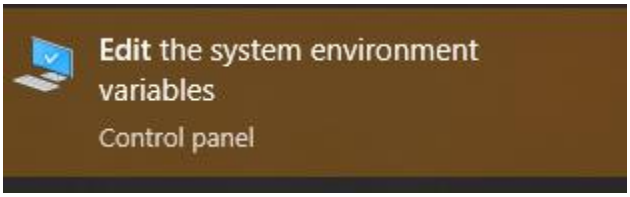
TERRAFORM

About:

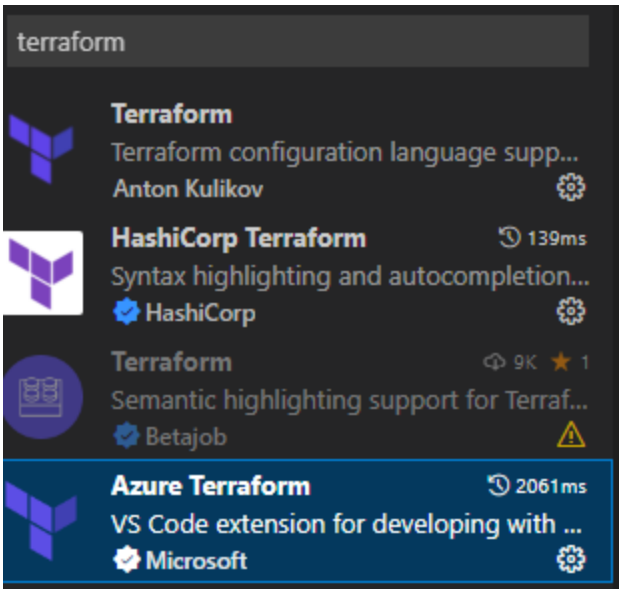
- Terraform and ARM are declarative languages, whereas PowerShell is imperative
- ARM is used for only Azure but Terraform is used many clouds like Azure, AWS, GCP & on premise
- ARM & Terraform are Infrastructure as code
- ARM is Json language
- Terraform is uses HCL language
- HCL = Hashicorp language
- Terraform is easiest language
- Using Terraform we can automate on any cloud.

Installing Terraform on local Machine:

- Open browser and download terraform
- Extract the Zip file
- Copy the path where we have extracted
- Search on windows search bar as “edit the system environment variables”



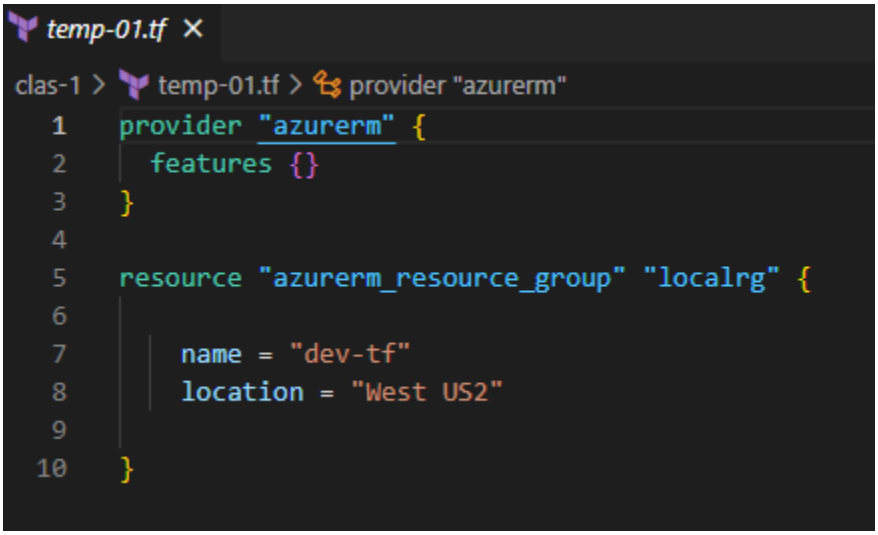
- Click on environment
- In system variables “path”
- Click on new
- Paste the path and \
    - Ex C:krish\
- To check whether Terraform installed or not follow below
- Open powershell
    - CMD: terraform –version
- Go to V.S code and install terraform extensions



- Install above 3
- Install Azure CLI also

Writing Terraform template for Resource Group:

- Create new folder like below
- Using power shell command go to that directory
    - CMD: cd C:\Krish\Devops\Terraform\clas-1
- Now write below code



