|  |  |
| --- | --- |
| Overview | 1 |
| Pre-Requisites | 2 |
| Azure Repositories | 3 |
| Configuration Settings | 4 |
| MAIN | 6 |
| TFVARS | 8 |
| EXECUTION | 10 |
| VERSION HISTORY | 11 |

CONTENT

**OVERVIEW**

In Azure, we can deploy resources from Portal, Azure cloud shell, Azure CLI and PowerShell. In order to streamline the process like all required resources to host an application or a simple VM, testing, and ensuring same application behavior in different environments, Infrastructure as a Code (IaaC) brings one solution to all these challenges. Define the infrastructure in the code which becomes part of your project. Just like the application code, you store the infrastructure code in a source repository and version it. Anyone on your team can run the code and deploy similar environments.

Among many options in IaaC for Azure, Azure Resource Manager (ARM) and Terraform are widely popular solutions. Also, desired state can be automated through Configuration as a Code (CaaC). Advantasure is using :

* Terraform - IaaC
* Azure Git - Version control
* YAML - CaaC
* Bamboo, Azure DevOps as CI/CD tool

This POC describes how to use the existing Terraform modules for deployment using Terraform CLI.

PRE-REQUISITES

1. Ensure you have Credentials to be used in Terraform modules Azure Devops services
2. Ensure sufficient access on Organization/ Project to manage repositories in Azure DevOps services.
3. Install [Terraform CLI] (https://www.terraform.io/downloads.html)
4. Install [Azure CLI] (<https://docs.microsoft.com/en-us/cli/azure/install-azure-cli>)

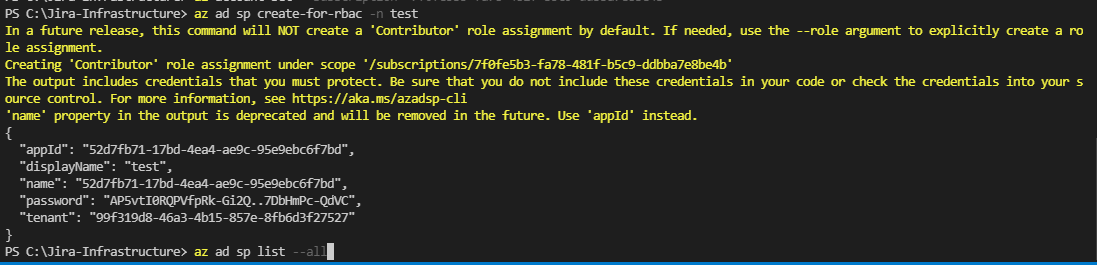
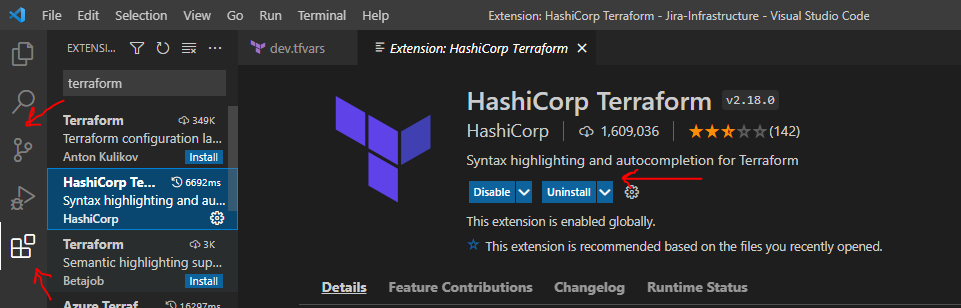
Note :-

- Terraform - Authenticating using the Azure CLI` is going to be same for WindowsOS too.

