Terraform Lab User Guide

This guide summarizes the key concepts and practical exercises covered in our Terraform lab session. It serves as a reference to help you understand Terraform's core features, including resource management, attribute referencing, and dependency management.

---

1. Understanding Variable Definition Precedence

Terraform has a clear order of precedence for defining variable values. This ensures that you can override values as needed, with the most explicit method taking the highest priority.

Order of Precedence (Highest to Lowest)

1. `-var` **command-line flags:** `terraform apply -var="key=value"`
2. `-var-file` **command-line flags:** `terraform apply -var-file="testing.tfvars"`
3. `.auto.tfvars` **files:** `production.auto.tfvars`
4. `terraform.tfvars` **and** `terraform.tfvars.json` **files:** These are automatically loaded
5. **Environment Variables:** `TF_VAR_key=value`
6. `default` **values in variable blocks:** `variable "key" { default = "value" }`

---

2. Using the `time_static` Resource

The `time_static` resource from the `hashicorp/time` provider is a logical resource that generates a timestamp. It is useful for creating unique identifiers or tracking creation times.

Lab Exercise

We used `time_static` to create a timestamp and exported its `id` attribute.

Exported Attributes

- `id`: The full RFC3339 timestamp
- `rfc3339`: Same as `id`
- `unix`: Unix timestamp (seconds since epoch)
- `year`, `month`, `day`, `hour`, `minute`, `second`: Numerical parts of the timestamp

---

3. Referencing Attributes with `local_file`

The `local_file` resource is used to create or manage a file on the local machine. We used a **reference expression** to link the content of this file to an attribute from another resource.

Lab Exercise

We created a `local_file` resource named `time` that depended on a `time_static` resource named `update`.

`main.tf` Example

```
terraform

resource "time_static" "update" {}

resource "local_file" "time" {
  filename = "/root/time.txt"
  content  = "Time stamp of this file is ${time_static.update.id}"
}
```

Key Takeaway

The reference expression `time_static.update.id` creates an **explicit dependency**. Terraform understands that `local_file.time` cannot be created until `time_static.update` is fully provisioned.

---

4. Inspecting State with `terraform show`

The `terraform show` command is crucial for inspecting the current state of your infrastructure as recorded in the `terraform.tfstate` file. It displays the attributes of all resources managed by Terraform.

Lab Exercise

We used `terraform show` to find the `id` of the `local_file` resource. This `id` is not a fixed identifier but rather a SHA1 hash of the file's content.

Command

```
bash

terraform show
```

Key Takeaway

The `id` of a `local_file` resource changes whenever its content changes, which is how Terraform tracks modifications.

---

5. Understanding Terraform Dependencies

Dependencies are the order in which resources are created and managed.

- **Explicit Dependency**: Created when one resource's configuration references an attribute of another resource. This is Terraform's default and most common form of dependency management.
- **Implicit Dependency**: Occurs when there is a logical order requirement between resources, but no direct attribute reference exists in the configuration. Terraform cannot detect these automatically.

---

6. The `tls_private_key` Resource

This is a **logical resource** from the `hashicorp/tls` provider. It does not create a cloud resource but generates a secure private key and stores it in the state file.

Lab Exercise

We created a `tls_private_key` resource named `pvtkey` and configured it to use the RSA algorithm with 4096 bits.

`key.tf` Example

```
terraform

resource "tls_private_key" "pvtkey" {
  algorithm = "RSA"
  rsa_bits  = 4096
}
```

---

7. Explicit Dependency in Action

We demonstrated an explicit dependency by creating a `local_file` to store the output of the `tls_private_key` resource.

Lab Exercise

We updated `key.tf` to create a new `local_file` resource that uses the `private_key_pem` attribute from the generated key.

Updated `key.tf` Example

```
terraform
```

```
resource "tls_private_key" "pvtkey" {
  algorithm = "RSA"
  rsa_bits  = 4096
}

resource "local_file" "key_details" {
  filename = "/root/key.txt"
  content  = tls_private_key.pvtkey.private_key_pem
}
```

---

8. Managing Implicit Dependencies with `depends_on`

To handle implicit dependencies, you must use the `depends_on` argument. This tells Terraform to explicitly wait for one or more resources to be created before provisioning the current resource.

Lab Exercise

We created two `local_file` resources, `krill` and `whale`. We used `depends_on` on the `whale` resource to ensure it is created only after the `krill` resource is complete.

`main.tf` Example

```
terraform

resource "local_file" "krill" {
  filename = "/root/krill"
  content  = "krill"
}

resource "local_file" "whale" {
  filename = "/root/whale"
  content  = "whale"

  depends_on = [
    local_file.krill,
  ]
}
```

This ensures the files are created in the correct order, even without an attribute reference.

Summary

This lab covered essential Terraform concepts:

1. **Variable precedence** - Understanding how Terraform resolves variable values
2. **Resource attributes** - Using `time_static` and `local_file` resources
3. **Reference expressions** - Creating explicit dependencies between resources
4. **State inspection** - Using `terraform show` to examine resource state
5. **Dependency management** - Both explicit and implicit dependencies
6. **Logical resources** - Working with `tls_private_key` for cryptographic operations
7. **Explicit dependency control** - Using `depends_on` for complex dependency scenarios

These concepts form the foundation for building more complex Terraform configurations and managing infrastructure dependencies effectively.