- Save and run below commands on terminal
- We need to login Azure 1st time using CLI command below
    - CMD: Az login
- Now run below commands on terminal

    - Terraform plan
    - Terraform apply
- Note:
    - “Localrg” on script is Alias. That will be useful to take reference in the template
    - The prompt was asking us to give “yes”, if we need to bypass that we need to give –auto-approve exp: terraform apply –auto-approve

Replacing Resource Group:

- Earlier we have created “dev-tf” resource group using above script, but if you want to replace with qa-rg
- Go to script and change “dev-tf” to “qa-rg” like below
- Before

```
temp-01.tf ×
clas-1 > temp-01.tf > provider "azurerm"
1   provider "azurerm" {
2     features {}
3   }
4
5   resource "azurerm_resource_group" "localrg" {
6
7       name = "dev-tf"
8       location = "West US2"
9
10  }
```

- After

```
temp-01.tf ●
clas-1 > temp-01.tf > resource "azurerm_resource_group" "local
1   provider "azurerm" {
2     features {}
3   }
4
5   resource "azurerm_resource_group" "localrg" {
6
7       name = "qa-tf"
8       location = "West US2"
9
10  }
```

- Now run below commands, then it will remove dev-tf resource group and create new qa-rg
    - Terraform plan
    - Terraform apply –auto-approve

Creating QA and Dev Resource group without deleting:

- Below script will deploy the **dev-tf** and **qa-rg** both resource groups without removing anything

```
provider "azurerm" {
  features {}
}

resource "azurerm_resource_group" "localrg" {

    name = "dev-tf"
    location = "West US2"

}

resource "azurerm_resource_group" "qarg" {

    name = "qa-rg"
    location = "West US2"

}
```

- Now run below commands, then it will deploy both resource groups
    - Terraform plan
    - Terraform apply –auto-approve
- Note: We must give unique Alias for both resource groups, otherwise it will replace the resource groups and only one will deploy.

Creating Storage account:

- Below script will Execute the storage account

```
provider "azurerm" {
  features {

  }
}

resource "azurerm_resource_group" "localrg" {
  name     = "qa-rg"
  location = "West US 2"
}

resource "azurerm_storage_account" "local-storage" {
  name                     = "krishstorage321"
  location                 = azurerm_resource_group.localrg.location
  resource_group_name      = azurerm_resource_group.localrg.name
  account_tier             = "Standard"
  account_replication_type = "GRS"

}
```

- Below PowerShell script will deploy the script
- Note: we must be in same directory before executing below commands

commands.ps1

class-2 > commands.ps1

```
#Terraform calling
terraform init

#validate the template
terraform validate

#Formating the script variables
terraform fmt

#it will create the plan
terraform plan


#it will deploy the script
terraform apply --auto-approve
```

- Above commands will deploy the storage account.

Changing Storage account Redundancy using above template:

- Now go to above script change account replication type from GRS to LRS
- Before

```
provider "azurerm" {
  features {

  }
}

resource "azurerm_resource_group" "localrg" {
  name     = "qa-rg"
  location = "West US 2"
}

resource "azurerm_storage_account" "local-storage" {
  name                     = "krishstorage321"
  location                 = azurerm_resource_group.localrg.location
  resource_group_name      = azurerm_resource_group.localrg.name
  account_tier             = "Standard"
  account_replication_type = "GRS"

}
```

- After

```
resource "azurerm_resource_group" "localrg" {
  name     = "qa-rg"
  location = "West US 2"
}

resource "azurerm_storage_account" "local-storage" {
  name                     = "krishstorage321"
  location                 = azurerm_resource_group.localrg.location
  resource_group_name      = azurerm_resource_group.localrg.name
  account_tier             = "Standard"
  account_replication_type = "LRS"

}
```

- Now run below commands to deploy
  - Terraform plan
  - Terraform apply –auto-approve

Creating Storage Account Using Variables:

- Earlier we have created storage account using hardcoded values, now we are using the variables file
- Create new file folder
- Create 2 new files as
    - main.tf
    - variables.tf
- Now write variables file as below

```
variable "resourceGroupName" {
  type    = string
  default = "dev-rg"
}

variable "location" {
  type    = string
  default = "West Us 3"
}

variable "storageaccountName" {
  type    = string
  default = "devstoragekrish5122"
}

variable "sku" {
  type    = string
  default = "LRS"
}
```

