.terraform.io/browse/providers

- In simple Terms – providers enable communication with platforms or services outside of Terraform

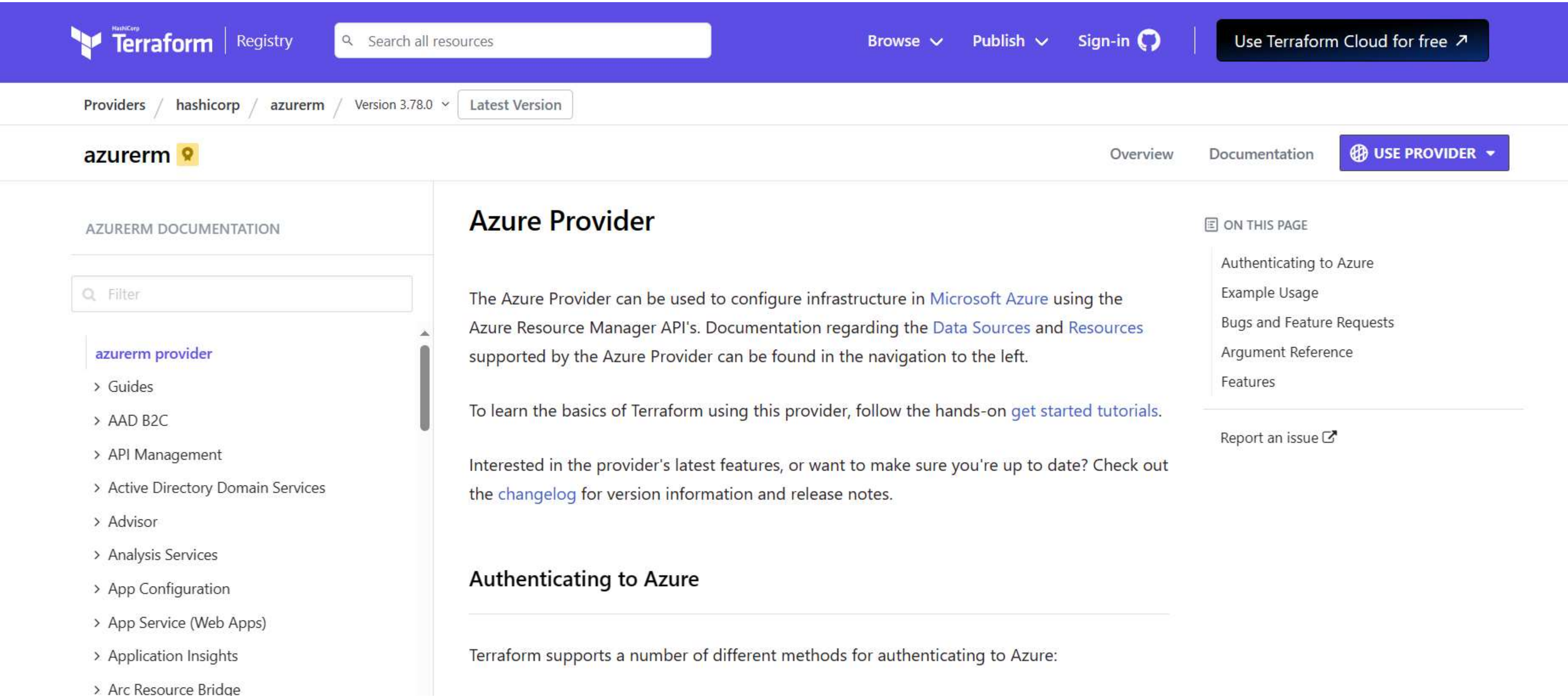
- For example – with Microsoft Azure, we would need to add the AzureRM Provider to Terraform before we can interact with Azure.

  https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs

AzureRM Provider



https://registry.terraform.io/providers/hashicorp/azurerm/latest/docs


HashiCorp Terraform | Registry
Search all resources
Browse
Publish
Sign-in
Use Terraform Cloud for free ↗
Providers / hashicorp / azurerm / Version 3.78.0
Latest Version
azurerm
Overview
Documentation
USE PROVIDER

AZURERM DOCUMENTATION

Filter

- azurerm provider
- Guides
- AAD B2C
- API Management
- Active Directory Domain Services
- Advisor
- Analysis Services
- App Configuration
- App Service (Web Apps)
- Application Insights
- Arc Resource Bridge

Azure Provider

The Azure Provider can be used to configure infrastructure in Microsoft Azure using the Azure Resource Manager API's. Documentation regarding the Data Sources and Resources supported by the Azure Provider can be found in the navigation to the left.

To learn the basics of Terraform using this provider, follow the hands-on get started tutorials.

Interested in the provider's latest features, or want to make sure you're up to date? Check out the changelog for version information and release notes.

