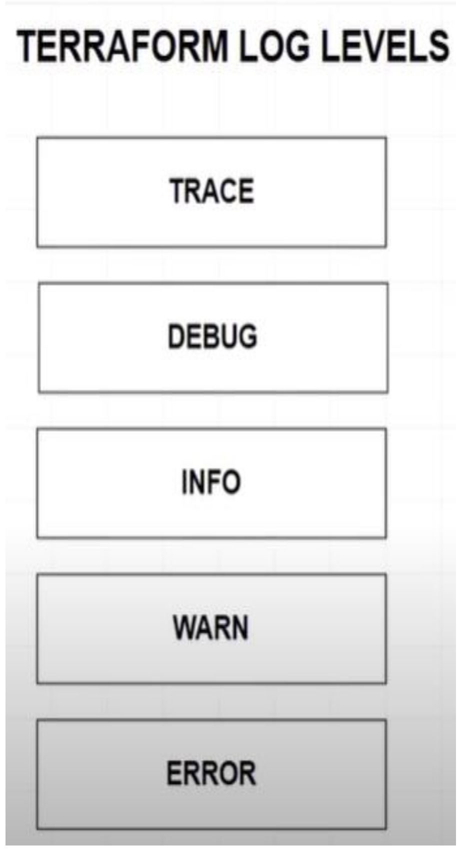
Debugging Terraform

Overview of Debugging Terraform

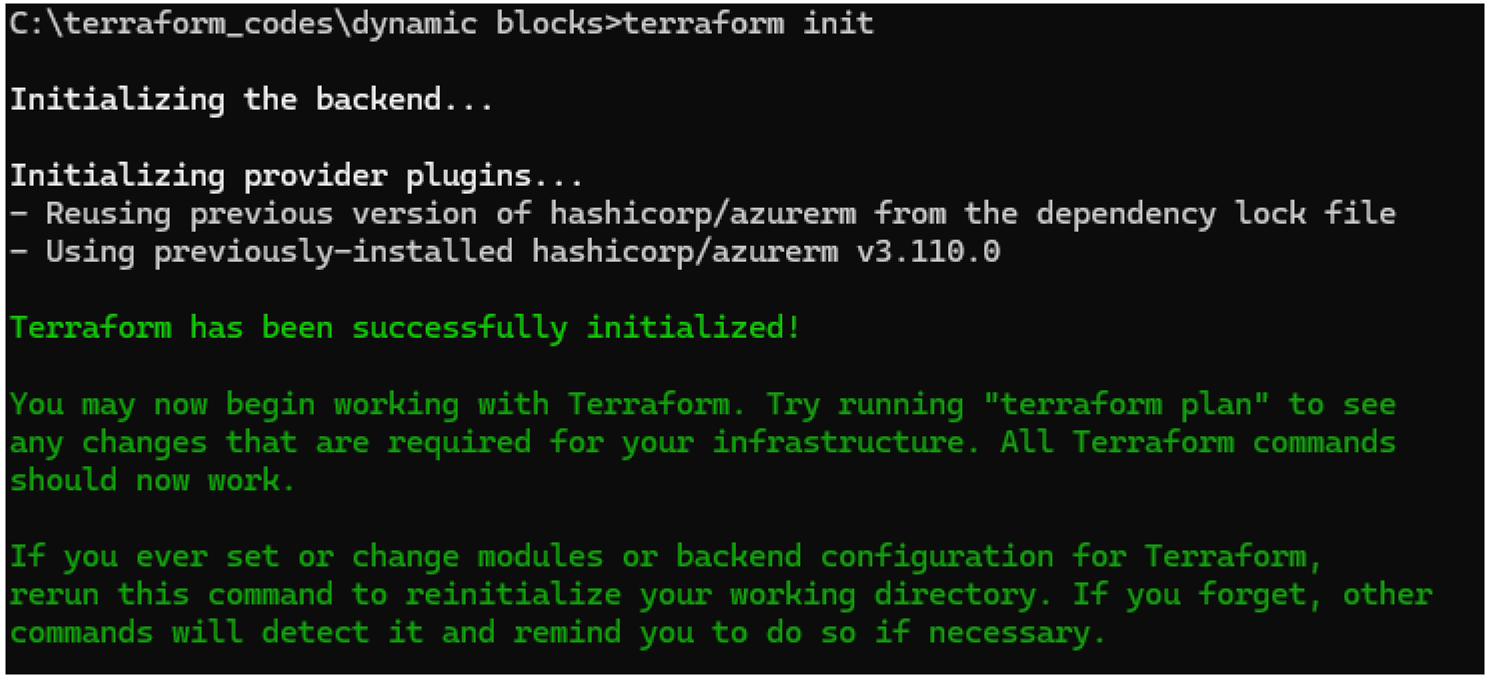
- Debugging is the process of finding the root cause of a specific issue.

• Terraform has detailed logs which can be enabled by setting the TF_LOG environment variable to any value.

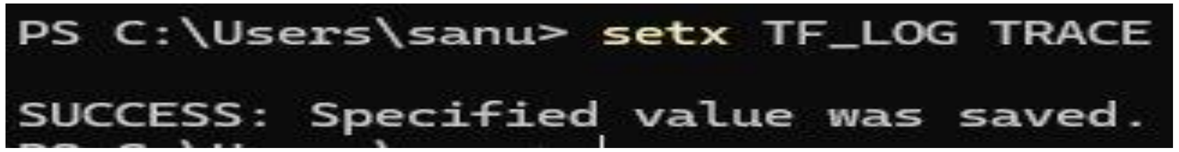
• You can set TF_LOG to one of the log levels TRACE, DEBUG, INFO, WARN, OR ERROR to change the verbosity of the logs.