- Now write template for storage account as below

```
provider "azurerm" {
  features {

  }
}

resource "azurerm_resource_group" "rg" {
  name     = var.resourceGroupName
  location = var.location
}

resource "azurerm_storage_account" "storage" {
  name                     = var.storageaccountName
  resource_group_name      = azurerm_resource_group.rg.name
  location                 = azurerm_resource_group.rg.location
  account_tier             = "Standard"
  account_replication_type = var.sku
}
```

- Below PowerShell script will deploy the script
- Note: we must be in same directory before executing below commands

commands.ps1

class-2 > commands.ps1

```
#Terraform calling
terraform init

#validate the template
terraform validate

#Formating the script variables
terraform fmt

#it will create the plan
terraform plan


#it will deploy the script
terraform apply --auto-approve
```

⇨ Now we have created storage account using the variables

Last minute change in Variables:

- Earlier we have written variable as per requirement, but now due to last minute change of requitement we need to change the variables
- Exp: we need to change the resource group location from east us to west us and name demo to dev we need to use below command to change the variables without touching the file

```
CMD: terraform plan -var 'resourceGroupName=dev-rg' -var 'location=westus' -out "dev.tfplan"
```

- **Note:** The above command will create new file as dev.tfplan
- Now to deploy changes we need to run below command

```
CMD: terraform apply "dev.tfplan"
```

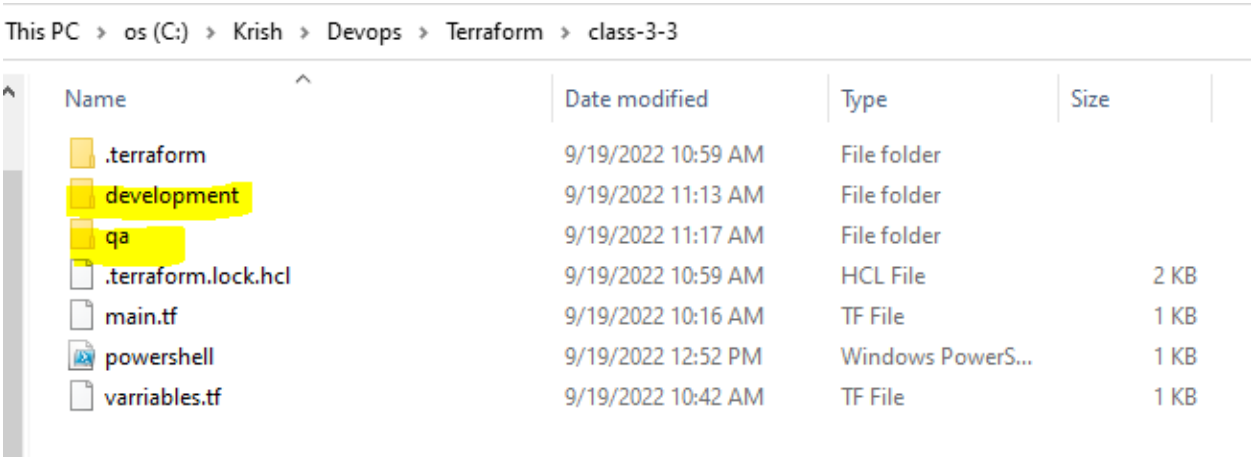
- Now we can deploy storage account with change of variables

Note: If we use only “terraform apply” instead of ‘terraform apply “dev.tfplan”’. It could not deploy change in variable it will go with default. So we need to use dev.tfplan to deplay change in variables

<u>Creating Multiple Storage accounts with their own resource groups:</u>

Requirement: we need to create 2 storage accounts with 2 resource groups.

- Create new folder and sub folders as below



- Now open VS code and create Variable file as below

```
variable "resourceGroupName" {
  type = string

}

variable "location" {
  type = string

}

variable "storageaccountName" {
  type = string

}

variable "sku" {
  type = string

}
```

- Now write below template for storage account in main file

```
provider "azurerm" {
  features {

  }
}

resource "azurerm_resource_group" "rg" {
  name     = var.resourceGroupName
  location = var.location
}

resource "azurerm_storage_account" "storage" {
  name                     = var.storageaccountName
  resource_group_name      = azurerm_resource_group.rg.name
  location                 = azurerm_resource_group.rg.location
  account_tier             = "Standard"
  account_replication_type = var.sku
}
```