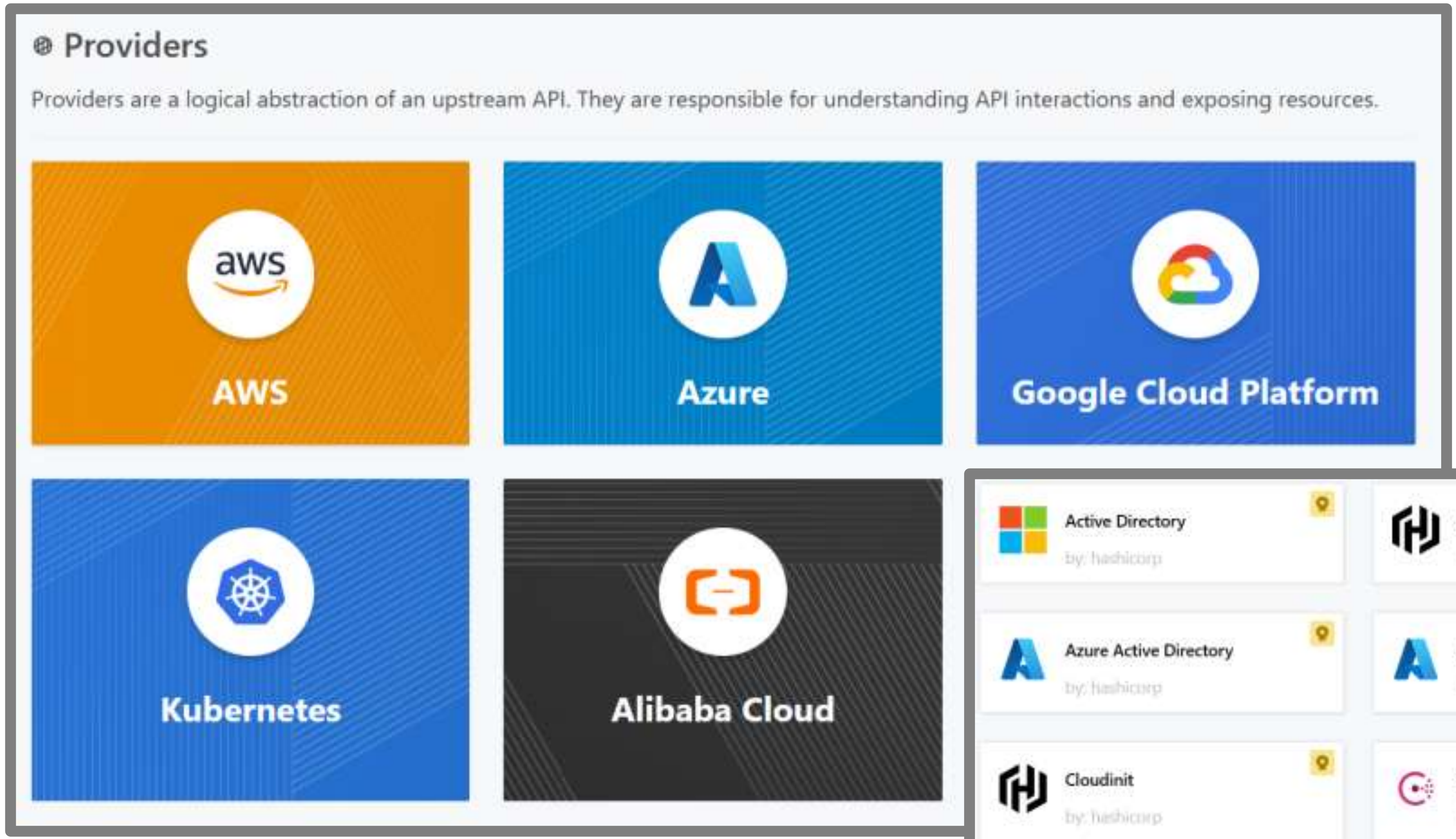
Authenticating to Azure

Terraform supports a number of different methods for authenticating to Azure:

ON THIS PAGE

- Authenticating to Azure
- Example Usage
- Bugs and Feature Requests
- Argument Reference
- Features

Report an issue



Providers

Providers are a logical abstraction of an upstream API. They are responsible for understanding API interactions and exposing resources.