* Azure CLI Login : az login
* SP Account creation : az ad sp create-for-rbac -n terraform
* Check SP accounts : az ad sp list --all
* Set Specific Subscription : az account set --subscription="SUBSCRIPTION\_ID"

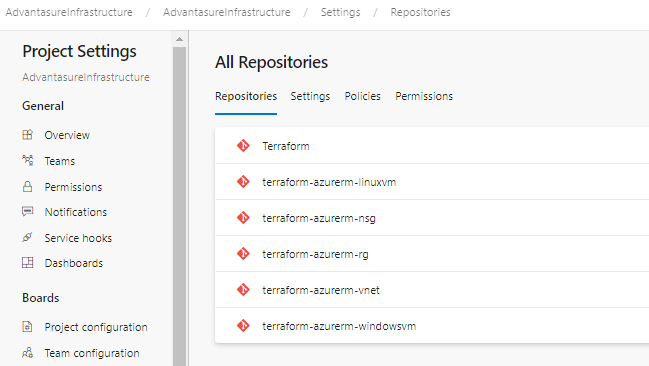
1. Install [VS Code Editor] (https://code.visualstudio.com/download)
2. Install [HashiCorp Terraform plugin for VS Code and Azure Git] (https://marketplace.visualstudio.com/items?itemName=HashiCorp.terraform)
3. To Deploy resources through code, you need a service principal account.

### 

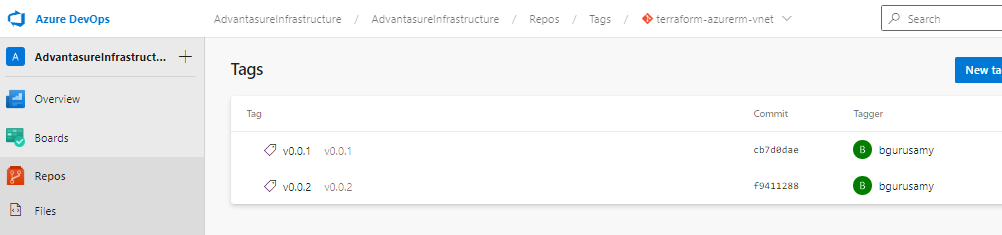
 

AZURE REPOSITORIES

All the resources modules are store Azure git the below location.



Also, each repository must have tags to publish as module and should be like below.



CONFIGURATION

To implement IaaC through Terraform, we need to set the configuration:

* Provider
* Terraform
* Backend (optional)
* Subscription
* Tenant

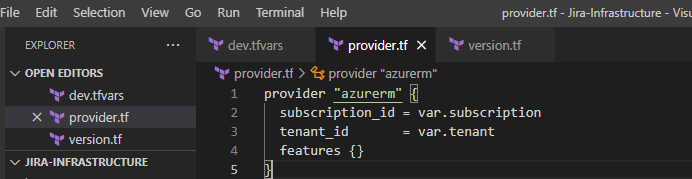
Every project is included with below files where we declare the setting values:

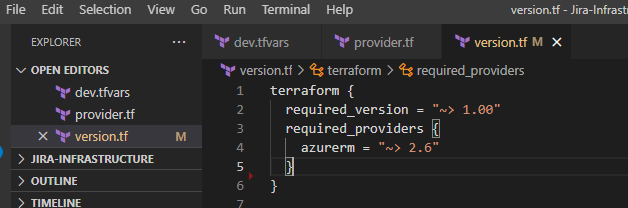
* Provider.tf
* Version.tf

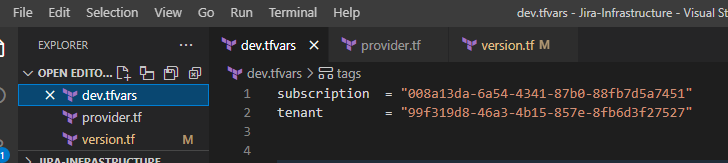
In Provider.tf file, we declare the provider (azurerm) and its version and other backend features details.

In version.tf, we declare terraform version and other backend settings.

We pass the subscription and Tenant information in tfvars file so that the same code can be deployed in any environment.



