Agenda: IaC using Terraform Templates

- Overview of Terraform
- Terraform Files Structure
- Terraform Commands
- Run a Terraform plan from Azure Cloud Shell
- Provision Terraform Tasks in Azure Pipeline – Classic Editor
- Provision Terraform Task in Azure Pipeline – YAML File

Overview of Terraform

- Terraform by HashiCorp, is an **open-source tool** for building, changing, and versioning infrastructure safely and efficiently. Terraform can manage existing and popular cloud service providers as well as custom in-house solutions.
- Terraform's command-line interface provides you with a simple way to provision Azure resources such as virtual machines, containers, and networks.
- Configuration files describe to Terraform the components needed to run a single application or your entire datacenter.
- Terraform generates an **execution plan** that describes what it will do to reach the desired state. You can review this execution plan and then **run the plan** to apply the configuration.
- As your configuration changes, Terraform determines what has changed and creates **incremental execution** plans to update only what's necessary.
- In addition to Azure, Terraform supports other public clouds and private cloud frameworks also.

Terraform components:

Configuration files.

- Text-based configuration files allow you to define infrastructure and application configuration.

Note: The order of items (such as variables and resources) as defined within the configuration file does not matter, because Terraform configurations are declarative.

- You typically write Terraform code in HCL (Hashicorp Language) but you can also express your Terraform configuration by using JSON.
  - **Terraform**. The Terraform format is easier for users to review, thereby making it more user friendly. It supports comments, and is the generally recommended format for most Terraform files. Terraform files ends in .tf
  - **JSON**. The JSON format is mainly for use by machines for creating, modifying, and updating configurations. However, it can also be used by Terraform operators if you prefer. JSON files end in .tf.json.

How are Terraform files structured?

- ***main.tf:*** This file is often called a **Terraform plan***.* This file holds your Terraform **configuration code**. Your Terraform plan specifies the infrastructure resources that you need.
- ***terraform.tfvars:*** A *.tfvars* file is a good way to maintain larger sets of variables.
- ***terraform.tfstate:*** It's called the state file. It's a JSON file that Terraform manages. It helps map the resources that you define in your plan to the running resources that your plan produces.

main.tf

```
terraform {
 required_version = "> 0.12.0"
# The below section if not specified the terraform.tfstate file is created in current directory of your machine.
 backend "azurerm" {
  storage_account_name = "dssdemstorage"
  container_name = "tfstate"
  key = "terraform.tfstate"
  access_key = "z1bZ0s+ prOMjjnnxSLX24cqD8yZdbn8efSj7tQ=="
  features{}
 }
}
# Configure the Azure provider
#Terraform relies on plugins called "providers" to interact with remote systems.
provider "azurerm" {
  version = "~>2.0"
  subscription_id = "f9baec73-91eb-4458-bd0d-965c1973526d"
  client_id = "e1a28c61-a0ab-49d0-bd7f-914545c228c9"
  client_secret = ".~f7Z5OcJN1b2yW7kV3D_SoP-j5v6v4~oZ"
  tenant_id = "82d8af3b-d3f9-465c-b724-0fb186cc28c7"

  # The "feature" block is required for AzureRM provider 2.x.
  #If you are using version 1.x, the "features" block is not allowed.
  #It's possible to configure the behaviour of certain resources using the features block
  features {}
}


variable "resource_group_name" {
  default = "my-rg"
  description = "The name of the resource group"
}
variable "resource_group_location" {
  default = "westus"
  description = "The location of the resource group"
}
variable "app_service_plan_name" {
  default     = "my-asp"
  description = "The name of the app service plan"
}
variable "app_service_name_prefix" {
  default     = "my-appsvc"
  description = "The beginning part of the app service name"
}

resource "random_integer" "app_service_name_suffix" {
  min = 1000
  max = 9999
}

#Creating a Resource Group
resource "azurerm_resource_group" "my" { # "azurerm_resource_group" is type and "my" is local name
  name     = var.resource_group_name
  location = var.resource_group_location
}
#Creating an App Service Plan
resource "azurerm_app_service_plan" "my" {   # "azurerm_app_service_plan" is type and "my" is local name
  name                = var.app_service_plan_name
  location            = azurerm_resource_group.my.location
  resource_group_name = azurerm_resource_group.my.name
  kind                = "Linux"
  reserved            = true
  sku {
    tier = "Basic"
    size = "B1"
```

```
 }
}
#Creating an App Service
resource "azurerm_app_service" "my" {
 name            = "${var.app_service_name_prefix}-${random_integer.app_service_name_suffix.result}"
 location        = azurerm_resource_group.my.location
 resource_group_name = azurerm_resource_group.my.name
 app_service_plan_id = azurerm_app_service_plan.my.id
}
output "website_hostname" {
 value       = azurerm_app_service.my.default_site_hostname
 description = "The hostname of the website"
}
```

