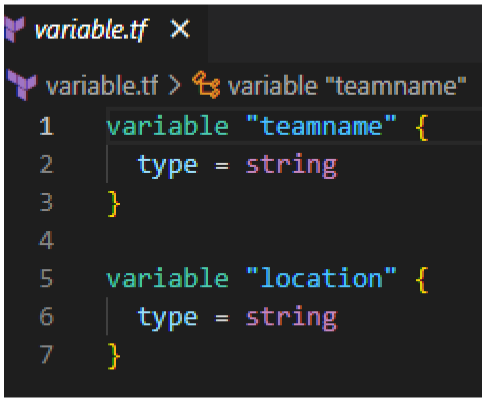
TERRAFORM DEVOPS

- We need to ensure that no one should touch or edit our state file
- We need to store our 'state' file in Storage account, with restricted access. So no one will touch or edit the file.
- **Note:** 'Backend' function only used for storage accounts.
- Now we need created new folder as 'Class-5'
- Create new file under that folder as "variables.tf" write as below



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

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

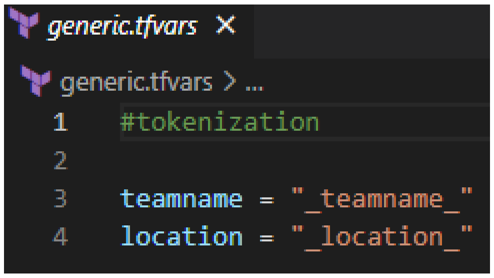
- Now go to Azure Portal and Create Below resources
    - Resource Group = demo-rg
    - Storage account name = demostorekrish000
    - Container Name = statecontianer
- We have created above resources one under one
- Now we need to store our state file in our "statecontainer"
- To save state file in container we need to use Azure backend declaration as below code. Even we have given our resource reference in backend.

- Now create “main.tf” write below code.

```
provider "azurerm" {
  feature{}
}

## Azure backend declaration

terraform {
  backend "azurerm" {
    resource_group_name  = "demo-rg"
    storage_account_name = "demostorekrish000"
    container_name       = "statecontainer"
    #name of the state file
    key                  = "${var.teamname}.tfstate"
  }
}
# resources

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

resource "azurerm_service_plan" "appserviceplan" {
  name                = "${var.teamname}plan01"
  resource_group_name = azurerm_resource_group.localrg.name
  location            = azurerm_resource_group.localrg.location
  sku_name            = "F1"
  os_type             = "Windows"
}

resource "azurerm_windows_web_app" "appservice" {
  name                = "${var.teamname}webappkrish000"
  resource_group_name = azurerm_resource_group.localrg.name
  location            = azurerm_service_plan.appserviceplan.location
  service_plan_id     = azurerm_service_plan.appserviceplan.id

  site_config {}
}
```

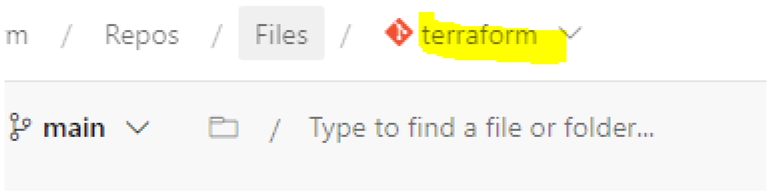
- **Note:** by using env.tfvars file we can deploy for one team, but if we want to deploy the script for multiple teams we need to create multiple .tfvars files but this this is really hard if we need many teams. So we need to create ‘generic.tfvars’ file, we can reuse this file as many time as we want. One file is enough.