AWS

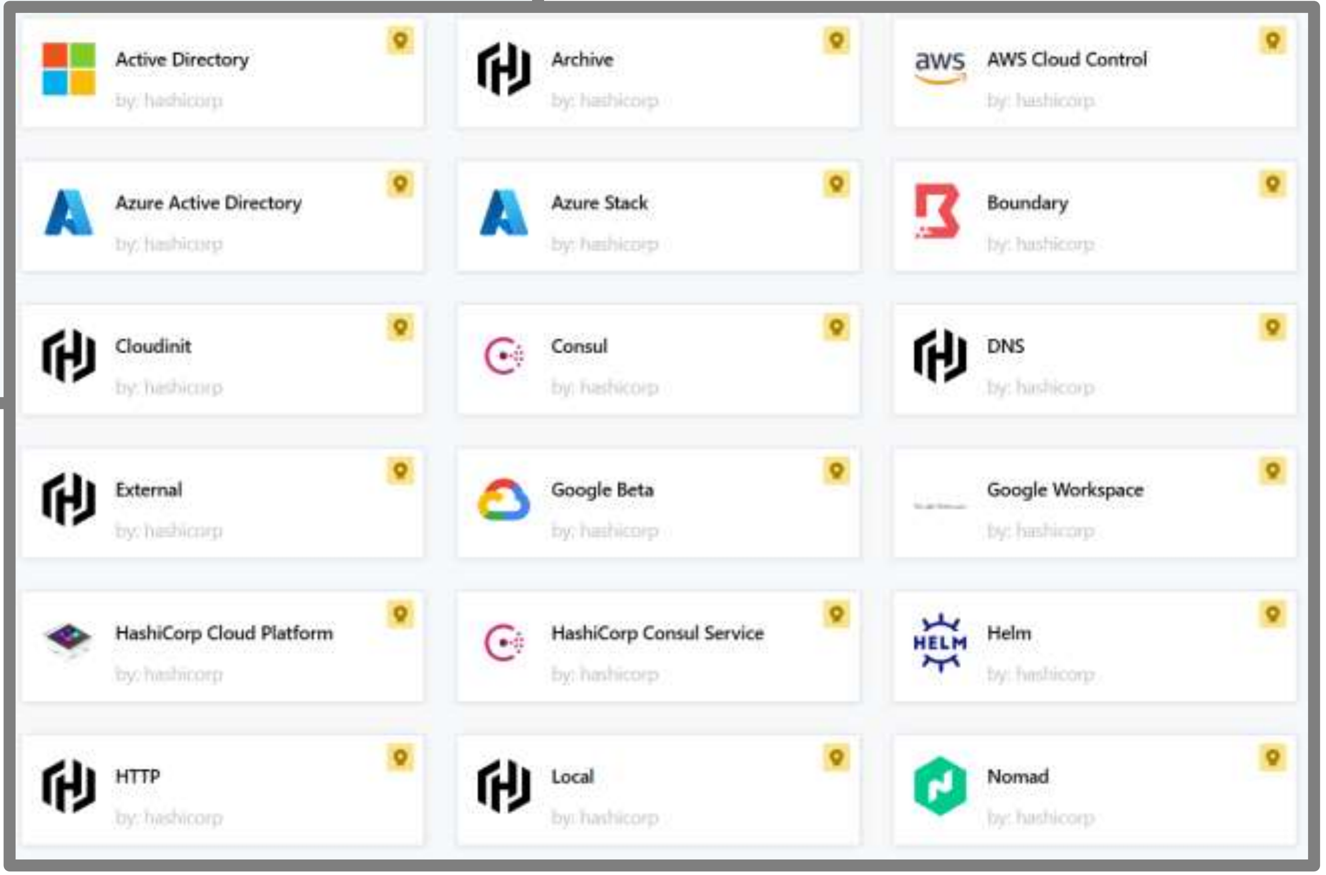
Azure

Google Cloud Platform

Kubernetes

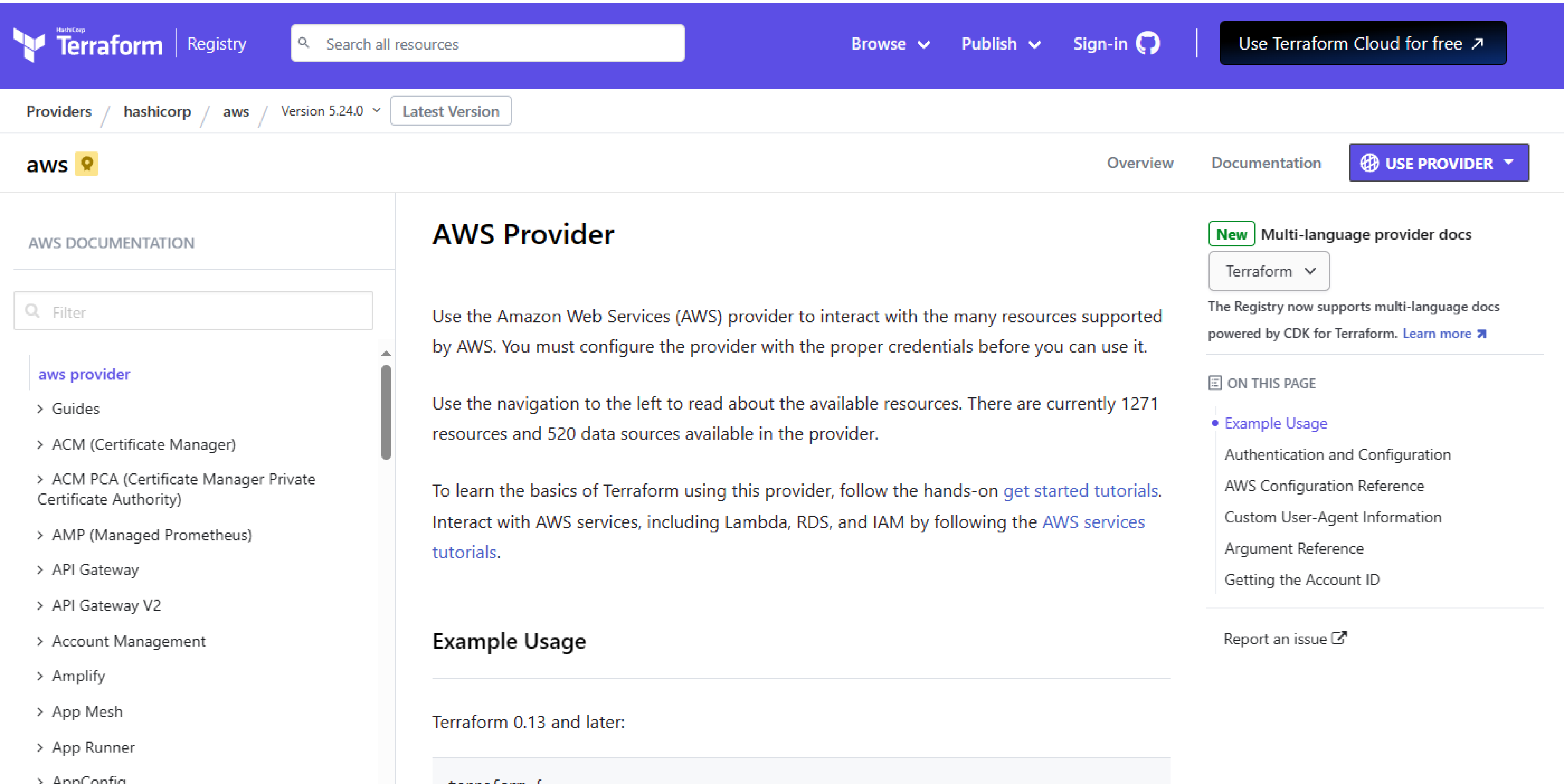
Alibaba Cloud

What other Providers are available?



Active Directory by: hashicorp	Archive by: hashicorp	AWS Cloud Control by: hashicorp
Azure Active Directory by: hashicorp	Azure Stack by: hashicorp	Boundary by: hashicorp
CloudInit by: hashicorp	Consul by: hashicorp	DNS by: hashicorp
External by: hashicorp	Google Beta by: hashicorp	Google Workspace by: hashicorp
HashiCorp Cloud Platform by: hashicorp	HashiCorp Consul Service by: hashicorp	Helm by: hashicorp
HTTP by: hashicorp	Local by: hashicorp	Nomad by: hashicorp

https://registry.terraform.io/browse/providers



Providers / hashicorp / aws / Version 5.24.0 Latest Version

aws

AWS DOCUMENTATION

- aws provider
- Guides
- ACM (Certificate Manager)
- ACM PCA (Certificate Manager Private Certificate Authority)
- AMP (Managed Prometheus)
- API Gateway
- API Gateway V2
- Account Management
- Amplify
- App Mesh
- App Runner
- AppConfig

AWS Provider

Use the Amazon Web Services (AWS) provider to interact with the many resources supported by AWS. You must configure the provider with the proper credentials before you can use it.

Use the navigation to the left to read about the available resources. There are currently 1271 resources and 520 data sources available in the provider.

To learn the basics of Terraform using this provider, follow the hands-on get started tutorials. Interact with AWS services, including Lambda, RDS, and IAM by following the AWS services tutorials.

Example Usage

Terraform 0.13 and later:

```
terraform {
```

New Multi-language provider docs

The Registry now supports multi-language docs powered by CDK for Terraform. Learn more

ON THIS PAGE

- Example Usage
- Authentication and Configuration
- AWS Configuration Reference
- Custom User-Agent Information
- Argument Reference
- Getting the Account ID

Report an issue

Process

1. Terraform code is typically arranged across a number of files, known as “tf files”. (Because they have the extension TF)

↓

2. These files define the infrastructure and its configuration (or changes!) that we want Terraform to apply.

↓

3. At the time of running Terraform, these files are analysed by Terraform, and turned into an execution plan to apply our changes.

Terraform Stages

- Running Terraform involves a number of stages of deployment:

`Terraform init`	This stage initialises the Terraform binaries, and downloads the required providers, based on what we have defined.
`Terraform plan`	This stage examines our TF files and provides an overview of the infrastructure changes – by providing an execution plan.
`Terraform apply`	This stage carries out the execution plan, and implements the changes. Note: this also runs plan.
`Terraform destroy`	This stage destroys the created infrastructure – use carefully!

The State File

Terraform must store information about your infrastructure within a file known as the “State File”.