terraform.tfvars

```
resource_group_name="our-rg"
resource_group_location="eastus"
app_service_plan_name="our-asp"
app_service_name_prefix="our-appsvc"
```

Terraform Commands

1. **Initialize**

   The **terraform init** command initializes your Terraform environment. This command downloads the plug-ins that you need. It also verifies that Terraform can access your plan's state file.

2. **Plan**

   The **terraform plan** command produces an execution plan that's based on your configuration. This command doesn't modify any infrastructure. It's just a way for a human to **review what changes** will be made if the plan is applied.

   You typically **omit this command** when running Terraform in a CI/CD pipeline. By the time your plan reaches the pipeline, your plan should express your infrastructure requirements and you should understand the effect that your plan will have.

3. **Apply**

   The **terraform apply** command runs your execution plan. Think of it as a way to apply the proposed changes that you get from the terraform plan command.

   The terraform apply command is an idempotent operation.

If your Terraform plan changes or some other process inadvertently changes your infrastructure, terraform apply places your infrastructure in the desired state.

4. **Destroy**

The **terraform destroy** command destroys all infrastructure resources that are defined in your plan.

Terraform on Azure

You download Terraform for use in Azure via: Azure Marketplace, Terraform Marketplace, or Azure VMs.

If you download Terraform for the Windows operating system:

Download: https://www.terraform.io/downloads.html

1. Find the install package, which is bundled as a zip file.
2. Copy files from the zip to a local directory such as **d:\terraform**. That is the Terraform PATH, so make sure that the Terraform binary is available on the PATH.
3. To set the PATH environment variable, run the command **set PATH=%PATH%;d:\terraform**, or point to wherever you have placed the Terraform executable.
4. Open an administrator command window at **d:\Terraform** and run the command Terraform to verify the installation. You should be able to view the terraform help output.

Docs overview | hashicorp/azurerm | Terraform Registry

Run a basic Terraform plan from Azure Cloud Shell

Create a Plan and Set Variables

1. Save the above Script in main.tf file

Provision Infrastructure

2. terraform init
3. terraform plan
4. terraform apply

Verify the results

5. terraform output

Examine the state file

6. cat terraform.tfstate

Destroy your infrastructure

7. terraform destroy

Example:

terraform init

terraform plan -out=tfplan

terraform apply -auto-approve tfplan

Terraform will exit with an error status if there are any variables whose values could not be set.

Provision Azure resources in Azure Pipelines with Terraform (Classic Editor)