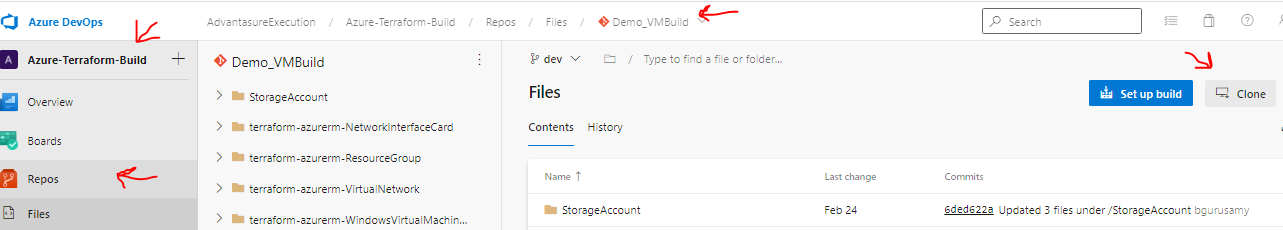




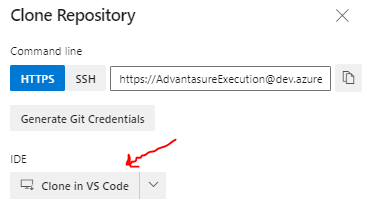
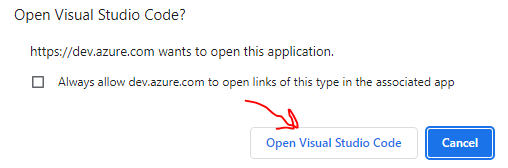
MAIN

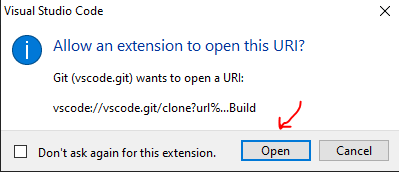
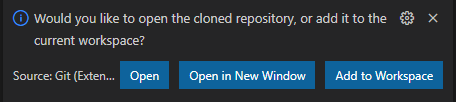
This main.tf can also be referred as ‘Exterior Module’ where we call all the resource modules and pass the tfvars file references.

In Azure DevOps > Project > Repos > click on clone button:



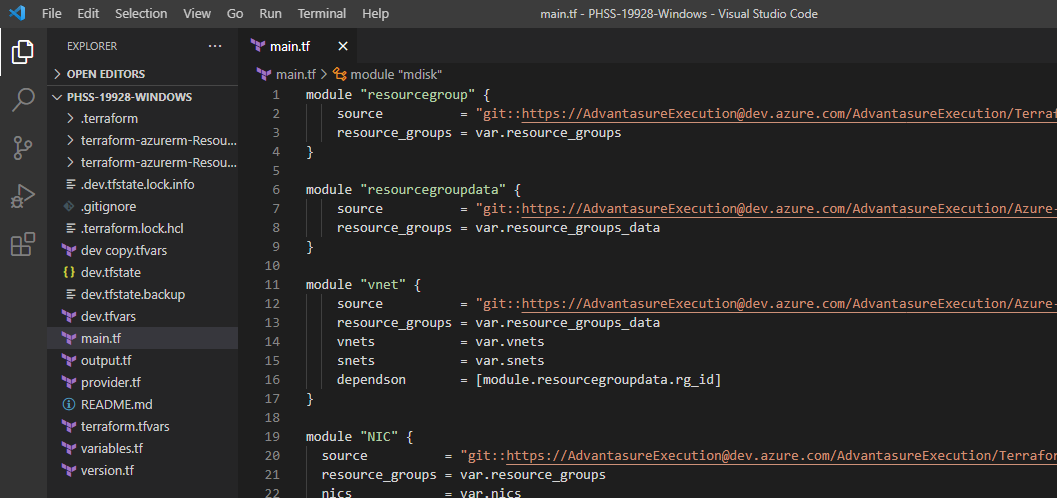
**NOTE**: DO NOT CLONE/MODIFY MODULES REPOS (FROM MODULES PORJECT) AS ALL OTHER BUILD FILES ARE DEPENDENT ON THE SAME.

