TASKS ON TERRAFORM

- **NAME: S.MUZZAMMIL HUSSAIN**
- **DATE: 19/07/2025**
- **BATCH: 11**
- **NO.OF TASKS: 1**

Task 1. Explore these providers → local, random,azure and null

Exploring Terraform Providers: Local, Random, Azure, and Null

Below is an overview of each requested provider, including their purpose, key resources, and common use cases.

1. Local Provider

Purpose:
Manages files and resources on the local machine where Terraform is executed.

Key Resource:

- local_file: Create, update, or delete files on the local filesystem.

Example Usage

```
resource "local_file" "example" {
  content  = "Hello from Terraform!"
  filename = "${path.module}/example.txt"
}
```

Use Cases

- Writing configuration or output files as part of a deployment.
- Storing secrets or configurations locally for test or development setups.

Note: Primarily useful in automation, demos, or CI tasks where manipulating local files is necessary[1][2].

2. Random Provider

Purpose:
Generates random values for use within your Terraform configurations to ensure uniqueness or unpredictability.

Key Resources:

- random_id: Generates a random identifier.
- random_integer: Produces a random integer within a specified range.
- random_string: Creates a random string of characters.
- random_password: Generates complex passwords for use in secrets.
- random_uuid: Generates a random UUID.

- random_shuffle and random_pet: Additional options for unique outputs.

Example Usage

```
resource "random_password" "pw" {
  length  = 16
  special = true
}
```

Use Cases

- Creating unique names or IDs for resources.
- Generating secure, random passwords for cloud resources or services.
- Randomizing test data or instance selections.

Note: Values are stored in state and only change if the resource arguments change or you force regeneration[3][4][5].

3. Azure Provider (AzureRM)

Purpose:
Manages all Microsoft Azure cloud resources (also often written as the azurerm provider).

Key Features:

- Provision and manage a wide variety of Azure resources: virtual machines, storage accounts, networks, databases, Kubernetes clusters, and more.
- Support for resource tagging, authentication via service principal, managed identity, and Azure CLI.

Example Usage

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

resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = "East US"
}
```

Use Cases

- Infrastructure-as-Code for production Azure environments.
- Automating deployment, scaling, and destruction of cloud services.
- Integrating Azure infrastructure with CI/CD.

Note: AzureRM is maintained by both Microsoft and HashiCorp and supports most Azure services. Often used with best practices for authentication/authorization[6][7][8][9].

4. Null Provider

Purpose:
Provides a “do nothing” resource and is mainly used for orchestration, workaround scripting, or along with provisioners.

Key Resource:

- null_resource: A resource that doesn't actually manage infrastructure but can trigger actions via provisioners or scripts.

Example Usage

```
resource "null_resource" "example" {
  provisioner "local-exec" {
    command = "echo Hello, World!"
  }

  triggers = {
    always_run = "${timestamp()}"
  }
}
```

Use Cases

- Running scripts or commands that have no actual infrastructure backing.
- Orchestrating workflows that require explicit triggers (via triggers argument).
- Workarounds for provider/resource limitations or bootstrapping environments.

Note: Use null resources cautiously, as they can complicate Terraform state and configuration clarity. They're intended for exceptional cases, not core infrastructure management[10111213].

Usage Comparison Table

Provider	Main Use	Key Resources/Features	Typical Scenarios
Local	Local file management	local_file	File generation, scripts, automation
Random	Random value generation	random_password, random_id, random_string	Unique names, passwords, test data
Azure	Cloud resource management	VM, storage, networking, etc.	Azure infrastructure as code
Null	Orchestration, scripting	null_resource	Script runner, hacky workflows

Each provider serves a distinct purpose. Choose based on what you need to automate or manage within your Terraform workflow.

Add and configure providers

To **add and configure providers** like local, random, azurerm (Azure), and null in your Terraform configuration, you need two key sections:
required_providers block—declares which providers, source, and version you want.
One **provider** block per provider—handles provider-specific settings1356.
Below is a clear example that adds all four providers:

```
terraform {
        required_providers {
            local = {
                source = "hashicorp/local"
                version = "<=2.5.2"
            }
            random = {
                source = "hashicorp/random"
            }
            azure = {
                source = "hashicorp/azurerm"
            }
            null= {
                source= “hashicorp/null"
                version= “3.2.4”

            }
        }
    }
```

We create a file named providers.tf using command

vi providers.tf

Enter the content above to add the providers

Enter the contents as follows

```
terraform {
                required_providers {
                        local = {
                                source = "hashicorp/local"
                                version = "⩽2.5.2"
                        }
                        random = {
                                source = "hashicorp/random"
                        }
                        azure = {
                                source = "hashicorp/azurerm"
                        }
                        null = {
                                source = "hashicorp/null"
                                version = "3.2.4"
                        }
                }
        }

~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
                                                                    15,57         All
```

Initialize terraform using **terraform init**

```
mujju@VMterra:~/b11/1907$ vi providers.tf
mujju@VMterra:~/b11/1907$ terraform init
Initializing the backend...
Initializing provider plugins...
- Finding latest version of hashicorp/azurerm...
- Finding hashicorp/null versions matching "3.2.4"...
- Finding hashicorp/local versions matching "⩽ 2.5.2"...
- Finding latest version of hashicorp/random...
- Installing hashicorp/azurerm v4.37.0...
- Installed hashicorp/azurerm v4.37.0 (signed by HashiCorp)
- Installing hashicorp/null v3.2.4...
- Installed hashicorp/null v3.2.4 (signed by HashiCorp)
- Installing hashicorp/local v2.5.2...
- Installed hashicorp/local v2.5.2 (signed by HashiCorp)
- Installing hashicorp/random v3.7.2...
- Installed hashicorp/random v3.7.2 (signed by HashiCorp)
Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
mujju@VMterra:~/b11/1907$
```

Check the contents using **ls**

```
mujju@VMterra:~/b11/1907$ ls
providers.tf
mujju@VMterra:~/b11/1907$ cat *
terraform {
                required_providers {
                        local = {
                                source = "hashicorp/local"
                                version = "⩽2.5.2"
                        }
                        random = {
                                source = "hashicorp/random"
                        }
                        azure = {
                                source = "hashicorp/azurerm"
                        }
                        null = {
                                source = "hashicorp/null"
                                version = "3.2.4"
                        }
                }
        }


mujju@VMterra:~/b11/1907$ terraform fmt
providers.tf
mujju@VMterra:~/b11/1907$ cat *
terraform {
  required_providers {
    local = {
      source  = "hashicorp/local"
      version = "⩽2.5.2"
    }
    random = {
      source = "hashicorp/random"
    }
    azure = {
      source = "hashicorp/azurerm"
    }
    null = {
      source  = "hashicorp/null"
      version = "3.2.4"
    }
  }
```

Run the command **tree -a**