Terraform Init

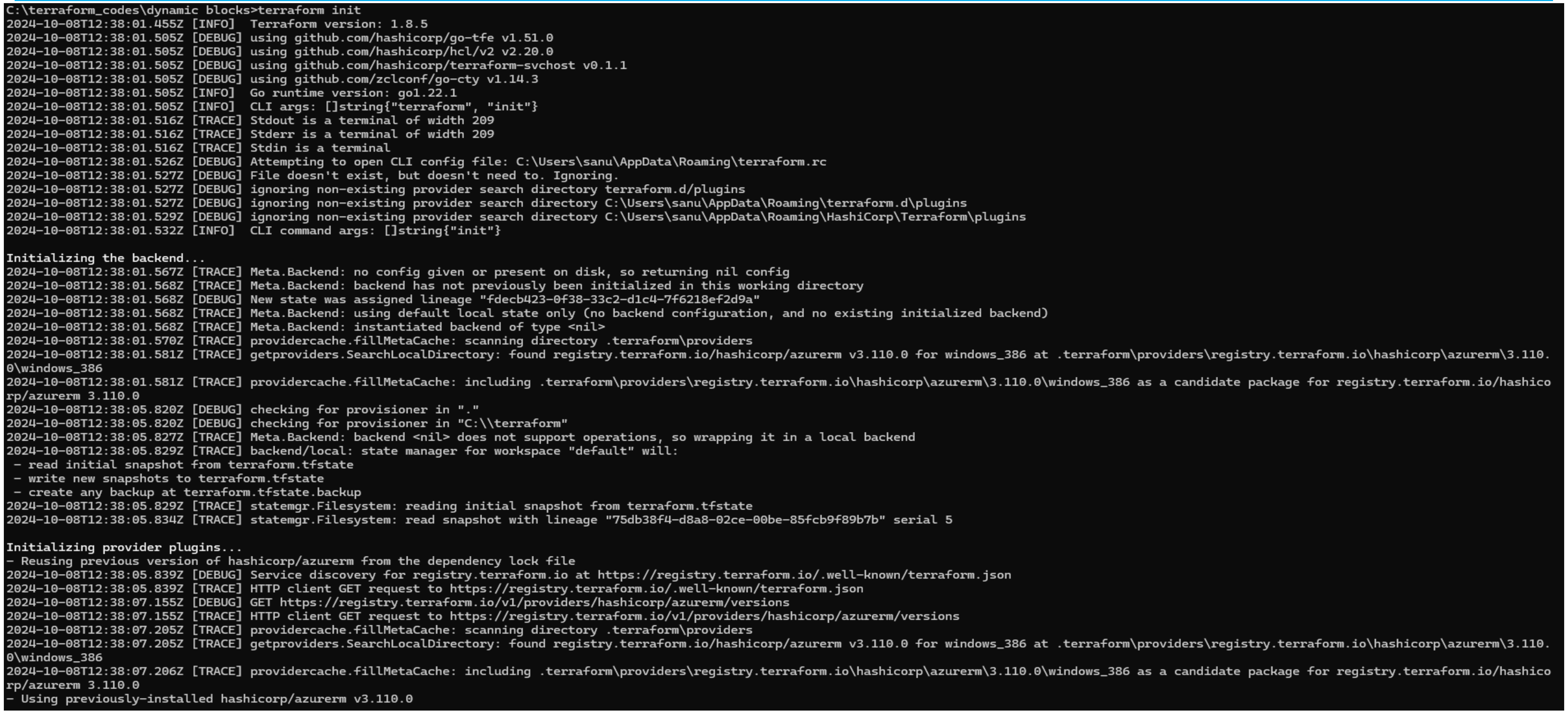


```
C:\terraform_codes\dynamic blocks>terraform init

Initializing the backend...

Initializing provider plugins...
- Reusing previous version of hashicorp/azurerm from the dependency lock file
- Using previously-installed hashicorp/azurerm v3.110.0

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.
```



```
PS C:\Users\sanu> setx TF_LOG TRACE

SUCCESS: Specified value was saved.
```