Choose Repository location in your local machine then a notification will appear as above to launch.

NOTE: For most of the resources, modules are already created. We just need to refer those modules in this main.tf terraform file.



In main.tf, we call the required modules (to create or refer existing module based on repo name /version) and pass the required arguments using variables.

These variables are defined in variables.tf file separately to maintain consistency.

Also, to retrieve specific results, we use output.tf file to extract the resource details.

This file requires four types of parameters:

Name

Name of the module must be unique within the file and it is case-sensitive. This is part of the syntax to call any module and helps in extracting output of the resource. (e.g. dependson argument)

Source

We need to pass the module location, can be local or any source control

Arguments

To know which arguments are required for the resource to create/refer, please refer to variables.tf for the respective modules. (e.g., in the above image, ‘resourcegroup’ module has ‘resource\_groups’ argument is a variable reference within the resource group module located in the source parameter value.)

Dependson

Dependson parameter is required to pass only the resource has any dependency. The values will be output of the other modules. (e.g., dependson = [mdule.resourcegroup.rg\_id]

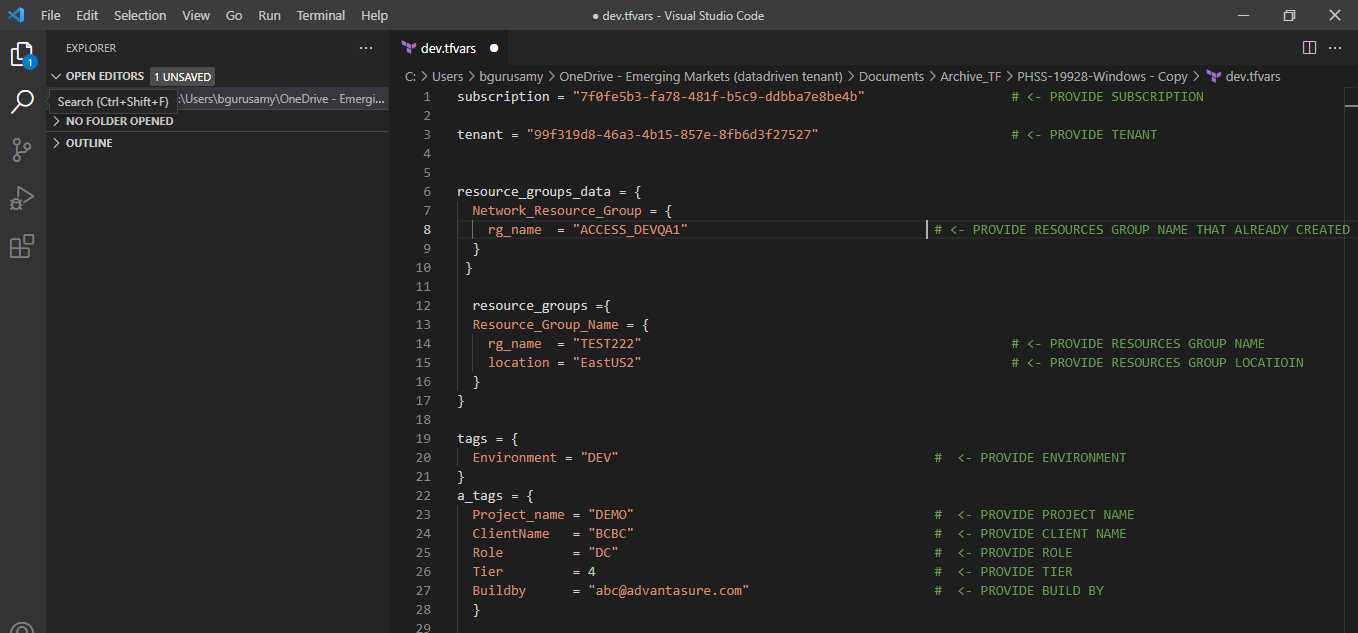
NOTE : This parameter may not be available for all modules. Also, this is declared directly in source main file of resource module and hence not visible in variables.tf.