1. Add the following to Azure Repository (HelloWorldApp.Templates\main.tf)

main.tf

```
terraform {
  required_version = "> 0.12.0"
# The below section if not specified the terraform.tfstate file is created in current directory of your machine.
  backend "azurerm" {
    storage_account_name = "dsscatalogstorage1"
    container_name = "terraform"
    key = "terraform.tfstate"
    access_key = "2mSwIFykwoP9g3mxCxywe03qu392TTz+o1osusD6FBKISXHUGsl+0MXF4nugBlLd9Znki5hK9z
B833TyAXgwIw=="
  }
}
# Configure the Azure provider
#Terraform relies on plugins called "providers" to interact with remote systems.
provider "azurerm" {
  subscription_id = "51081bf2-da0d-4998-9462-b59b512f8690"
  client_id = "cc02bf6e-ac3c-40c6-a139-bc2262ceb187"
  client_secret = ".Y3I6I4d--kmgMZYT.d6BXNa7oc-MMFKO1"
  tenant_id = "82d8af3b-d3f9-465c-b724-0fb186cc28c7"
  # The "feature" block is required for AzureRM provider 2.x.
  #If you are using version 1.x, the "features" block is not allowed.
  #It's possible to configure the behaviour of certain resources using the features block
  features {}
}

variable "resource_group_name" {
 default = "my-rg"
```

```
  description = "The name of the resource group"
}
variable "resource_group_location" {
  default = "westus"
  description = "The location of the resource group"
}
variable "app_service_plan_name" {
  default     = "my-asp"
  description = "The name of the app service plan"
}
variable "app_service_name_prefix" {
  default     = "my-appsvc"
  description = "The beginning part of the app service name"
}

resource "random_integer" "app_service_name_suffix" {
  min = 1000
  max = 9999
}

#Creating a Resource Group
resource "azurerm_resource_group" "my" { # "azurerm_resource_group" is type and "my" is local name
  name     = var.resource_group_name
  location = var.resource_group_location
}
#Creating an App Service Plan
resource "azurerm_app_service_plan" "my" {   # "azurerm_app_service_plan" is type and "my" is local name
  name                = var.app_service_plan_name
  location            = azurerm_resource_group.my.location
  resource_group_name = azurerm_resource_group.my.name
  kind                = "Linux"
  reserved            = true
  sku {
    tier = "Basic"
    size = "B1"
  }
```

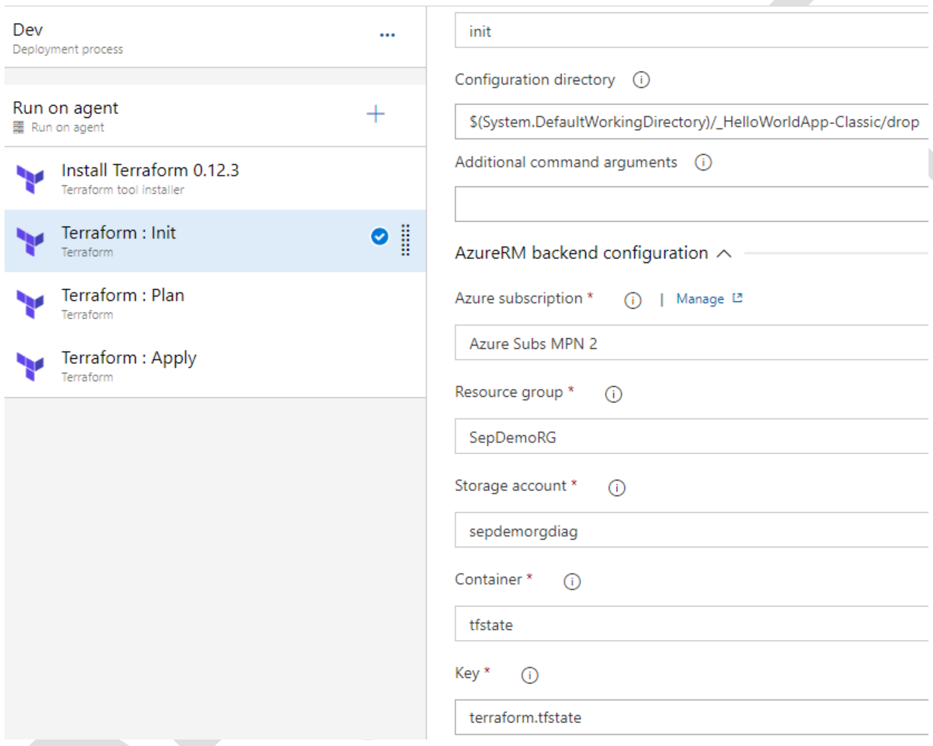
```
}
#Creating an App Service
resource "azurerm_app_service" "my" {
  name             = "${var.app_service_name_prefix}-${random_integer.app_service_name_suffix.result}"
  location         = azurerm_resource_group.my.location
  resource_group_name = azurerm_resource_group.my.name
  app_service_plan_id = azurerm_app_service_plan.my.id
}

resource "azurerm_storage_account" "my" {
  name                 = "dssexamplesa"
  resource_group_name      = azurerm_resource_group.my.name
  location              = azurerm_resource_group.my.location
  account_tier           = "Standard"
  account_replication_type = "LRS"
}

resource "azurerm_mssql_server" "my" {
  name                     = "dssmssqlserver456"
  resource_group_name          = azurerm_resource_group.my.name
  location                   = azurerm_resource_group.my.location
  administrator_login          = "dssadmin"
  administrator_login_password = "Password@123"
  version                    = "12.0"
  minimum_tls_version          = "1.2"
}

resource "azurerm_mssql_database" "my" {
  name          = "DemoDb"
  server_id      = azurerm_mssql_server.my.id
  collation      = "SQL_Latin1_General_CP1_CI_AS"
  sku_name       = "Basic"
}

output "website_hostname" {
  value       = azurerm_app_service.my.default_site_hostname
```

```
 description = "The hostname of the website"
}
```