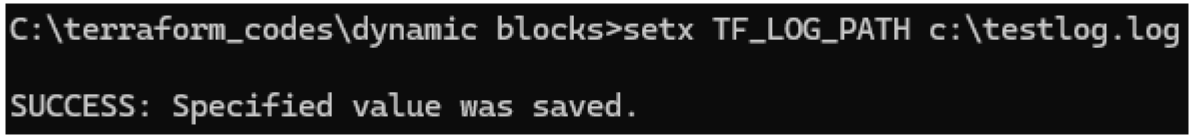
Terraform Init with Trace



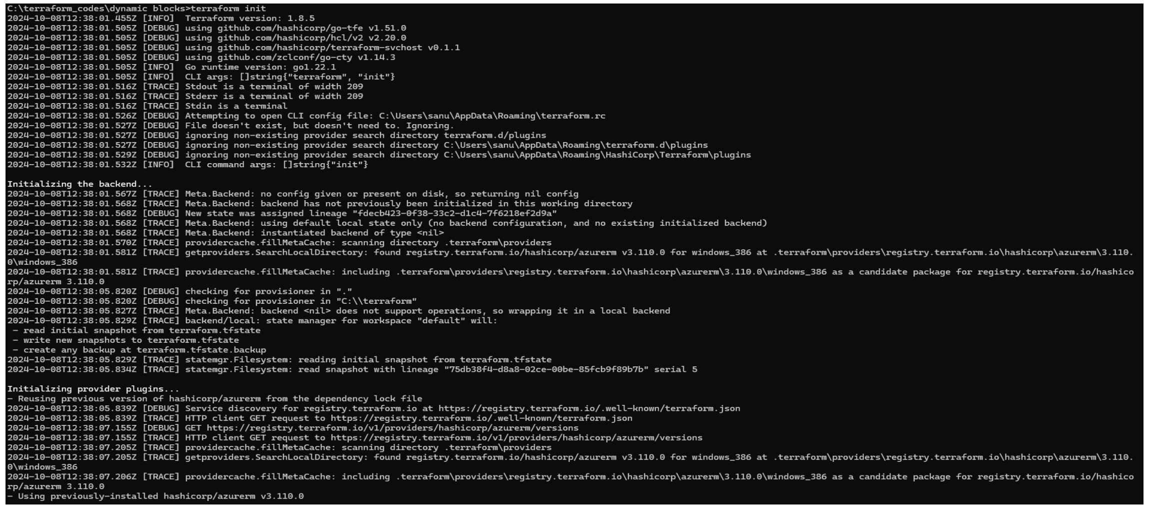
```
C:\terraform_codes\dynamic blocks>terraform init
2024-10-08T12:38:01.455Z [INFO]  Terraform version: 1.8.5
2024-10-08T12:38:01.505Z [DEBUG] using github.com/hashicorp/go-tfe v1.51.0
2024-10-08T12:38:01.505Z [DEBUG] using github.com/hashicorp/hcl/v2 v2.20.0
2024-10-08T12:38:01.505Z [DEBUG] using github.com/hashicorp/terraform-svchost v0.1.1
2024-10-08T12:38:01.505Z [DEBUG] using github.com/zclconf/go-cty v1.14.3
2024-10-08T12:38:01.505Z [INFO]  Go runtime version: go1.22.1
2024-10-08T12:38:01.505Z [INFO]  CLI args: []string{"terraform", "init"}
2024-10-08T12:38:01.516Z [TRACE] Stdout is a terminal of width 209
2024-10-08T12:38:01.516Z [TRACE] Stderr is a terminal of width 209
2024-10-08T12:38:01.516Z [TRACE] Stdin is a terminal
2024-10-08T12:38:01.526Z [DEBUG] Attempting to open CLI config file: C:\Users\sanu\AppData\Roaming\terraform.rc
2024-10-08T12:38:01.527Z [DEBUG] File doesn't exist, but doesn't need to. Ignoring.
2024-10-08T12:38:01.527Z [DEBUG] ignoring non-existing provider search directory terraform.d/plugins
2024-10-08T12:38:01.527Z [DEBUG] ignoring non-existing provider search directory C:\Users\sanu\AppData\Roaming\terraform.d\plugins
2024-10-08T12:38:01.529Z [DEBUG] ignoring non-existing provider search directory C:\Users\sanu\AppData\Roaming\HashiCorp\Terraform\plugins
2024-10-08T12:38:01.532Z [INFO]  CLI command args: []string{"init"}

Initializing the backend...
2024-10-08T12:38:01.567Z [TRACE] Meta.Backend: no config given or present on disk, so returning nil config
2024-10-08T12:38:01.568Z [TRACE] Meta.Backend: backend has not previously been initialized in this working directory
2024-10-08T12:38:01.568Z [DEBUG] New state was assigned lineage "fdecb423-0f38-33c2-d1c4-7f6218ef2d9a"
2024-10-08T12:38:01.568Z [TRACE] Meta.Backend: using default local state only (no backend configuration, and no existing initialized backend)
2024-10-08T12:38:01.568Z [TRACE] Meta.Backend: instantiated backend of type <nil>
2024-10-08T12:38:01.570Z [TRACE] providercache.fillMetaCache: scanning directory .terraform\providers
2024-10-08T12:38:01.581Z [TRACE] getproviders.SearchLocalDirectory: found registry.terraform.io/hashicorp/azurerm v3.110.0 for windows_386 at .terraform\providers\registry.terraform.io\hashicorp\azurerm\3.110.
0\windows_386
2024-10-08T12:38:01.581Z [TRACE] providercache.fillMetaCache: including .terraform\providers\registry.terraform.io\hashicorp\azurerm\3.110.0\windows_386 as a candidate package for registry.terraform.io/hashico
rp/azurerm 3.110.0
2024-10-08T12:38:05.820Z [DEBUG] checking for provisioner in "."
2024-10-08T12:38:05.820Z [DEBUG] checking for provisioner in "C:\\terraform"
2024-10-08T12:38:05.827Z [TRACE] Meta.Backend: backend <nil> does not support operations, so wrapping it in a local backend
2024-10-08T12:38:05.829Z [TRACE] backend/local: state manager for workspace "default" will:
 - read initial snapshot from terraform.tfstate
 - write new snapshots to terraform.tfstate
 - create any backup at terraform.tfstate.backup
2024-10-08T12:38:05.829Z [TRACE] statemgr.Filesystem: reading initial snapshot from terraform.tfstate
2024-10-08T12:38:05.834Z [TRACE] statemgr.Filesystem: read snapshot with lineage "75db38f4-d8a8-02ce-00be-85fcb9f89b7b" serial 5

Initializing provider plugins...
- Reusing previous version of hashicorp/azurerm from the dependency lock file
2024-10-08T12:38:05.839Z [DEBUG] Service discovery for registry.terraform.io at https://registry.terraform.io/.well-known/terraform.json
2024-10-08T12:38:05.839Z [TRACE] HTTP client GET request to https://registry.terraform.io/.well-known/terraform.json
2024-10-08T12:38:07.155Z [DEBUG] GET https://registry.terraform.io/v1/providers/hashicorp/azurerm/versions
2024-10-08T12:38:07.155Z [TRACE] HTTP client GET request to https://registry.terraform.io/v1/providers/hashicorp/azurerm/versions
2024-10-08T12:38:07.205Z [TRACE] providercache.fillMetaCache: scanning directory .terraform\providers
2024-10-08T12:38:07.205Z [TRACE] getproviders.SearchLocalDirectory: found registry.terraform.io/hashicorp/azurerm v3.110.0 for windows_386 at .terraform\providers\registry.terraform.io\hashicorp\azurerm\3.110.
0\windows_386
2024-10-08T12:38:07.206Z [TRACE] providercache.fillMetaCache: including .terraform\providers\registry.terraform.io\hashicorp\azurerm\3.110.0\windows_386 as a candidate package for registry.terraform.io/hashico
rp/azurerm 3.110.0
- Using previously-installed hashicorp/azurerm v3.110.0
```