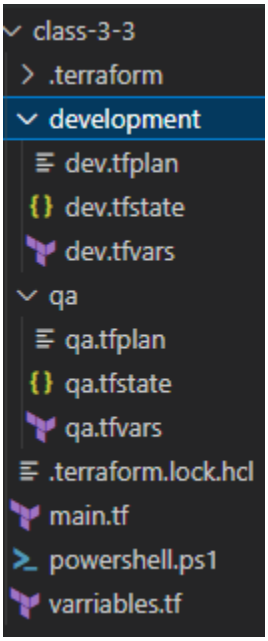
- Now we need to create new files as dynamic variable files for development and qa
- Now under development folder create a file as “dev.tfvars” write code as below

```
resourceGroupName = "dev-rg"
location = "West US2"
storageaccountName = "devkrishstore321"
sku = "LRS"
```

- Now under qa folder create a file as “qa.tfvars” write code as below

```
resourceGroupName = "qa-rg"
location = "West US"
storageaccountName = "qakrishstore321"
sku = "LRS"
```

- Directory in VS code



Deploying 2 environments using above templates:

- Now we need to run Below commands

```
powershell.ps1 class-3 ●   powershell.ps1 class-3-3 ×
class-3-3 > powershell.ps1
1   #creating storage accounts and resource groups multiple teams
2
3   terraform init
4
5   terraform validate
6
7   terraform fmt
8
9   #Deploying script for development
10
11  terraform plan -var-file="./development/dev.tfvars" -out="./development/dev.tfplan" -state="./development/dev.tfstate"
12
13  terraform apply -state-out="./development/dev.tfstate" "./development/dev.tfplan"
14
15  #Deploying script for qa
16
17  terraform plan -var-file="./qa/qa.tfvars" -out="./qa/qa.tfplan" -state="./qa/qa.tfstate"
18
19  terraform apply -state-out="./qa/qa.tfstate" "./qa/qa.tfplan"
20
```

⇨ Above commands will deploy both environments using template.

Creating Reusable Template for Storage Account:

- Earlier in ARM template we have used CONCAT for merging names as one. So that could give one name to. That can also reusable.
- In Terraform also we can use same, but no need to use concat but we can do by ${var.ex}, please check below how it works.
    - Exp: variable : exp = “abc”
        - ${var.exp}-krish = abc-krish
- Now we need to create class-4 folder and in that 2 sub folders as
    1. development
    2. qa

- Now go to VS code create below files in class-4 folder
  1. Main (script)
  2. Variable
- Now write Variables as below

variable.tf

Class-4 > variable.tf > variable "sku"

```
variable "location" {
  type = string
}

variable "sku" {
  type = string
}

variable "teamname" {
  type = string
}

variable "regioncode" {
  type = string
}
```

- Now create new file as dev.tfvars under development folder, write below code in that

dev.tfvars

Class-4 > development > dev.tfvars >

```
teamname = "dev"
location = "West US2"
regioncode = "westeus2"
sku = "LRS"
```

- Now create new file as qa.rfvars under qa folder, write below code in that

qa.tfvars

Class-4 > qa > qa.tfvars > sku

```
teamname = "qa"
location = "east US2"
regioncode = "eastus2"
sku = "GRS"
```

- Now write below code in main file

main.tf

Class-4 > main.tf > provider "azurerm"

```
provider "azurerm" {
  features {

  }
}

resource "azurerm_resource_group" "rg" {
  name     = "${var.teamname}-rg"
  location = var.location
}

resource "azurerm_storage_account" "storage" {
  name                     = "${var.teamname}krish00${var.regioncode}"
  resource_group_name      = azurerm_resource_group.rg.name
  location                 = azurerm_resource_group.rg.location
  account_tier             = "Standard"
  account_replication_type = var.sku
}
```

- Now run below commands one by one to deploy the environments.