- Now we need to create “generic.tfvars” file and code as below



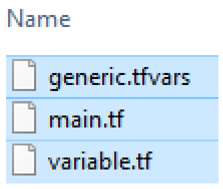
```
#tokenization

teamname = "_teamname_"
location = "_location_"
```

- Now go to Devops portal and create new project as “terraform”
- Click on below dropdown and create create new Repository as “infrascode”

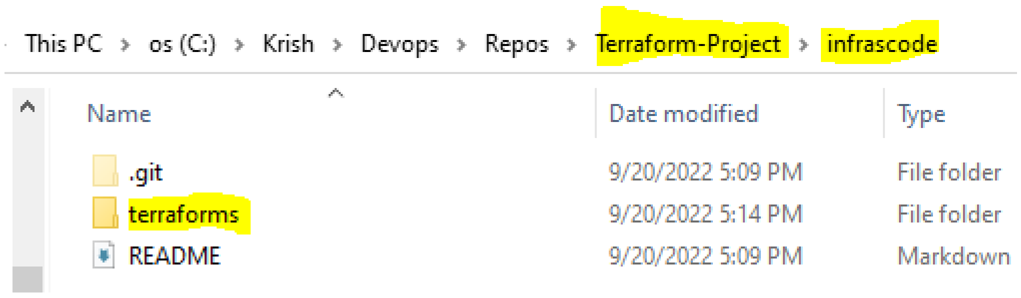


m / Repos / Files / terraform
main / Type to find a file or folder...

- Clone the project in VS code save as in Terraform-project folder.
- Now open file manager and copy below files from class-5 folder and paste in ‘Terraform project/infascode’ folder
    - main.tf
    - variable.tf
    - generic.tf



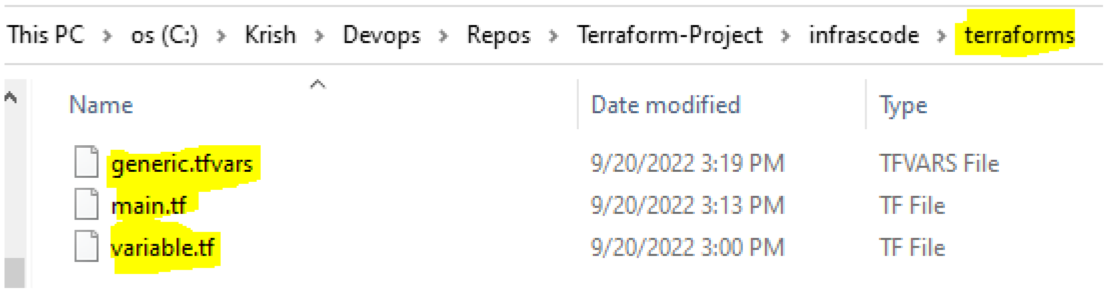
- Go to “infracode” and create new folder as “terraforms” as below



This PC › os (C:) › Krish › Devops › Repos › Terraform-Project › infrascode

Name	Date modified	Type
.git	9/20/2022 5:09 PM	File folder
terraforms	9/20/2022 5:14 PM	File folder
README	9/20/2022 5:09 PM	Markdown

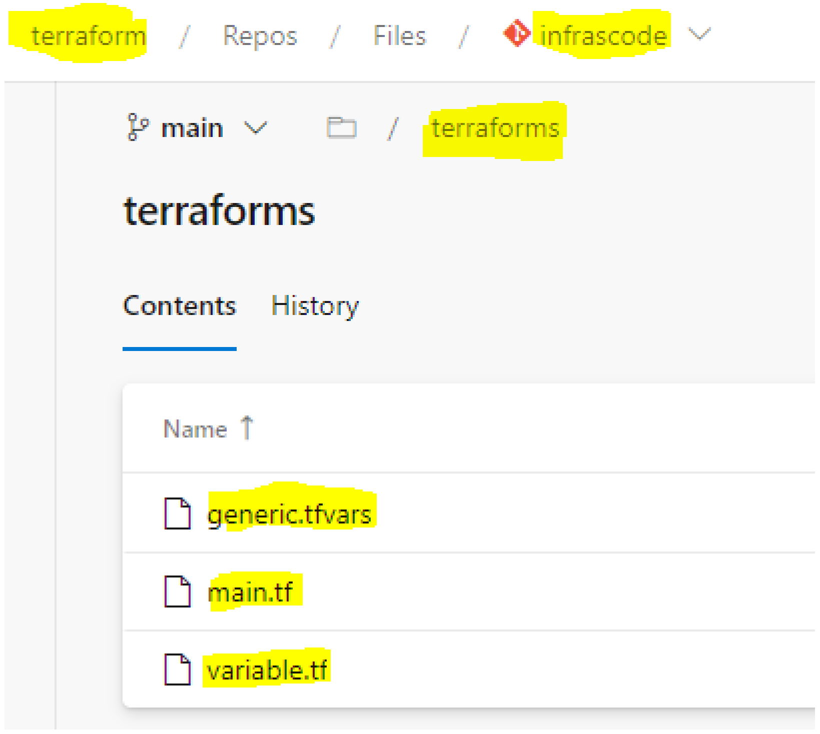
- Now paste above files in the terraforms folder as below



