Terraform - Hands-On - Practice Assessment 1

Part 1 - Multiple Choice Questions: (Highlight the correct answer in bold)

1. What language is Terraform using?

   b. HCL

2. Terraform can be run on which of the following operating systems?

   d. All of the above

3. Is Terraform available as a single executable binary?

   a. Yes

4. What file extension is used for Terraform configuration file?

   a. .tf

5. Which of the following is NOT a text editor for creating Terraform files?

   d. Microsoft Word

6. Which of these is NOT a Terraform command?

   b. Compile

7. Which command is used to initialize a working directory containing Terraform configuration files?

   a. terraform init

8. Before running terraform apply, which command should be executed to see the planned actions?

   b. terraform plan

9. The command to find Terraform's version is**:**

   c. terraform --v

10. What is the purpose of the terraform show command?

    a. To display the current state or saved plan

11. Which of the following is a valid Terraform resource type?

    d. All of the above

12. What is the terraform destroy command used for?

    a. To remove all previously created infrastructure

13. What is Terraform mainly used for?

    b. Infrastructure as Code

14. Which file is used by Terraform to track the current state of the infrastructure?

    a. terraform.tfstate

Part 2 – Hands-On Labs

Lab 1: Setting Up a Terraform Project in Visual Studio Code

- **Install Visual Studio Code**
  - If you do not already have Visual Studio Code, download and install it from the official website.
- **Install Terraform Extension in VS Code**
  - Open Visual Studio Code.
  - Go to Extensions
  - Search for "Terraform" and install the extension by HashiCorp.
- **Create a New Project Folder**
  - Create a new folder on your computer where you will store your Terraform files.
  - Open this folder in Visual Studio Code (File > Open Folder).
- **Initialize a New Terraform Configuration File**
  - Create a new file in the folder with the **.tf** extension, for example, **main.tf**.
  - Write a simple Terraform configuration or leave it blank for now.

Lab 2: Basic Local File Operation

- **Define a Local File Resource**
  - In **main.tf**, start by defining a resource to create a local file. For example:

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

- **Initialize Terraform**
  - Open the terminal in VS Code (Terminal > New Terminal).
  - Run **terraform init** to initialize the Terraform project. This command sets up Terraform to run your configuration.
- **Apply Configuration**
  - Run **terraform apply** to apply your configuration.
  - Confirm the action in the terminal when prompted.
  - This step will create a file named **example.txt** with the content "Hello, Terraform!" in your project directory.

Lab 3: Handling Sensitive File Operations

- **Create a Sensitive File Resource**
  - Now, let us handle a sensitive file operation. For example, you might want to create a file that contains sensitive information.
  - In **main.tf**, add a new resource block:

```
resource "local_file" "sensitive_file" {
  filename = "${path.module}/sensitive.txt"
  content  = var.sensitive_content

}
```

- **Define Variables**
  - Create a new file named **variables.tf** and define a variable for the sensitive content:

```
variable "sensitive_content" {
  description = "Sensitive content for the file"
  type        = string
  sensitive   = true
}
```

- **Add Sensitive Content**
  - Create a **terraform.tfvars** file to store the value of the sensitive content.
  - Add your sensitive content in **terraform.tfvars**, like:

sensitive_content = "Secret Information Here"

- **Re-run Terraform Apply**
  - Run **terraform apply** again in your terminal.
  - Confirm the action when prompted.
  - Terraform will now create another file named **sensitive.txt** with the sensitive content, and it will treat the content as sensitive in its output.
  - **Verify the Files**
  - Check your project directory. You should see two new files: **example.txt** and **sensitive.txt**, each with the specified content.

Conclusion and Best Practices

- Always use version control (like Git) to manage your Terraform files.
- Avoid committing sensitive data, like **terraform.tfvars** with sensitive default values, to version control.
- Add .gitignore file to the terraform project
- Commit the files to your Kanini Private git repository with proper commit message.
- Regularly refer to Terraform documentation for best practices and updates.