Storing the Logs to File

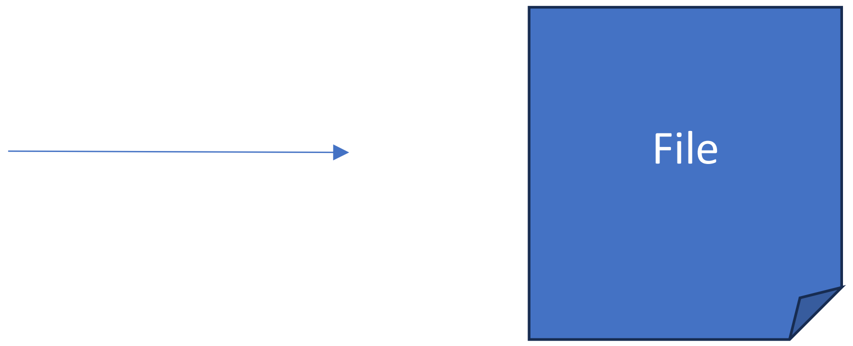
To persist logged output you can set **TF_LOG_PATH** in order to force the log to always be appended to a specific file when logging is enabled.



```
C:\terraform_codes\dynamic blocks>setx TF_LOG_PATH c:\testlog.log

SUCCESS: Specified value was saved.
```



```
C:\terraform_codes\dynamic blocks>terraform init
2024-10-08T12:38:01.455Z [INFO]  Terraform version: 1.8.5
2024-10-08T12:38:01.505Z [DEBUG] using github.com/hashicorp/go-tfe v1.51.0
2024-10-08T12:38:01.505Z [DEBUG] using github.com/hashicorp/hcl/v2 v2.20.0
2024-10-08T12:38:01.505Z [DEBUG] using github.com/hashicorp/terraform-svchost v0.1.1
2024-10-08T12:38:01.505Z [DEBUG] using github.com/zclconf/go-cty v1.14.3
2024-10-08T12:38:01.505Z [INFO]  Go runtime version: go1.22.1
2024-10-08T12:38:01.505Z [INFO]  CLI args: []string{"terraform", "init"}
2024-10-08T12:38:01.516Z [TRACE] Stdout is a terminal of width 209
2024-10-08T12:38:01.516Z [TRACE] Stderr is a terminal of width 209
2024-10-08T12:38:01.516Z [TRACE] Stdin is a terminal
2024-10-08T12:38:01.526Z [DEBUG] Attempting to open CLI config file: C:\Users\sanu\AppData\Roaming\terraform.rc
2024-10-08T12:38:01.527Z [DEBUG] File doesn't exist, but doesn't need to. Ignoring.
2024-10-08T12:38:01.527Z [DEBUG] ignoring non-existing provider search directory terraform.d/plugins
2024-10-08T12:38:01.527Z [DEBUG] ignoring non-existing provider search directory C:\Users\sanu\AppData\Roaming\terraform.d\plugins
2024-10-08T12:38:01.529Z [DEBUG] ignoring non-existing provider search directory C:\Users\sanu\AppData\Roaming\HashiCorp\Terraform\plugins
2024-10-08T12:38:01.532Z [INFO]  CLI command args: []string{"init"}

Initializing the backend...
2024-10-08T12:38:01.567Z [TRACE] Meta.Backend: no config given or present on disk, so returning nil config
2024-10-08T12:38:01.568Z [TRACE] Meta.Backend: backend has not previously been initialized in this working directory
2024-10-08T12:38:01.568Z [DEBUG] New state was assigned lineage "fdecb423-0f38-33c2-d1c4-7f6218ef2d9a"
2024-10-08T12:38:01.568Z [TRACE] Meta.Backend: using default local state only (no backend configuration, and no existing initialized backend)
2024-10-08T12:38:01.568Z [TRACE] Meta.Backend: instantiated backend of type <nil>
2024-10-08T12:38:01.570Z [TRACE] providercache.fillMetaCache: scanning directory .terraform\providers
2024-10-08T12:38:01.581Z [TRACE] getproviders.SearchLocalDirectory: found registry.terraform.io/hashicorp/azurerm v3.110.0 for windows_386 at .terraform\providers\registry.terraform.io\hashicorp\azurerm\3.110.
0\windows_386
2024-10-08T12:38:01.581Z [TRACE] providercache.fillMetaCache: including .terraform\providers\registry.terraform.io\hashicorp\azurerm\3.110.0\windows_386 as a candidate package for registry.terraform.io/hashico
rp/azurerm 3.110.0
2024-10-08T12:38:05.820Z [DEBUG] checking for provisioner in "."
2024-10-08T12:38:05.820Z [DEBUG] checking for provisioner in "C:\\terraform"
2024-10-08T12:38:05.827Z [TRACE] Meta.Backend: backend <nil> does not support operations, so wrapping it in a local backend
2024-10-08T12:38:05.829Z [TRACE] backend/local: state manager for workspace "default" will:
 - read initial snapshot from terraform.tfstate
 - write new snapshots to terraform.tfstate
 - create any backup at terraform.tfstate.backup
2024-10-08T12:38:05.829Z [TRACE] statemgr.Filesystem: reading initial snapshot from terraform.tfstate
2024-10-08T12:38:05.834Z [TRACE] statemgr.Filesystem: read snapshot with lineage "75db38f4-d8a8-02ce-00be-85fcb9f89b7b" serial 5

Initializing provider plugins...
- Reusing previous version of hashicorp/azurerm from the dependency lock file
2024-10-08T12:38:05.839Z [DEBUG] Service discovery for registry.terraform.io at https://registry.terraform.io/.well-known/terraform.json
2024-10-08T12:38:05.839Z [TRACE] HTTP client GET request to https://registry.terraform.io/.well-known/terraform.json
2024-10-08T12:38:07.155Z [DEBUG] GET https://registry.terraform.io/v1/providers/hashicorp/azurerm/versions
2024-10-08T12:38:07.155Z [TRACE] HTTP client GET request to https://registry.terraform.io/v1/providers/hashicorp/azurerm/versions
2024-10-08T12:38:07.205Z [TRACE] providercache.fillMetaCache: scanning directory .terraform\providers
2024-10-08T12:38:07.205Z [TRACE] getproviders.SearchLocalDirectory: found registry.terraform.io/hashicorp/azurerm v3.110.0 for windows_386 at .terraform\providers\registry.terraform.io\hashicorp\azurerm\3.110.
0\windows_386
2024-10-08T12:38:07.206Z [TRACE] providercache.fillMetaCache: including .terraform\providers\registry.terraform.io\hashicorp\azurerm\3.110.0\windows_386 as a candidate package for registry.terraform.io/hashico
rp/azurerm 3.110.0
- Using previously-installed hashicorp/azurerm v3.110.0
```



Join us in our Adventure

https://www.linkedin.com/in/akash-kumar-480b3858/

https://www.instagram.com/akash_sinha08/

Terraform Modules

Example of Terraform Module

Example 1:

Your manager has asked you to create resources for a project. You are required to set up the following services:

- Azure App Service Plan
- Azure Function App
- Storage Account
- Key Vault
- Event Hub

Example 2:

Your manager has asked you to create resources for a project. You are required to set up the following services:

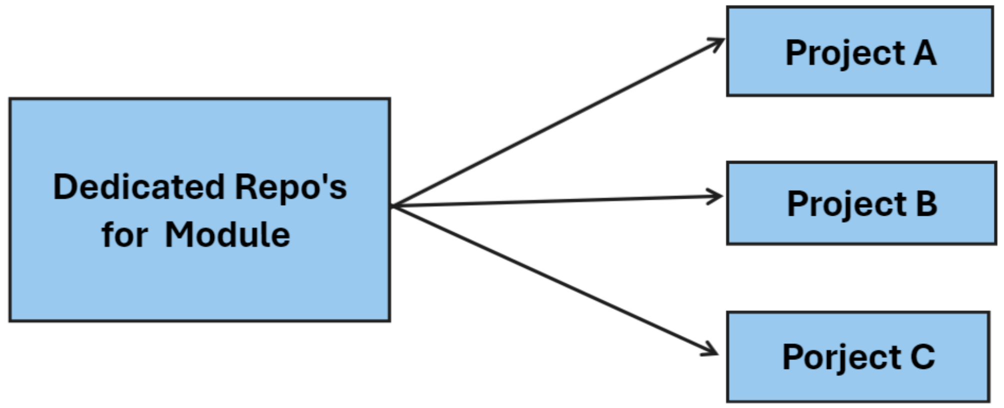
- Azure Kubernetes Service (AKS)
- Azure Container Registry (ACR)
- API Management (APIM)
- Key Vault

Overview of Terraform Module

- Terraform Modules allow us to centralize the resource configuration, and it makes it easier for multiple projects to re-use the Terraform code.