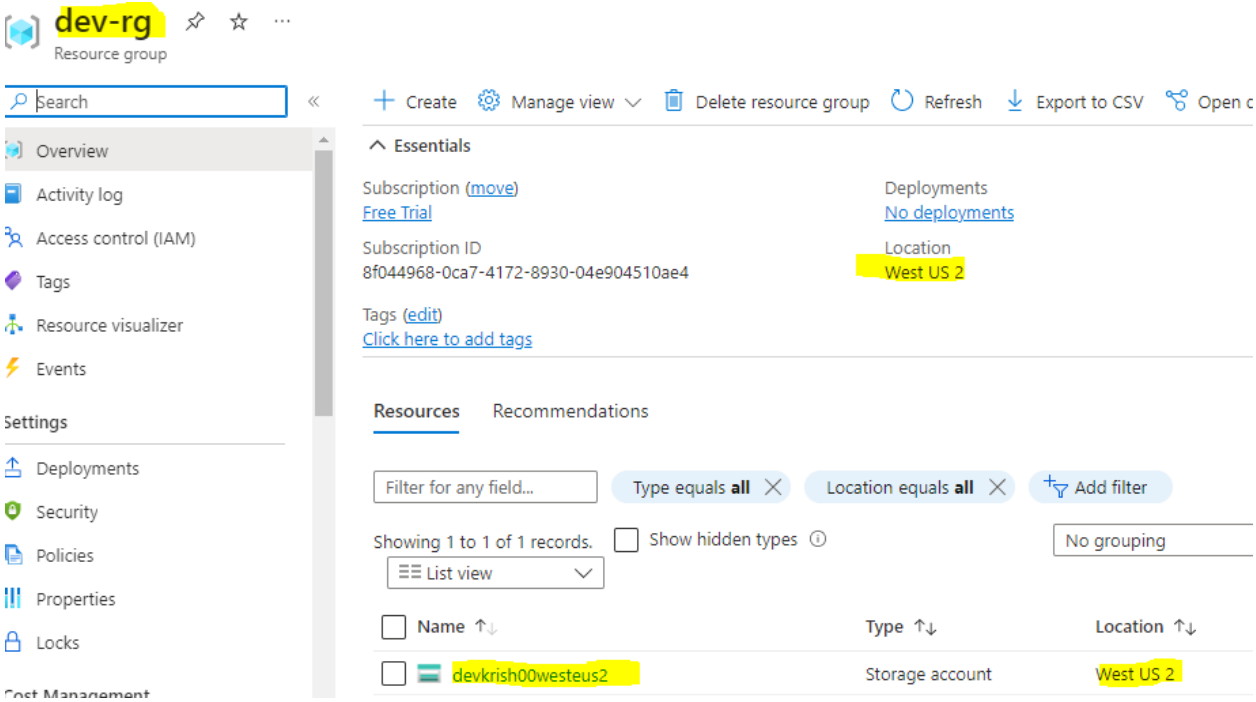
powershell.ps1

Class-4 > powershell.ps1

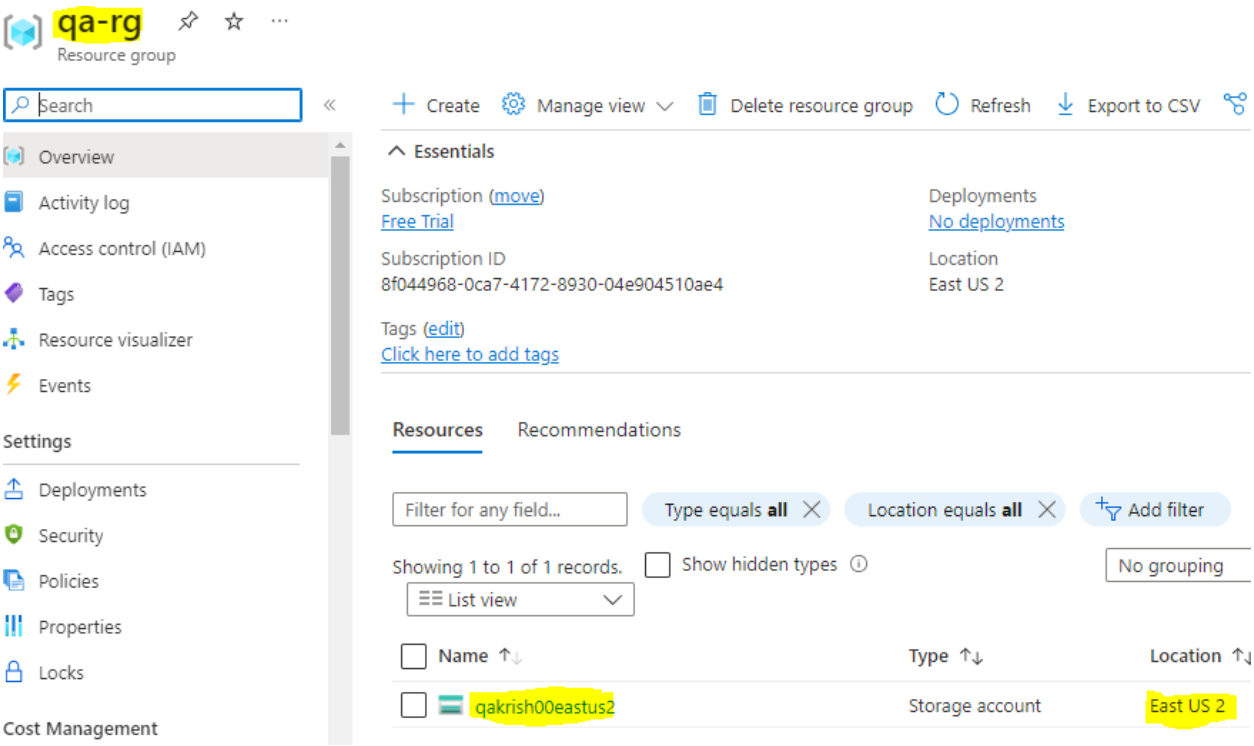
```
#creating storage accounts and resource groups multiple teams

terraform init

terraform validate

terraform fmt

#Deploying script for development

terraform plan -var-file="./development/dev.tfvars" -out="./development/dev.tfplan" -state="./development/dev.tfstate"

terraform apply -state-out="./development/dev.tfstate" "./development/dev.tfplan"

#Deploying script for qa

terraform plan -var-file="./qa/qa.tfvars" -out="./qa/qa.tfplan" -state="./qa/qa.tfstate"

terraform apply -state-out="./qa/qa.tfstate" "./qa/qa.tfplan"
```