TFVARS

Follow below example to update in tfvars.

NOTE : follow the definition/syntax/type of variable and declare variables accordingly. (e.g., if the variable type is map and required to pass multiple values, create nested map and pass the key value pair accordingly)

1. Please provide following details.
2. Subscription ID
3. Tenant ID
4. Resource Groups
5. rg\_name = "\*\*\*\*\*\*\*\*\*\*" # <- PROVIDE RESOURCES GROUP NAME WHICH IS ALREADY CREATED
6. rg\_name = “\*\*\*\*\*\*\*\*\*” # <- PROVIDE RESOURCES GROUP NAME -NEW RESOURCES TO BE CREATE.
7. Location = “\*\*\*\*\*\*\*\*” # <- PROVIDE RESOURCES GROUP LOCATIOIN
8. Tags
9. Environment =”\*\*\*” # <- PROVIDE ENVIRONMENT -NEW RESOURCES TO BE CREATE.
10. a\_tags = {
11. Project\_name = "\*\*\*\*" # <- PROVIDE PROJECT NAME
12. ClientName = "\*\*\*\*" # <- PROVIDE CLIENT NAME
13. Role = "\*\*" # <- PROVIDE ROLE
14. Tier = \* # <- PROVIDE TIER
15. Buildby = "\*\*\*\*\*\*\*\*\*" # <- PROVIDE BUILD BY



1. Vnets
2. Vnet name = “\*\*\*\*\*” # <- PROVIDE VNET NAME WHICH IS ALREADY CREATED
3. Address space = “\*\*\*\*\*” # <- PROVIDE VNET SPACE WHICH IS ALREADY CREATED
4. Snets
5. snet\_name = "\*\*\*\*\*\*\*\*\*\*\*" # <- PROVIDE SUBNET NAME WHICH IS ALREADY CREATED
6. address\_prefix = ["\*\*\*\*\*\*"] # <- PROVIDE SUBNET SPACE WHICH IS ALREADY CREATED
7. nics
8. nic name = "\*\*\*\*\*\*\*\*\*\*\*" # <- PROVIDE NIC NAME TO BE 77CREATED
9. static ip = "\*\*\*\*\*\*\*" # <- PROVIDE NIC IP TO VM TO BE CREATED.

IF YOU NEED TO CREATE MULTIPLE VMS YOU HAVE TO PROVIDE AS SHOWN BELOW

1. nic2
2. nic name = "\*\*\*\*\*\*\*\*\*\*\*" # <- PROVIDE NIC NAME TO BE 77CREATED
3. static ip = "\*\*\*\*\*\*\*” # <- PROVIDE NIC IP TO VM TO BE CREATED

EXECUTION

Create and Deploy Resources.

Once all the changes are correctly done your local TFVAR FILE, the next step is to deploy the infrastructure resources to the Azure portal described command as follow.

title: Terraform Command Basics

description: Learn Terraform Commands like init, validate, plan, apply and destroy

## Step-01: Introduction

- Understand basic Terraform Commands

1. terraform init

Used to Initialize a working directory containing terraform config files. This is the first command that should be run after writing a new Terraform configuration Downloads Providers.

2. terraform validate

Validates the terraform configurations files in that respective directory to ensure they are syntactically valid and internally consistent.

3. terraform plan - terraform plan -var-file="dev.tfvars" -state="dev.tfstate"

Creates an execution plan - Terraform performs a refresh and determines what actions are necessary to achieve the desired state specified in configuration files and provide details resources to be created.

Used to apply the changes required to reach the desired state of the configuration.

4. terraform apply - terraform apply -var-file="dev.tfvars" -state="dev.tfstate" -auto-approve

By default, apply scans the current directory for the configuration and applies the changes appropriately.

5. terraform destroy - terraform destroy -var-file="dev.tfvars" -state="dev.tfstate" -auto-approve

Used to destroy the Terraform-managed infrastructure.

VERSION HISTORY

Author : Baskaran Guruswamy

Reviewer :

Creation Date : 21/03/2022