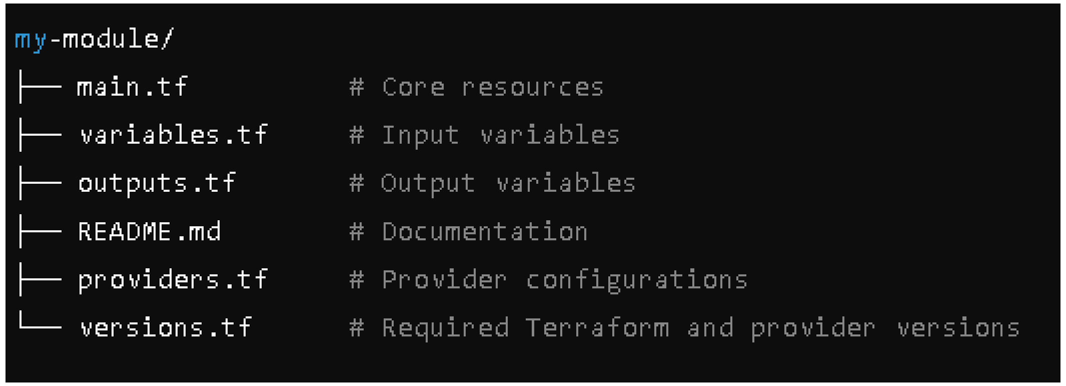
- Instead of writing code from scratch, we can use multiple ready-made modules available.

- The **DRY principle** in Terraform, like in general software development, stands for "**Don't Repeat Yourself.**" This principle encourages writing infrastructure code in a way that avoids duplication, promotes reuse, and ensures maintainability.



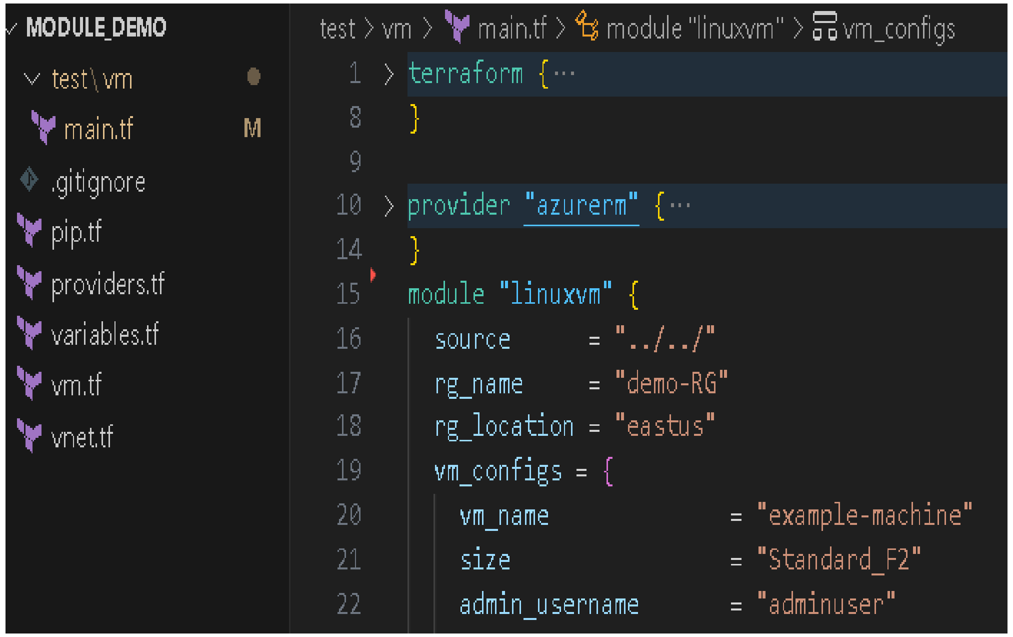
Best practices for creating Modules

- **Define a Clear Module Structure --**A well-structured Terraform module should be easy to navigate. The basic structure of a Terraform module includes:



```
my-module/
├── main.tf          # Core resources
├── variables.tf     # Input variables
├── outputs.tf       # Output variables
├── README.md        # Documentation
├── providers.tf     # Provider configurations
└── versions.tf      # Required Terraform and provider versions
```

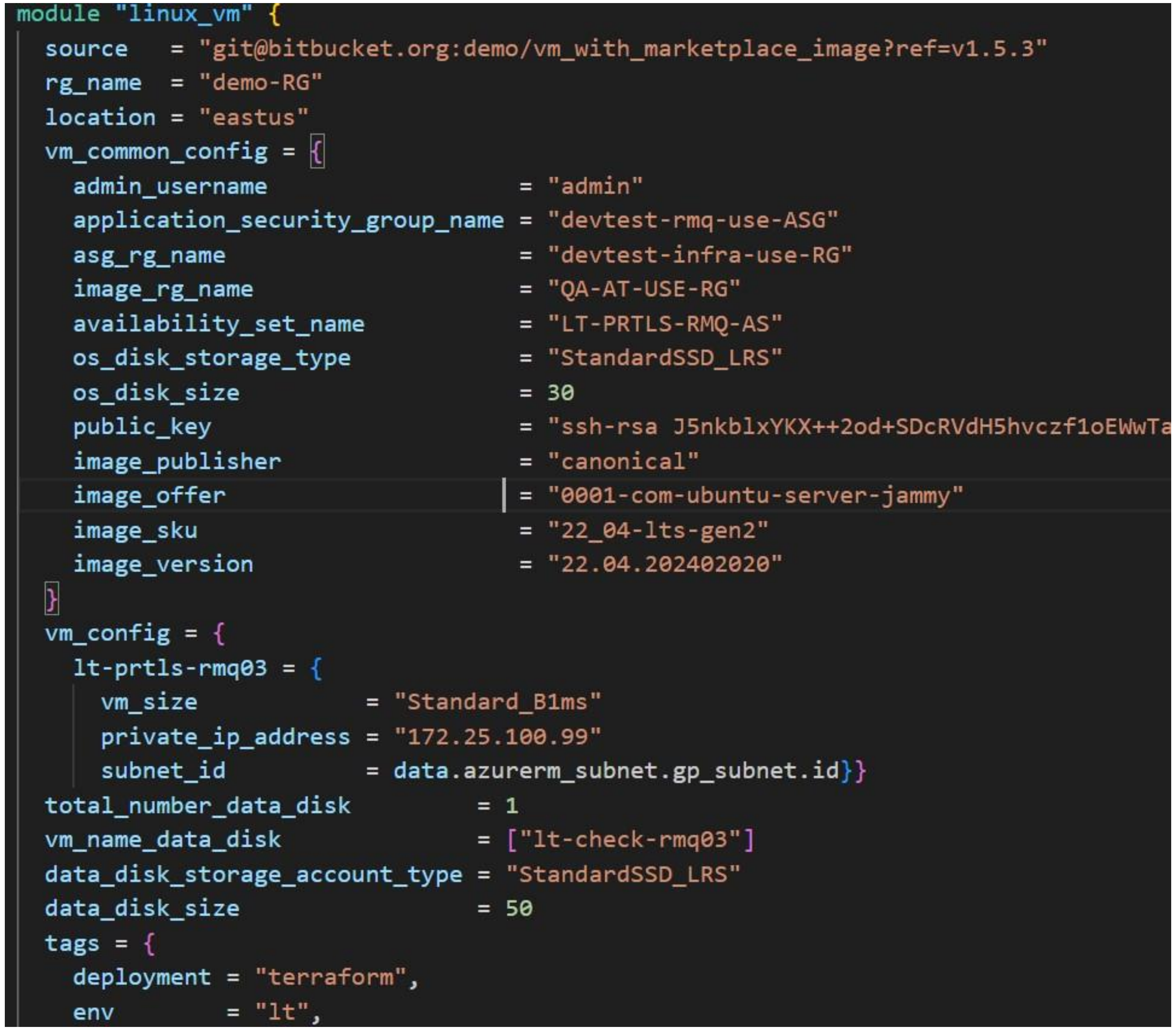
After creating the module, a dedicated test directory is set up to validate the module's functionality, with configurations and test cases to ensure the module behaves as expected in different scenarios.



```
terraform {···
}

provider "azurerm" {···
}
module "linuxvm" {
  source      = "../../"
  rg_name     = "demo-RG"
  rg_location = "eastus"
  vm_configs = {
    vm_name                = "example-machine"
    size                   = "Standard_F2"
    admin_username         = "adminuser"
```