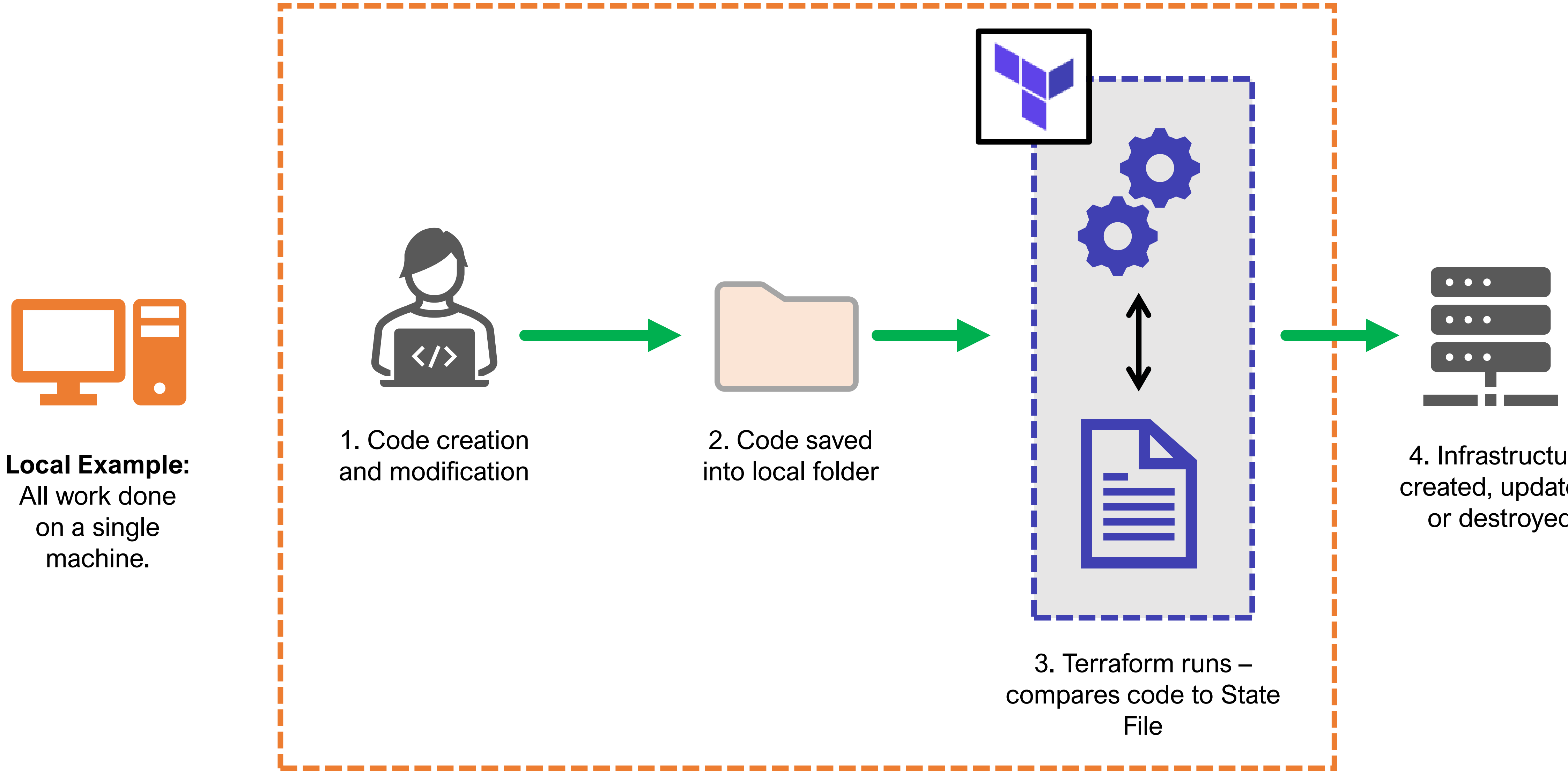
This is so that Terraform can work out changes required to the infrastructure based on your code or configuration changes.

The State File can be stored locally, or remotely, depending on the deployment type and needs.

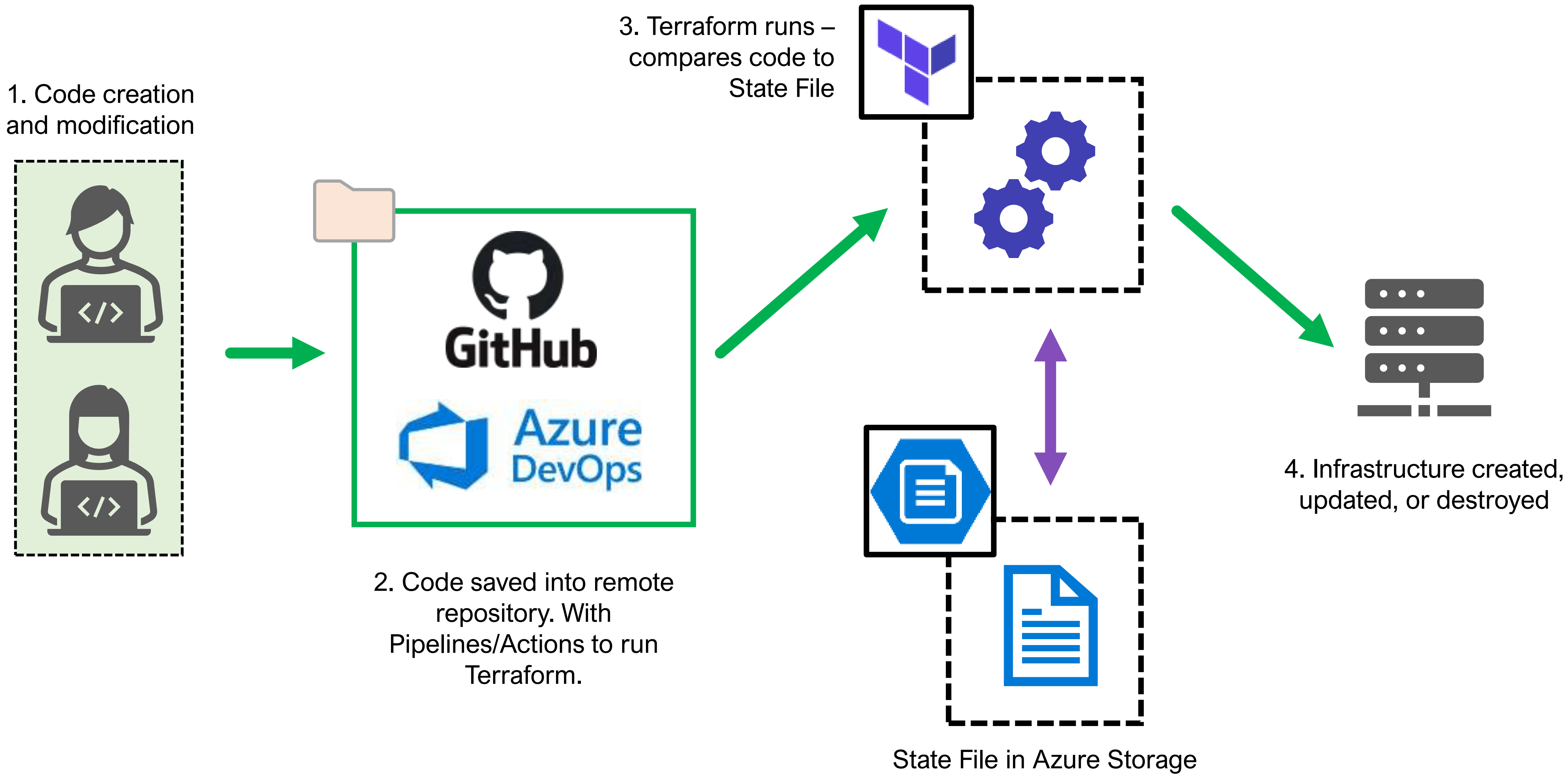
- Local State – learning/testing/labs/development ✓
- Remote State – using DevOps tooling or collaborating on code