- Now see below how they deployed

- Below is the dev team



- Now below QA team



- Now we can see the storage accounts we didn't declare completely anywhere but we got name with some combinations. Because we have used merging names.

Resource Iteration:

Earlier we have crated one storage account under one resource group. Now our requirement is to create multiple storage accounts under one resource group in incremental manner.

- While in ARM we have used the 'Copy Index' to do incremental now in terraform we can use 'Count Index'.
- We need to use reference as 'Data Source' while creating resources.
- Now go to Azure portal and create new resource group as 'demo-rg'.

- Now create new folder name it Class-4-2
- Create variables.tf file and write below

variable.tf

Class-4-2 > variable.tf > ...

```
variable "storageaccountname" {
  type    = string
  default = "demostorekrish00"
}

variable "sku" {
  type    = string
  default = "LRS"
}
```

- The name we have specified above that will be storage account name

- Now create main.tf file and write below

main.tf

Class-4-2 > main.tf > ...

```
provider "azurerm" {
  features {

  }
}

# data source
#Note: demo-rg resource group already created

data "azurerm_resource_group" "local" {
  name = "demo-rg"
}

#Multimple storage accounts

resource "azurerm_storage_account" "localstorage" {

  count                    = 3
  name                     = "${var.storageaccountname}${count.index}"
  resource_group_name      = data.azurerm_resource_group.local.name
  location                 = data.azurerm_resource_group.local.location
  account_tier             = "Standard"
  account_replication_type = var.sku

}
#now it will deploy below storage accounts
#demostorekrish000
#demostorekrish001
#demostorekrish002
```

- Note:
  - ⇨ 'demo-rg' we have already created as we shown screenshot earlier. Here we are deploying storage accounts in 'demo-rg' using data source function. Look at below in data we have specified the 'demo-rg'

```
# data source
#Note: demo-rg resource group already created

data "azurerm_resource_group" "local" {
  name = "demo-rg"
}
```

- ⇨ As specified below ‘count’ function it will create the resources in incremental manner

```
resource "azurerm_storage_account" "localstorage" {

  count                    = 3
  name                     = "${var.storageaccountname}${count.index}"
```

- ⇨ The count.index will deploy the storage account names as below

```
#demostorekrish000
#demostorekrish001
#demostorekrish002
```

- Save the file and run below commands to deploy the code.

powershell.ps1 ●

lass-4-2 > powershell.ps1

```
#Creating moultiple Storage accounts
#existing resoucegroup using data source and count

terraform init

terraform validate

terraform fmt

terraform plan

terraform apply --auto-approve
```

- After running above commands storage accounts will crated as below

demostorekrish000

demostorekrish001

demostorekrish002