Calling a Module

- Module source code can be present in a wide variety of locations including:
- GitHub, Local Paths, Terraform Registry, Azure Repos , HTTP URLs
- To reference a module, you need to make use of module block and source
- Terraform uses this during the module installation step of terraform init to download the source code to a directory on local disk so that other Terraform commands can use it.

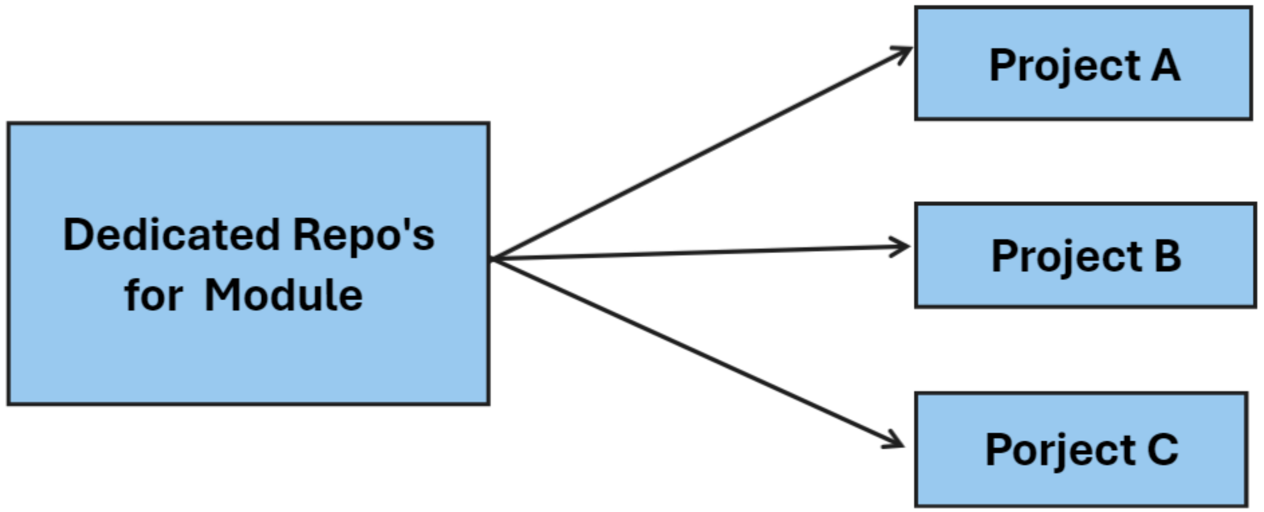


```
module "linux_vm" {
  source   = "git@bitbucket.org:demo/vm_with_marketplace_image?ref=v1.5.3"
  rg_name  = "demo-RG"
  location = "eastus"
  vm_common_config = {
    admin_username                  = "admin"
    application_security_group_name = "devtest-rmq-use-ASG"
    asg_rg_name                     = "devtest-infra-use-RG"
    image_rg_name                   = "QA-AT-USE-RG"
    availability_set_name           = "LT-PRTLS-RMQ-AS"
    os_disk_storage_type            = "StandardSSD_LRS"
    os_disk_size                    = 30
    public_key                      = "ssh-rsa J5nkblxYKX++2od+SDcRVdH5hvczf1oEWwTa
    image_publisher                 = "canonical"
    image_offer                     = "0001-com-ubuntu-server-jammy"
    image_sku                       = "22_04-lts-gen2"
    image_version                   = "22.04.202402020"
  }
  vm_config = {
    lt-prtls-rmq03 = {
      vm_size            = "Standard_B1ms"
      private_ip_address = "172.25.100.99"
      subnet_id          = data.azurerm_subnet.gp_subnet.id}}
  total_number_data_disk         = 1
  vm_name_data_disk              = ["lt-check-rmq03"]
  data_disk_storage_account_type = "StandardSSD_LRS"
  data_disk_size                 = 50
  tags = {
    deployment = "terraform",
    env        = "lt",
```

Root vs Child Modules

- Root Module resides in the main working directory of your Terraform configuration. This is the entry point for your infrastructure definition

- A module that has been called by another module is often referred to as a child module.