https://developer.hashicorp.com/terraform/language/state

Ways of Working – Local Example

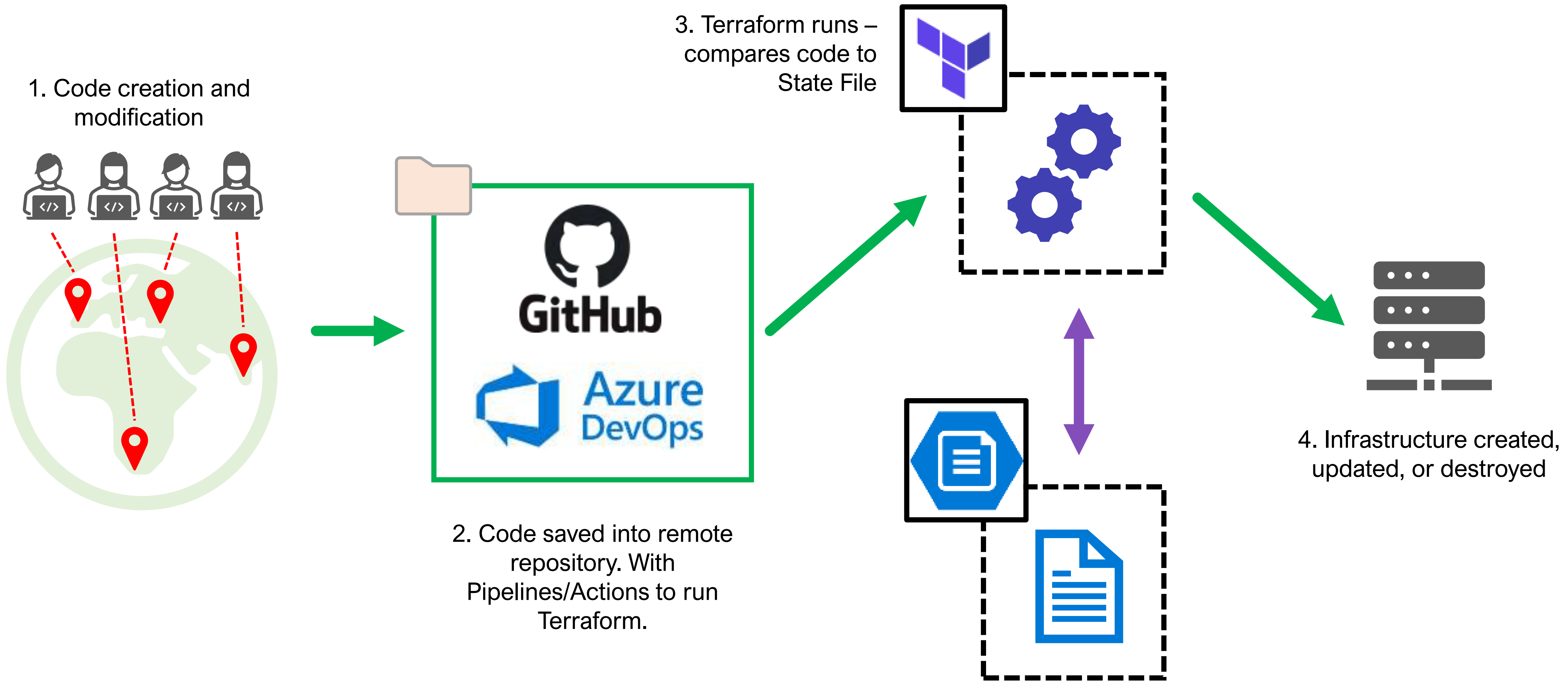

Local Example:
All work done on a single machine.
1. Code creation and modification
2. Code saved into local folder
3. Terraform runs – compares code to State File
4. Infrastructure created, updated, or destroyed

Ways of Working – A Remote Example



1. Code creation and modification
GitHub
Azure DevOps
2. Code saved into remote repository. With Pipelines/Actions to run Terraform.
3. Terraform runs – compares code to State File
State File in Azure Storage
4. Infrastructure created, updated, or destroyed

Ways of Working – An even more Remote Example



1. Code creation and modification

2. Code saved into remote repository. With Pipelines/Actions to run Terraform.

3. Terraform runs – compares code to State File

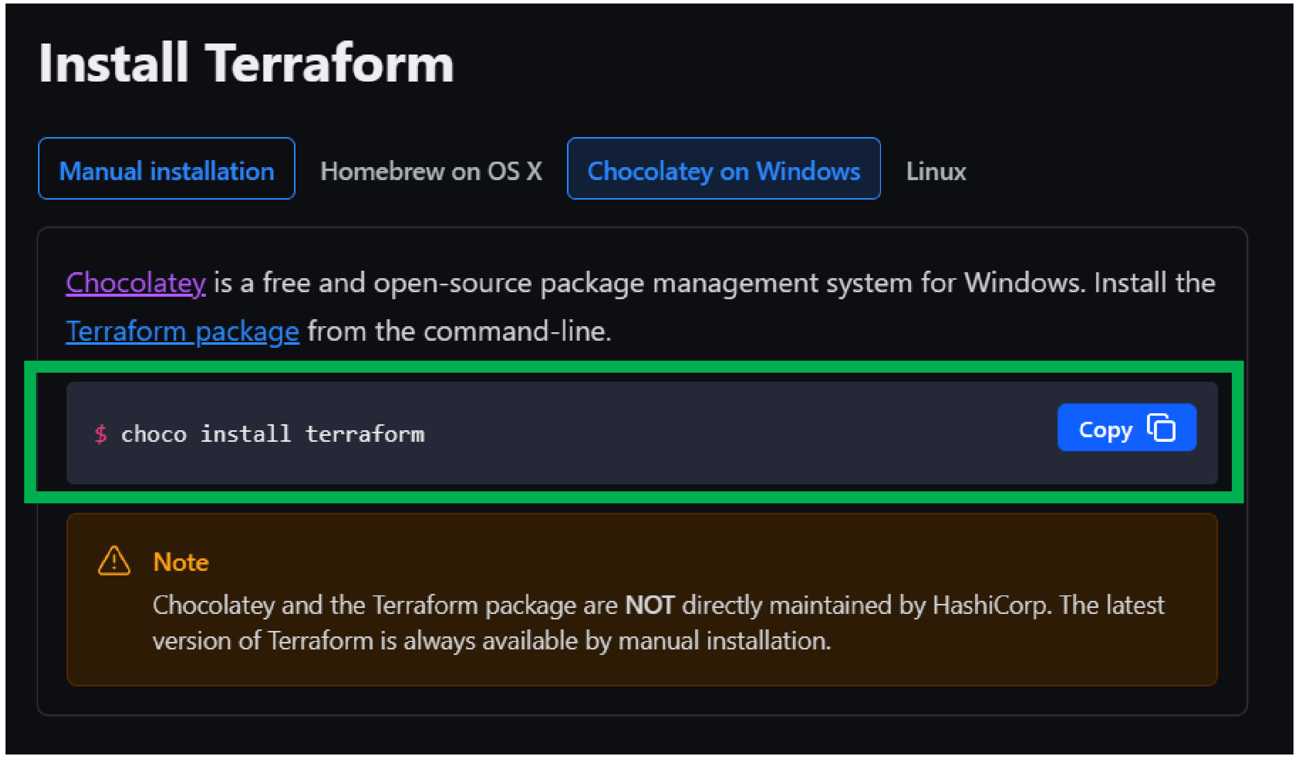
4. Infrastructure created, updated, or destroyed