2. In Azure Portal create a Storage Account and Container
3. In the build pipe line add a Copy files task
    a) Source Folder = HelloWorldApp.Templates
    b) Contents=*.tf
    c) Target Folder = $(build.artificatstagingdirectory)
4. Add a Classic Build Pipeline as below

Note: Please enter complete path for **Configuration directory**

Eg: $(System.DefaultWorkingDirectory)/_HelloWorldApp-Classic/drop/TerraformTemplates


Dev
Deployment process
Run on agent
Run on agent
Install Terraform 0.12.3
Terraform tool installer
Terraform : Init
Terraform
Terraform : Plan
Terraform
Terraform : Apply
Terraform

init

Configuration directory: $(System.DefaultWorkingDirectory)/_HelloWorldApp-Classic/drop

Additional command arguments:

AzureRM backend configuration

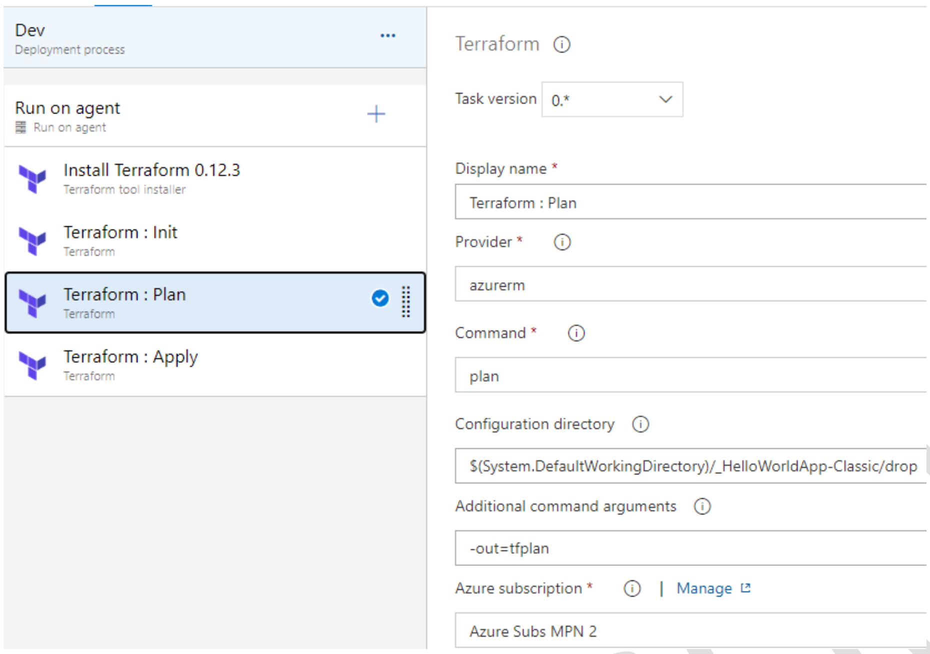
Azure subscription * | Manage: Azure Subs MPN 2

Resource group *: SepDemoRG

Storage account *: sepdemorgdiag

Container *: tfstate

Key *: terraform.tfstate

5. 