Join us in our Adventure

https://www.linkedin.com/in/akash-kumar-480b3858/

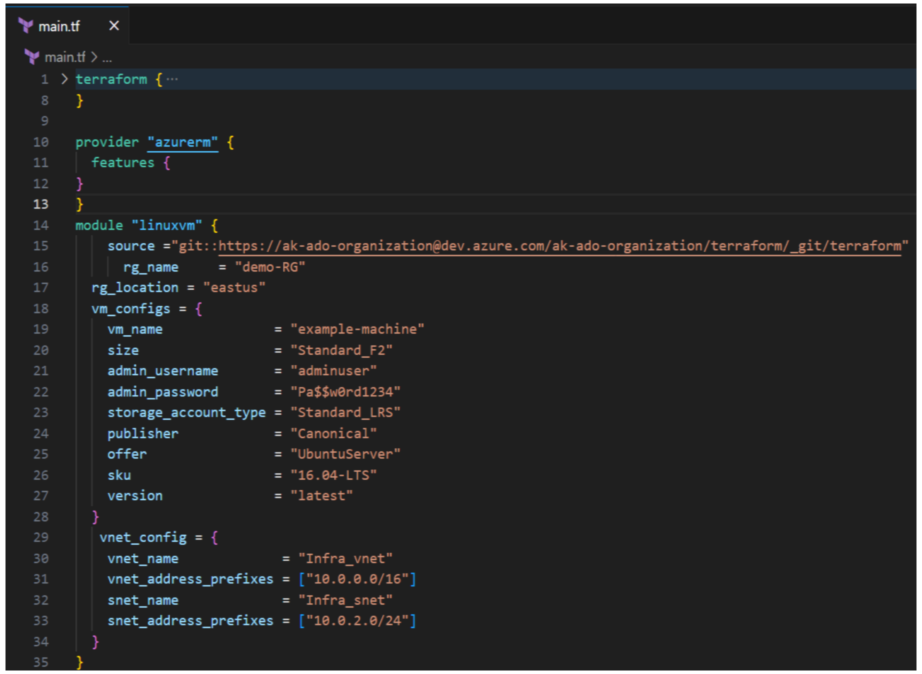
https://www.instagram.com/akash_sinha08/

Remote State Management

Basics of Backends

Backends determine where Terraform stores its state.

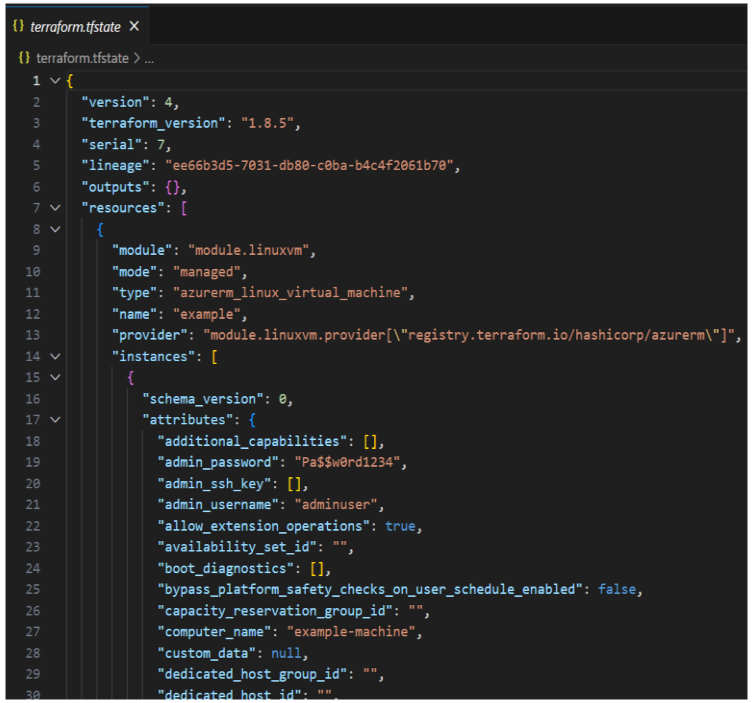
By default, Terraform implicitly uses a backend called local to store state as a local file on disk.



main.tf

main.tf > ...

```hcl
terraform {···
}

provider "azurerm" {
  features {
}
}
module "linuxvm" {
    source ="git::https://ak-ado-organization@dev.azure.com/ak-ado-organization/terraform/_git/terraform"
      rg_name      = "demo-RG"
  rg_location = "eastus"
  vm_configs = {
    vm_name              = "example-machine"
    size                 = "Standard_F2"
    admin_username       = "adminuser"
    admin_password       = "Pa$$w0rd1234"
    storage_account_type = "Standard_LRS"
    publisher            = "Canonical"
    offer                = "UbuntuServer"
    sku                  = "16.04-LTS"
    version              = "latest"
  }
   vnet_config = {
    vnet_name             = "Infra_vnet"
    vnet_address_prefixes = ["10.0.0.0/16"]
    snet_name             = "Infra_snet"
    snet_address_prefixes = ["10.0.2.0/24"]
  }
}
```



terraform.tfstate

terraform.tfstate > ...

```json
{
  "version": 4,
  "terraform_version": "1.8.5",
  "serial": 7,
  "lineage": "ee66b3d5-7031-db80-c0ba-b4c4f2061b70",
  "outputs": {},
  "resources": [
    {
      "module": "module.linuxvm",
      "mode": "managed",
      "type": "azurerm_linux_virtual_machine",
      "name": "example",
      "provider": "module.linuxvm.provider[\"registry.terraform.io/hashicorp/azurerm\"]",
      "instances": [
        {
          "schema_version": 0,
          "attributes": {
            "additional_capabilities": [],
            "admin_password": "Pa$$w0rd1234",
            "admin_ssh_key": [],
            "admin_username": "adminuser",
            "allow_extension_operations": true,
            "availability_set_id": "",
            "boot_diagnostics": [],
            "bypass_platform_safety_checks_on_user_schedule_enabled": false,
            "capacity_reservation_group_id": "",
            "computer_name": "example-machine",
            "custom_data": null,
            "dedicated_host_group_id": "",
            "dedicated_host_id": "",
```

Challenges with Local Backend

In modern infrastructure-as-code (IaC) practices, especially when using **Terraform**, collaboration among teams is key.

When working with Terraform, the **state file** holds critical information about the infrastructure, including resources, configurations, and dependencies

Managing the state file properly becomes essential for teams that collaborate on the same infrastructure project.

Why Not Store the State File Locally?

- **No Collaboration**: If the state file is stored locally on one developer’s laptop, other team members cannot access or modify the infrastructure without having access to the same state file. Each team member may have different local copies, leading to inconsistencies and conflicts.

- **Risk of Data Loss**: If the local machine is lost, damaged, or corrupted, the Terraform state file could be lost, resulting in a significant issue as it would be difficult to track the state of infrastructure and make safe updates.

- **Concurrency Issues**: Local state files don’t support locking mechanisms, meaning multiple team members could make concurrent changes, leading to conflicts and potentially overwriting changes.