State File in Azure Storage

Why is Terraform useful in the EUC World?

- Create infrastructure easily, repeatably, in different locations/platforms/regions.
- Enables Rapid Development / Testing
- Cost Effective Test Environments – create on demand, destroy once used.
- Scale up/down/in/out as required
- Expansion – use functions like count and variable methods like maps
- Work safely and in a standardised way across distributed teams.
- Write once, deploy many times.
- **Enables Version control of EUC Infrastructure**

Installing Terraform



Chocolatey is a free and open-source package management system for Windows. Install the Terraform package from the command-line.

```
$ choco install terraform
```

Note
Chocolatey and the Terraform package are **NOT** directly maintained by HashiCorp. The latest version of Terraform is always available by manual installation.

Recommended minimum software:

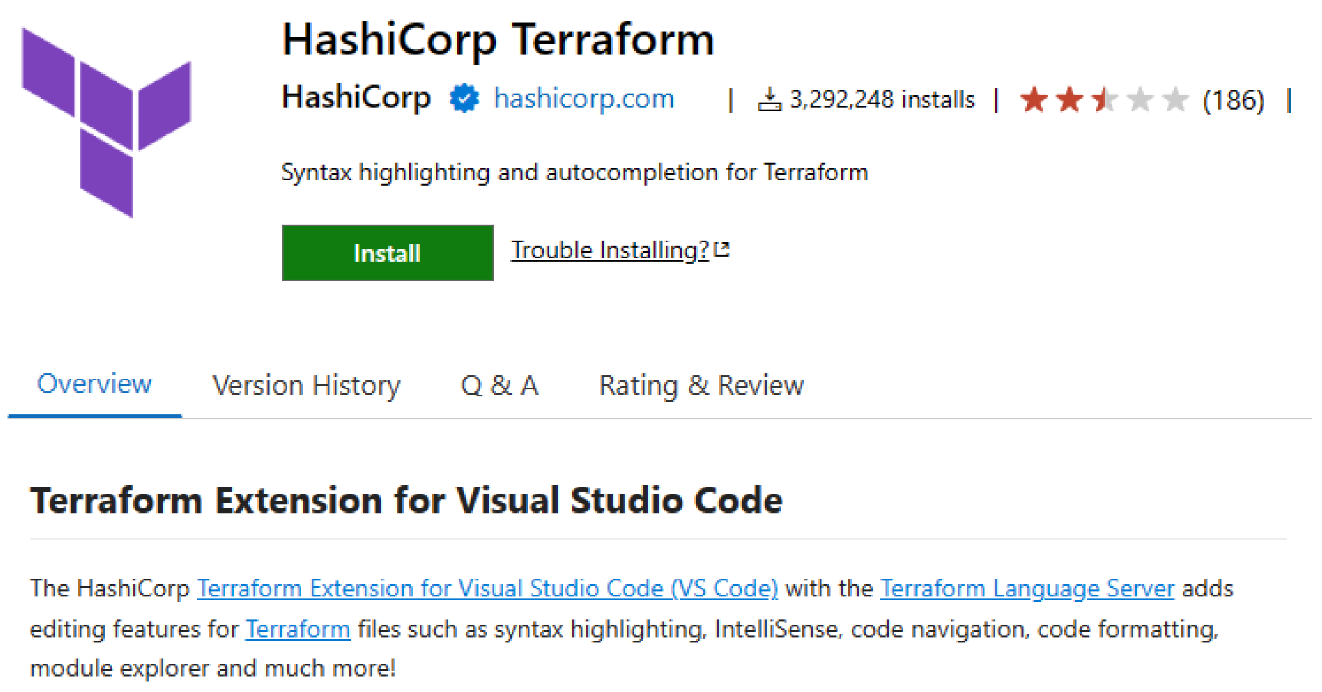
- Terraform
- Visual Studio Code + Extensions

https://community.chocolatey.org/

```
choco install terraform -y
choco install azure-cli -y
choco install vscode -y
```

https://developer.hashicorp.com/terraform/tutorials/azure-get-started/install-cli

Installing Terraform – VSCode Plugin



HashiCorp Terraform

HashiCorp hashicorp.com | 3,292,248 installs | (186) |

Syntax highlighting and autocompletion for Terraform

Terraform Extension for Visual Studio Code

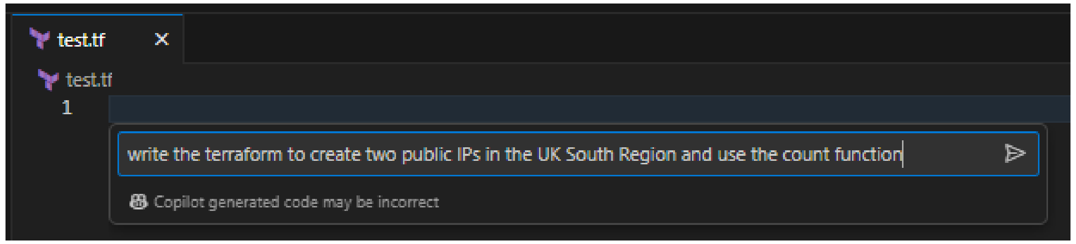
The HashiCorp Terraform Extension for Visual Studio Code (VS Code) with the Terraform Language Server adds editing features for Terraform files such as syntax highlighting, IntelliSense, code navigation, code formatting, module explorer and much more!

