Objective 3: Understand Terraform basics

▼ Handle Terraform and provider installation and versioning

- HashiCorp Terraform Tutorial
- This tutorial goes through the process of installing Terraform and provider installation and versioning

▼ Providers

- The primary construct of the Terraform language are `resources`, the `behaviors` of resources rely on the resource `types`, resource types are defined by `providers`.
- Providers have a set of resource types that defines which arguments are accepted, what attributes it exports, and how changes are applied to APIs.
- Providers require their own configuration for regions, authentication etc.
- **Configuration**
  - providers are configured with a provider block:

    ```
    provider "google" {
    project = "acme-app"
    region  = "us-central1"
    }
    ```

    ```
    #The google provider is assumed to be the provider for the resource type named
    ```

  - configuration arguments like project and region are evaluated in order
  - 2 meta-arguments available for provider blocks:
    - `version - to specify a version` and
    - `alias - to use same provider with different config for different resources`
  - provider blocks are not required if not explicitly configured Tf uses an empty default config when a resource from the provider is added
- **Initialization**
  - when a new provider is added to configuration Tf has to initialize the provider before it can be used

    - `terraform init` downloads and initializes any providers
    - only installs to current working directory, other directories can have other versions installed
- **Versions**
    - versions should be configured in production to avoid breaking changes
    - the `required_providers` block should be used in the Tf block:

      ```
      terraform {
         required_providers {
         aws = "~> 1.0"
              }
        }
      ```

    - When terraform init is re-run with providers already installed, it will use an already-installed provider that meets the constraints in preference to downloading a new version
    - to upgrade all modules run `terraform init -upgrade`
- **Multiple Provider Instances**
    - we can have multiple configs for the same provider by using the alias meta-argument to allow for multiple regions per provider, targeting multiple Docker hosts, etc.

      ```
      # The default provider configuration
        provider "aws" {
         region = "us-east-1"
        }

      # Additional provider configuration for west coast region
        provider "aws" {
            alias  = "west"
            region = "us-west-2"
        }
      ```

- **Third Party Plugins**
    - anyone can develop and distribute 3rd party Tf provers
    - need to be manually downloaded because they are not supported by terraform init
    - download must go in the user plugin directory - Windows: %APPDATA%\terraform.d\plugins | Others: ~/.terraform.d/plugins
- **Plugin Cache**

    - terraform init downloads plugins into a subdirectory of the working directory so each working dir is self contained. This means with more than one configuration with the same provider has a separate copy of the plugin for each config
    - plugins can be large so this isn't performant - Tf allows for a shared local directory for plugin cache. This has to be manually created in the CLI Configuration File.

      ```
      # (Note that the CLI configuration file is _not_ the same as the .tf files
      #  used to configure infrastructure.)

       plugin_cache_dir = "$HOME/.terraform.d/plugin-cache"
      ```

▼ Terraform Settings

- **Terraform Block Syntax**
    - only constant values can be used

      ```
       terraform {
       # ...
       }
      ```

- **Configuring a Terraform Backend**
    - this determines how state is stored, how operations are performed, remote back-ends for teams etc.

      ```
      terraform {
          backend "s3" {
          # (backend-specific settings...)
          }
      }
      ```

- **Specifying a Required Terraform Version**
- **Specifying Required Provider Versions**
- **Experimental Language Features**

▼ Describe plug-in based architecture

- Terraform is build on plug-in based architecture. Providers and provisioners used in configuration are plugins (AWS, Heroku). Anyone can create a new plugin. Build Infrastructure– Initialization

▼ Demonstrate using multiple providers

- Build Infrastructure– Providers

▼ Describe how Terraform finds and fetches providers

- Resource types are defined by providers
- Provider configuration is created with a provider block, the provider name is the name in the block header
- When a new provider is added Terraform has to initialize it before its used with the `terraform init` command. This downloads and installs the providers plugin

▼ Explain when to use and not use provisioners and when to use local-exec or remote-exec

- Provisioners - provisioners are used to model specific actions on the local machine or on a remote machine to prepare infrastructure objects
- Provisioners are there if needed but they add complexity and uncertainty (should only be used as a last result)
- Provisioners should be used if no other option will work.
- Use cases:
    - Passing data into virtual machines and other compute resources
    - running config management software
- local-exec - invokes a local executable after the resource is created. Invokes a process on the machine not on the resource.
- remote-exec - invokes a script on a remote resource after it is created.

⏮Objective 1 & 2 || Objective 4 ⏩

🔙README