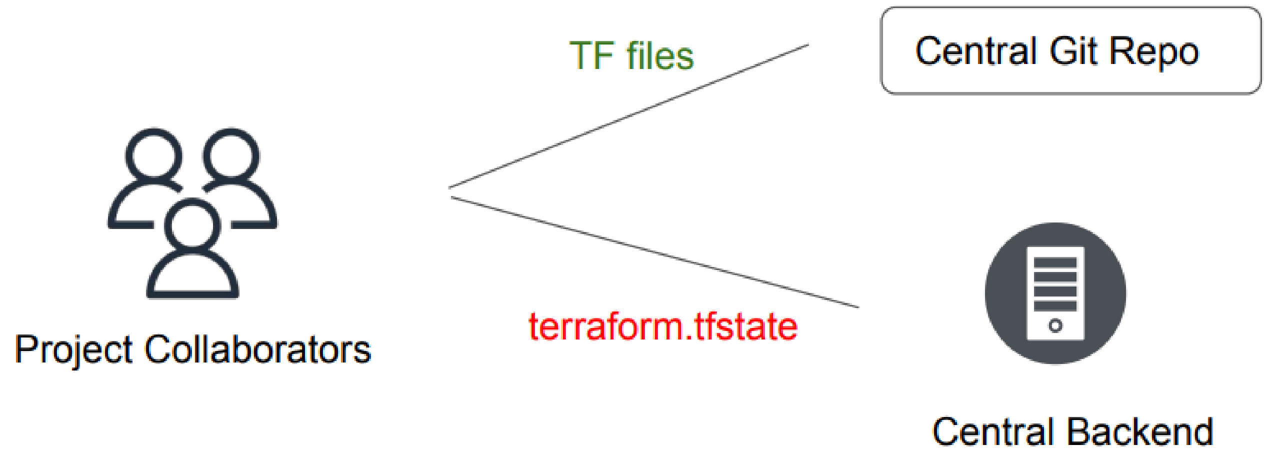
- **Auditability and Versioning**: Locally stored state files don’t inherently provide a clear history of changes. Collaboration tools that integrate with remote backends offer better versioning and audit logs.

Ideal Architecture

Following describes one of the recommended architectures:

- 1. The Terraform code is stored in a Git repository, providing advantages such as collaboration, versioning, change management, and automation.

- 2. The State file is stored in a Central backend.



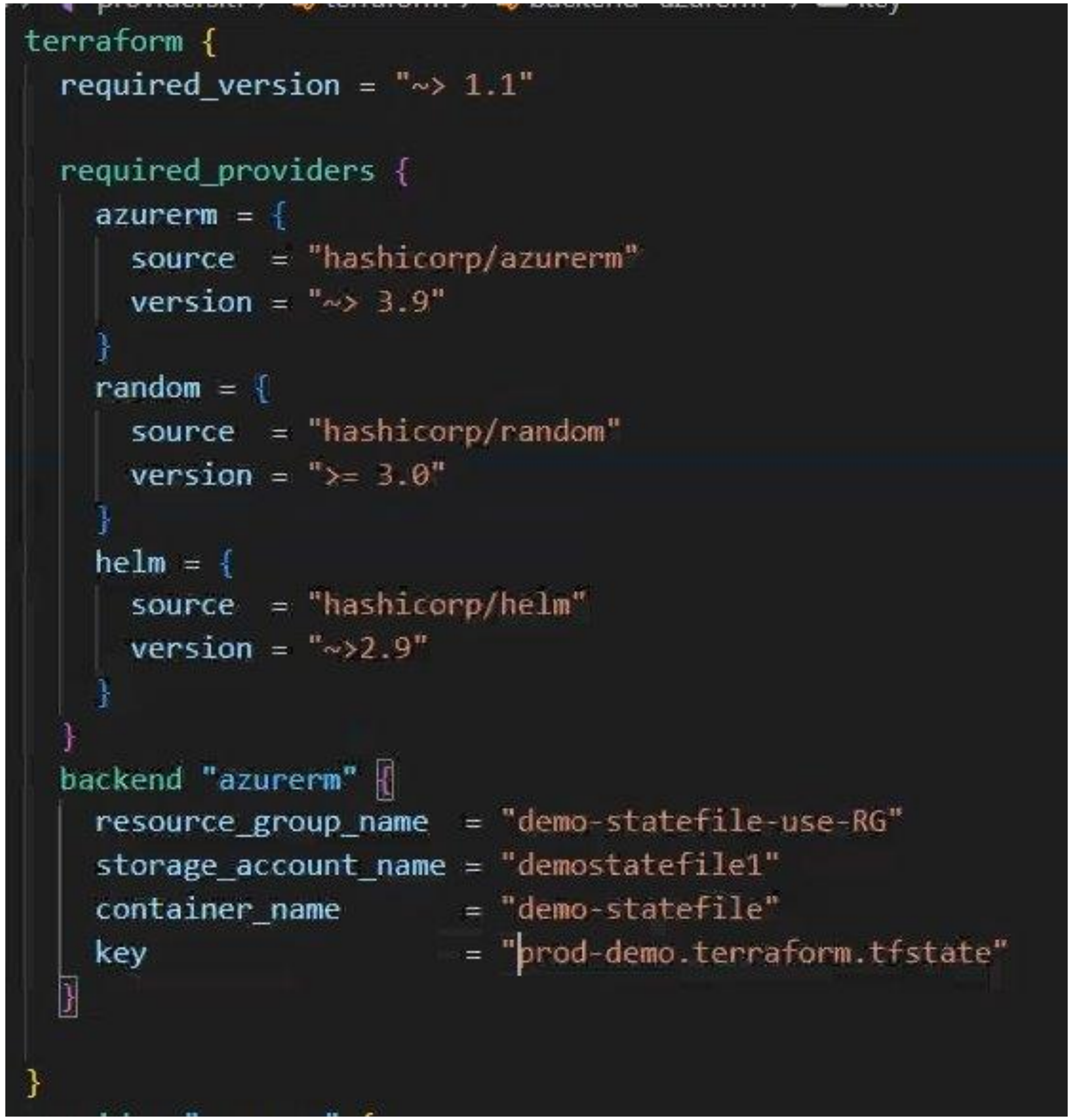
Backends Supported in Terraform

Terraform supports multiple backends that allows remote service related operations.

Some of the popular backends include:

- **Amazon S3**:
- **Azure Blob Storage**
- **Google Cloud Storage (GCS)**
- **HashiCorp Consul**
- **HashiCorp Terraform Cloud/Enterprise**

Azurerm backend Terraform



```
terraform {
  required_version = "~> 1.1"

  required_providers {
    azurerm = {
      source  = "hashicorp/azurerm"
      version = "~> 3.9"
    }
    random = {
      source  = "hashicorp/random"
      version = ">= 3.0"
    }
    helm = {
      source  = "hashicorp/helm"
      version = "~>2.9"
    }
  }
  backend "azurerm" {
    resource_group_name  = "demo-statefile-use-RG"
    storage_account_name = "demostatefile1"
    container_name       = "demo-statefile"
    key                  = "prod-demo.terraform.tfstate"
  }

}
```

Join us in our Adventure

https://www.linkedin.com/in/akash-kumar-480b3858/

https://www.instagram.com/akash_sinha08/