Features:

- Intellisense
- Syntax Validation and Highlighting
- Code Naviation
- Code Formatting
- Code Snippets
- Terraform Cloud Integration

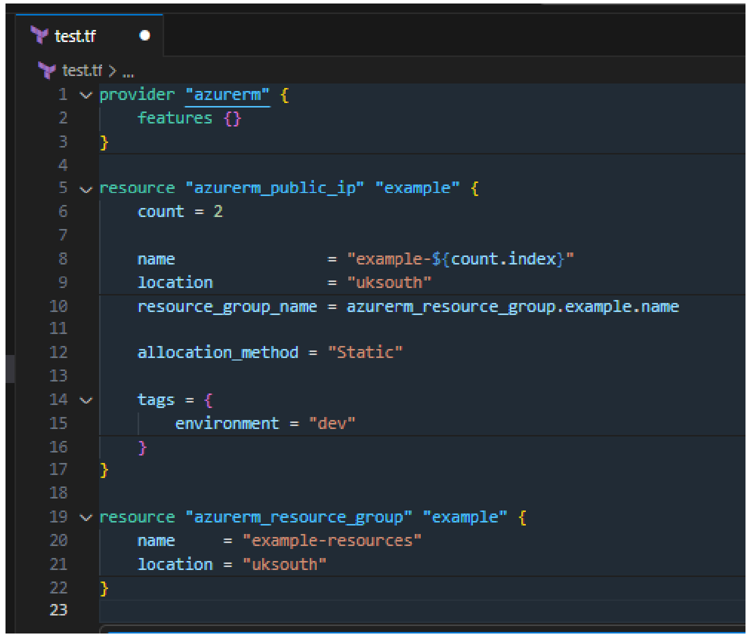
https://marketplace.visualstudio.com/items?itemName=HashiCorp.terraform

A helping hand… Try GitHub Copilot!



```
write the terraform to create two public IPs in the UK South Region and use the count function
```