Dev
Deployment process

Run on agent
Run on agent

Install Terraform 0.12.3
Terraform tool installer

Terraform : Init
Terraform

Terraform : Plan
Terraform

Terraform : Apply
Terraform

Terraform

Task version: 0.*

Display name *: Terraform : Plan

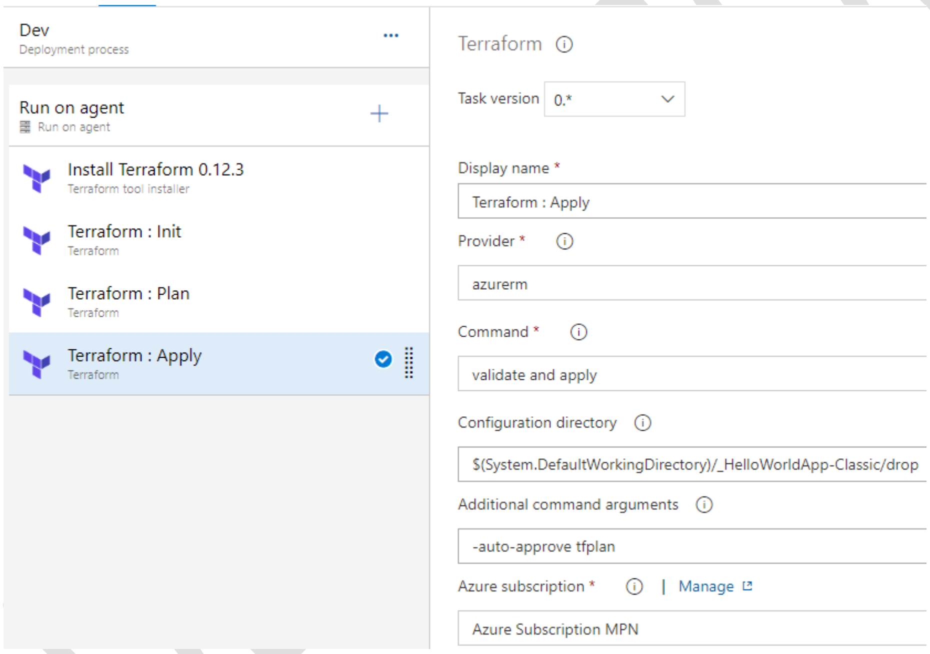
Provider *: azurerm

Command *: plan

Configuration directory: $(System.DefaultWorkingDirectory)/_HelloWorldApp-Classic/drop

Additional command arguments: -out=tfplan

Azure subscription * | Manage: Azure Subs MPN 2

6. 

Dev
Deployment process

Run on agent
Run on agent

Install Terraform 0.12.3
Terraform tool installer

Terraform : Init
Terraform

Terraform : Plan
Terraform

Terraform : Apply
Terraform

Terraform

Task version: 0.*

Display name *: Terraform : Apply

Provider *: azurerm

Command *: validate and apply

Configuration directory: $(System.DefaultWorkingDirectory)/_HelloWorldApp-Classic/drop

Additional command arguments: -auto-approve tfplan

Azure subscription * | Manage: Azure Subscription MPN

7. Save and Create Release.

[Docs overview | hashicorp/azurerm | Terraform Registry]

Provision Azure resources in Azure Pipelines with Terraform (YAML)

```
trigger:

- none


pool:

  vmImage: 'windows-latest'
```

```
steps:
- task: TerraformInstaller@0
  displayName: 'Install Terraform 0.12.3'
  inputs:
    terraformVersion: '0.12.3'

- task: TerraformTaskV1@0
  displayName: 'Terraform Init'
  inputs:
    provider: 'azurerm'
    command: 'init'
    backendServiceArm: 'Azure Connection'
    backendAzureRmResourceGroupName: 'DemoRG'
    backendAzureRmStorageAccountName: 'dssdemosa'
    backendAzureRmContainerName: 'demo'
    backendAzureRmKey: 'terraform.tfstate'

- task: TerraformTaskV1@0
  displayName: 'Terraform Plan'
  inputs:
    provider: 'azurerm'
    command: 'plan'
    commandOptions: '-out=tfplan'
    workingDirectory: '$(System.DefaultWorkingDirectory)/_HelloWorldApp-Classic/drop'
    environmentServiceNameAzureRM: 'Azure Connection'

- task: TerraformTaskV1@0
  displayName: 'Terraform Apply'
  inputs:
    provider: 'azurerm'
    command: 'apply'
    commandOptions: '-auto-approve tfplan'
    workingDirectory: '$(System.DefaultWorkingDirectory)/_HelloWorldApp-Classic/drop'
    backendServiceArm: 'Azure Connection'
    environmentServiceNameAzureRM: 'Azure Connection'
```

Documentation:

https://docs.microsoft.com/en-us/azure/developer/terraform/