Useful Links:

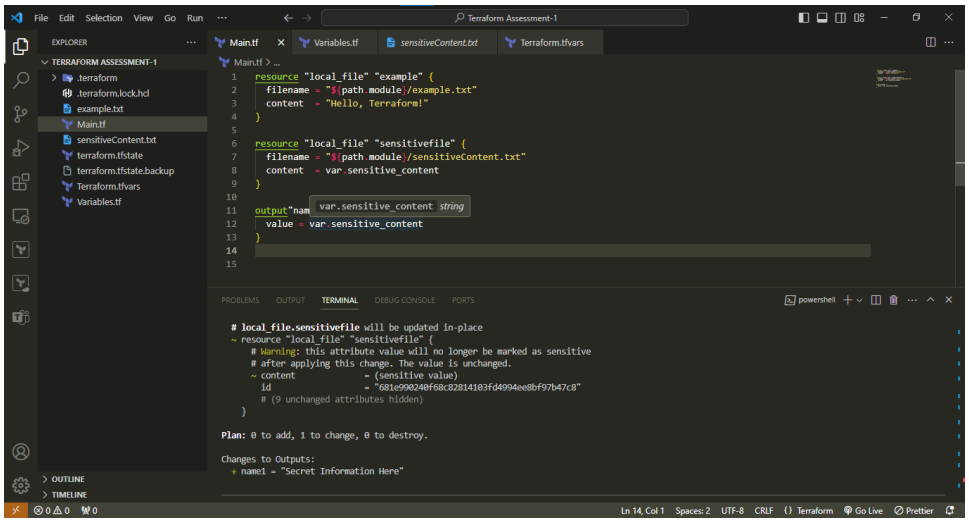
Local Provider: https://registry.terraform.io/providers/hashicorp/local/latest/docs

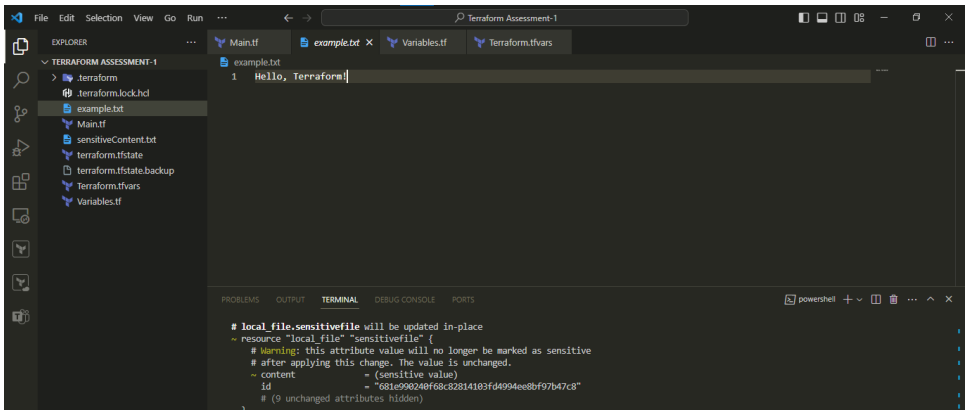
Providers: https://registry.terraform.io/browse/providers

Use Cases: https://developer.hashicorp.com/terraform/tutorials/applications

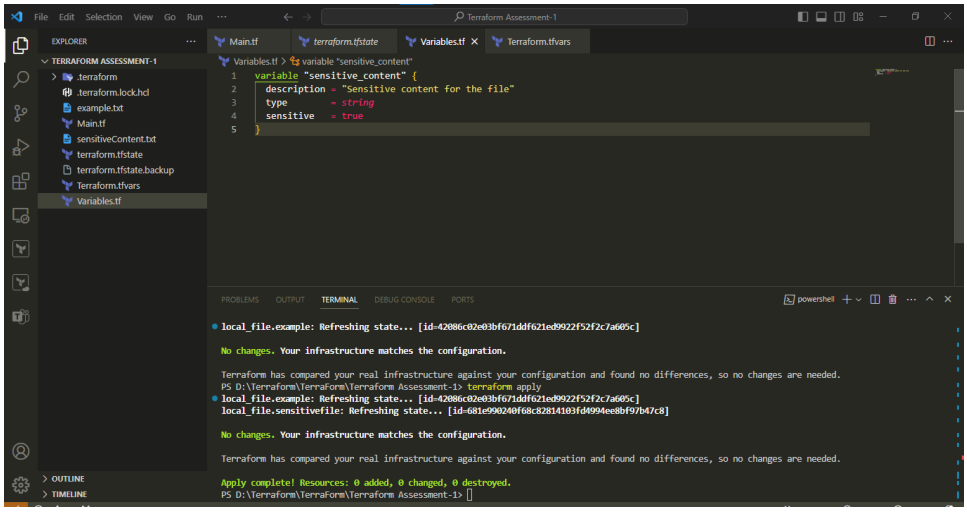
Outputs:

2.


Terraform Assessment-1
EXPLORER
TERRAFORM ASSESSMENT-1
.terraform
.terraform.lock.hcl
example.txt
Main.tf
sensitiveContent.txt
terraform.tfstate
terraform.tfstate.backup
Terraform.tfvars
Variables.tf
Main.tf
Variables.tf
sensitiveContent.txt
Terraform.tfvars
resource "local_file" "example" {
filename = "${path.module}/example.txt"
content = "Hello, Terraform!"
}
resource "local_file" "sensitivefile" {
filename = "${path.module}/sensitiveContent.txt"
content = var.sensitive_content
}
output"nam
var.sensitive_content string
value = var.sensitive_content
}
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
PORTS
# local_file.sensitivefile will be updated in-place
~ resource "local_file" "sensitivefile" {
# Warning: this attribute value will no longer be marked as sensitive
# after applying this change. The value is unchanged.
~ content = (sensitive value)
id = "681e990240f68c82814103fd4994ee8bf97b47c8"
# (9 unchanged attributes hidden)
}
Plan: 0 to add, 1 to change, 0 to destroy.
Changes to Outputs:
+ name1 = "Secret Information Here"
OUTLINE
TIMELINE


Terraform Assessment-1
EXPLORER
TERRAFORM ASSESSMENT-1
.terraform
.terraform.lock.hcl
example.txt
Main.tf
sensitiveContent.txt
terraform.tfstate
terraform.tfstate.backup
Terraform.tfvars
Variables.tf
Main.tf
example.txt
Variables.tf
Terraform.tfvars
Hello, Terraform!
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
PORTS
# local_file.sensitivefile will be updated in-place
~ resource "local_file" "sensitivefile" {
# Warning: this attribute value will no longer be marked as sensitive
# after applying this change. The value is unchanged.
~ content = (sensitive value)
id = "681e990240f68c82814103fd4994ee8bf97b47c8"
# (9 unchanged attributes hidden)

3.


Terraform Assessment-1
EXPLORER
TERRAFORM ASSESSMENT-1
.terraform
.terraform.lock.hcl
example.txt
Main.tf
sensitiveContent.txt
terraform.tfstate
terraform.tfstate.backup
Terraform.tfvars
Variables.tf
Main.tf
terraform.tfstate
Variables.tf
Terraform.tfvars
variable "sensitive_content" {
description = "Sensitive content for the file"
type = string
sensitive = true
}
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE
PORTS
local_file.example: Refreshing state... [id=42086c02e03bf671ddf621ed9922f52f2c7a605c]
No changes. Your infrastructure matches the configuration.
Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.
PS D:\Terraform\TerraForm\Terraform Assessment-1> terraform apply
local_file.example: Refreshing state... [id=42086c02e03bf671ddf621ed9922f52f2c7a605c]
local_file.sensitivefile: Refreshing state... [id=681e990240f68c82814103fd4994ee8bf97b47c8]
No changes. Your infrastructure matches the configuration.
Terraform has compared your real infrastructure against your configuration and found no differences, so no changes are needed.
Apply complete! Resources: 0 added, 0 changed, 0 destroyed.
PS D:\Terraform\TerraForm\Terraform Assessment-1>
OUTLINE
TIMELINE