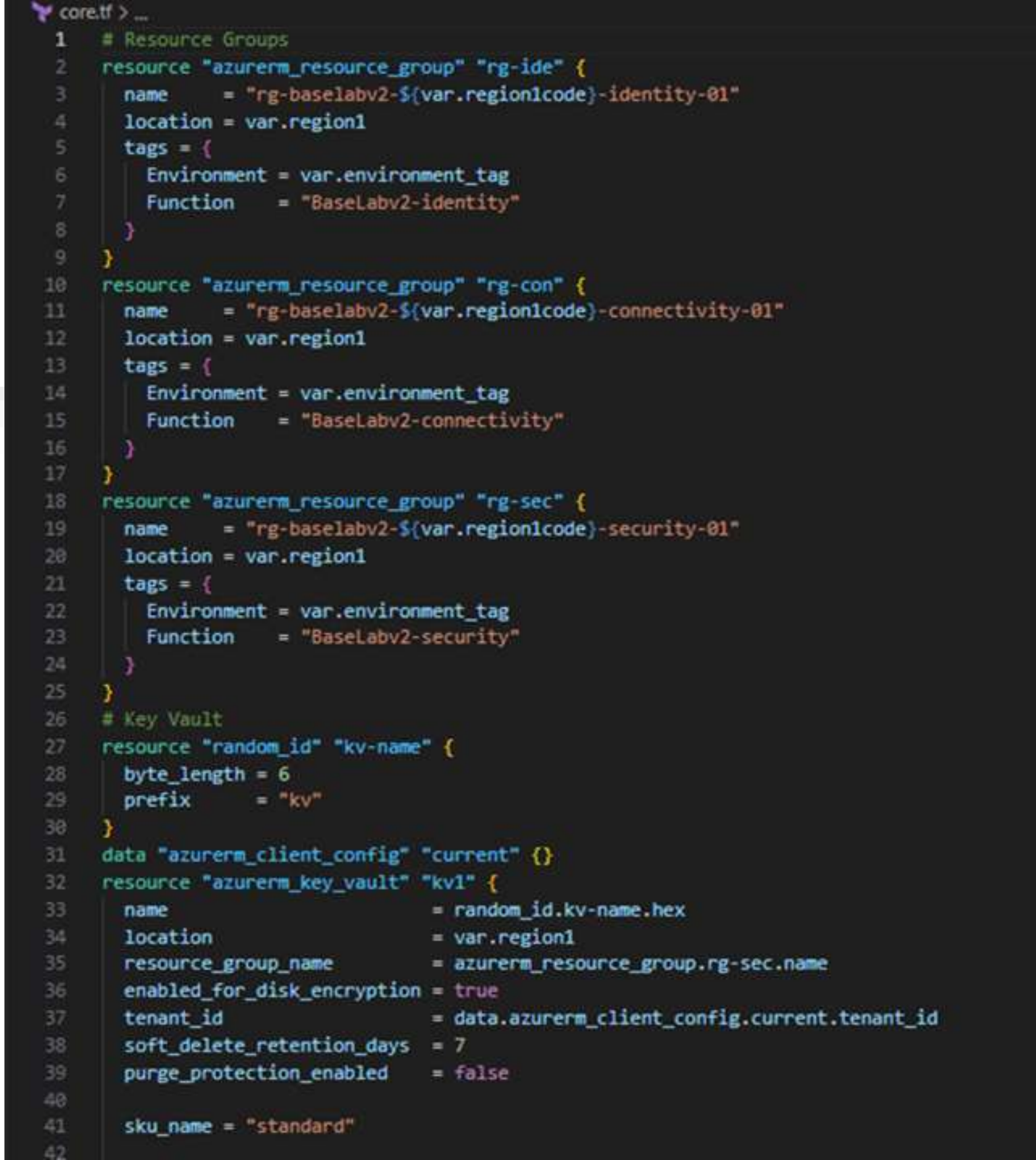
Copilot generated code may be incorrect



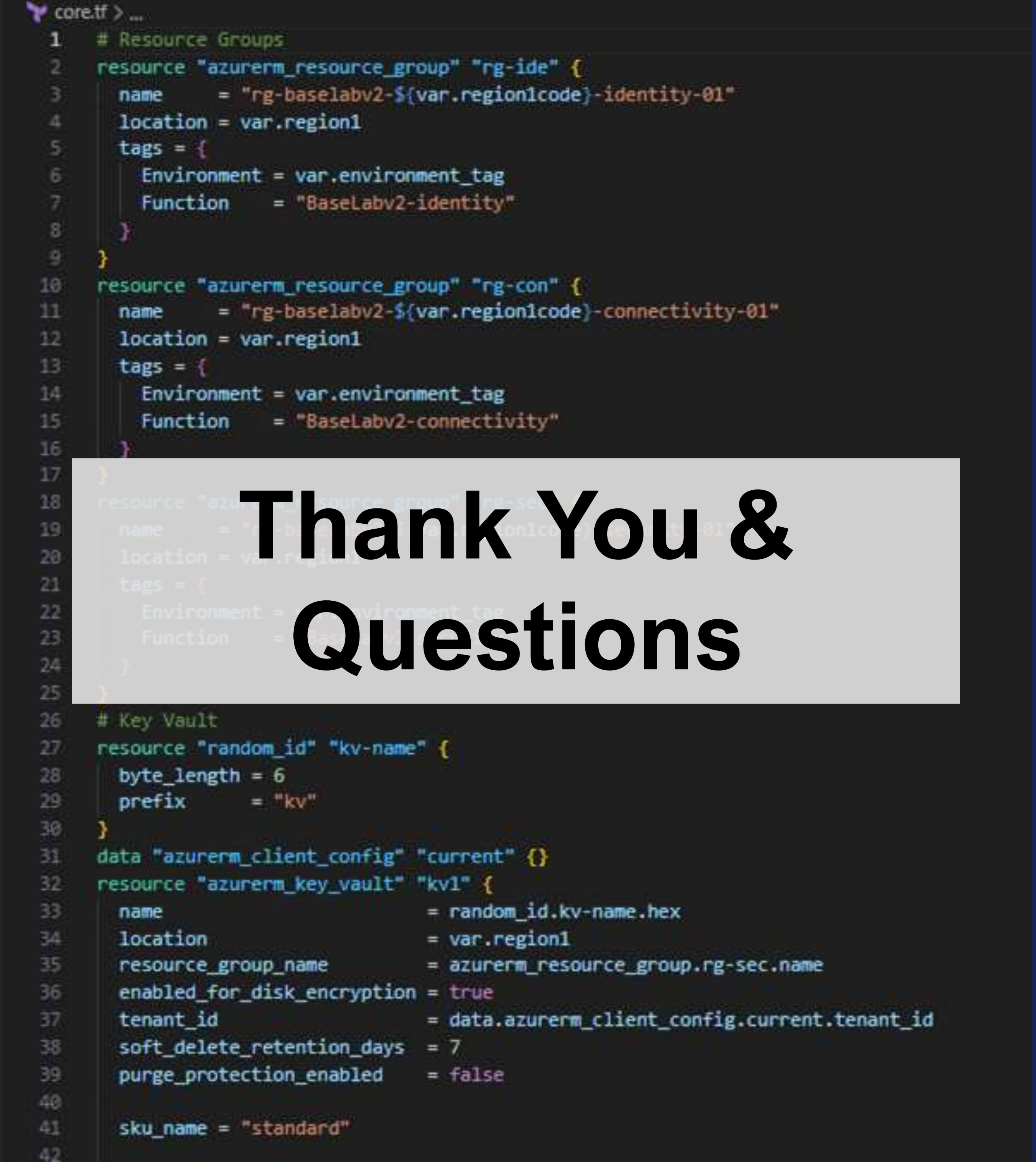
```
provider "azurerm" {
    features {}
}

resource "azurerm_public_ip" "example" {
    count = 2

    name                = "example-${count.index}"
    location            = "uksouth"
    resource_group_name = azurerm_resource_group.example.name

    allocation_method = "Static"

    tags = {
        environment = "dev"
    }
}

resource "azurerm_resource_group" "example" {
    name     = "example-resources"
    location = "uksouth"
}

```

Demo Time!

- Everything I am using today is available in my Terraform-Azure repo: https://github.com/jakewalsh90/Terraform-Azure
- All you need is Terraform, VSCode, Azure CLI, and an Azure Subscription.
- We will explore & demo the following:
  - Deployment of a Lab Environment
  - Run through code files (whilst it deploys – or the demo gods ruin my day).
  - Changes if time permits



```
# Resource Groups
resource "azurerm_resource_group" "rg-ide" {
  name     = "rg-baselabv2-${var.region1code}-identity-01"
  location = var.region1
  tags = {
    Environment = var.environment_tag
    Function    = "BaseLabv2-identity"
  }
}
resource "azurerm_resource_group" "rg-con" {
  name     = "rg-baselabv2-${var.region1code}-connectivity-01"
  location = var.region1
  tags = {
    Environment = var.environment_tag
    Function    = "BaseLabv2-connectivity"
  }
}
resource "azurerm_resource_group" "rg-sec" {
  name     = "rg-baselabv2-${var.region1code}-security-01"
  location = var.region1
  tags = {
    Environment = var.environment_tag
    Function    = "BaseLabv2-security"
  }
}
# Key Vault
resource "random_id" "kv-name" {
  byte_length = 6
  prefix      = "kv"
}
data "azurerm_client_config" "current" {}
resource "azurerm_key_vault" "kv1" {
  name                        = random_id.kv-name.hex
  location                    = var.region1
  resource_group_name         = azurerm_resource_group.rg-sec.name
  enabled_for_disk_encryption = true
  tenant_id                   = data.azurerm_client_config.current.tenant_id
  soft_delete_retention_days  = 7
  purge_protection_enabled    = false

  sku_name = "standard"
```



core.tf > ...

```
# Resource Groups
resource "azurerm_resource_group" "rg-ide" {
  name     = "rg-baselabv2-${var.region1code}-identity-01"
  location = var.region1
  tags = {
    Environment = var.environment_tag
    Function    = "BaseLabv2-identity"
  }
}
resource "azurerm_resource_group" "rg-con" {
  name     = "rg-baselabv2-${var.region1code}-connectivity-01"
  location = var.region1
  tags = {
    Environment = var.environment_tag
    Function    = "BaseLabv2-connectivity"
  }
}
resource "azurerm_resource_group" "rg-sec" {
  name     = "rg-base         on1code}     y-01"
  location = var.region1
  tags = {
    Environment =    environment_tag
    Function    = "Bas
  }
}
# Key Vault
resource "random_id" "kv-name" {
  byte_length = 6
  prefix      = "kv"
}
data "azurerm_client_config" "current" {}
resource "azurerm_key_vault" "kv1" {
  name                        = random_id.kv-name.hex
  location                    = var.region1
  resource_group_name         = azurerm_resource_group.rg-sec.name
  enabled_for_disk_encryption = true
  tenant_id                   = data.azurerm_client_config.current.tenant_id
  soft_delete_retention_days  = 7
  purge_protection_enabled    = false

  sku_name = "standard"
```

Thank You & Questions

EUC FORUM