This PC > os (C:) > Krish > Devops > Repos > Terraform-Project > infrascode > terraforms

Name	Date modified	Type
generic.tfvars	9/20/2022 3:19 PM	TFVARS File
main.tf	9/20/2022 3:13 PM	TF File
variable.tf	9/20/2022 3:00 PM	TF File

- Now go to VS code commit the changes and sync or publish to Remote repository.
- Now go to Devops portal and see that terraforms folder has uploaded.



terraform / Repos / Files / infrascode

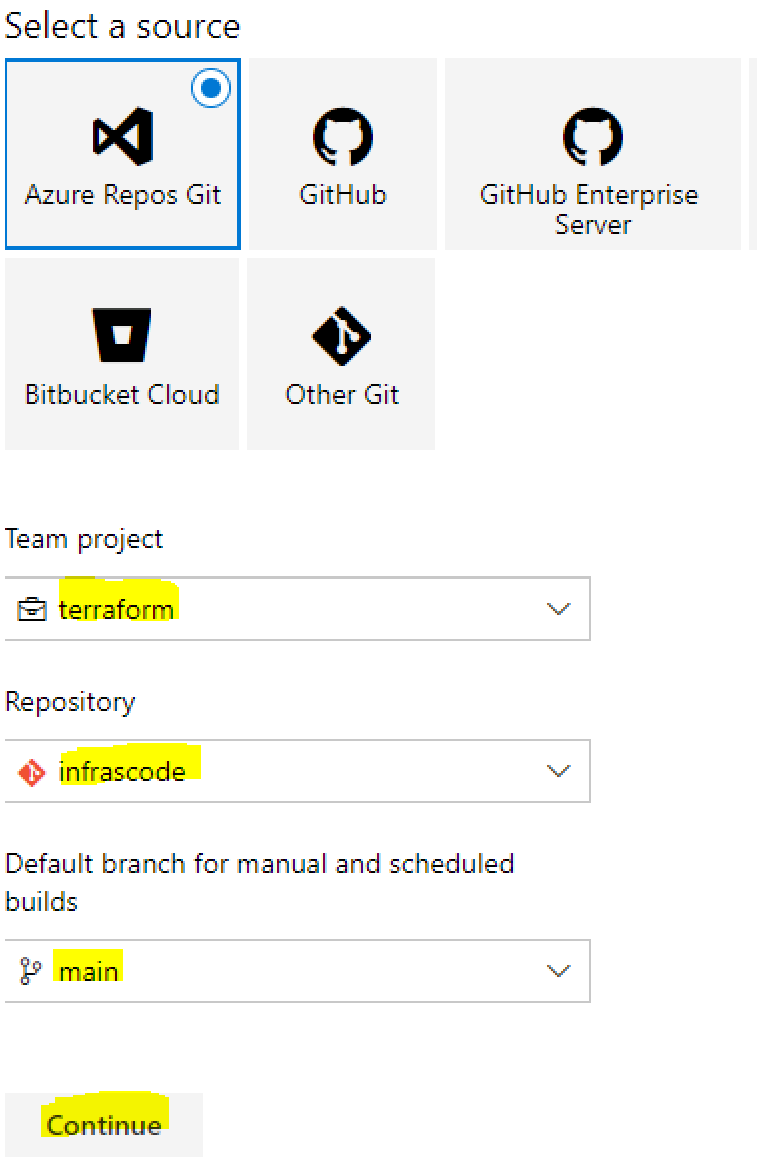
main / terraforms

terraforms

Contents History

Name ↑
generic.tfvars
main.tf
variable.tf

- Now go to Pipeline and create new pipeline using classic editor



Select a source

Team project

terraform

Repository

infrascode

Default branch for manual and scheduled builds

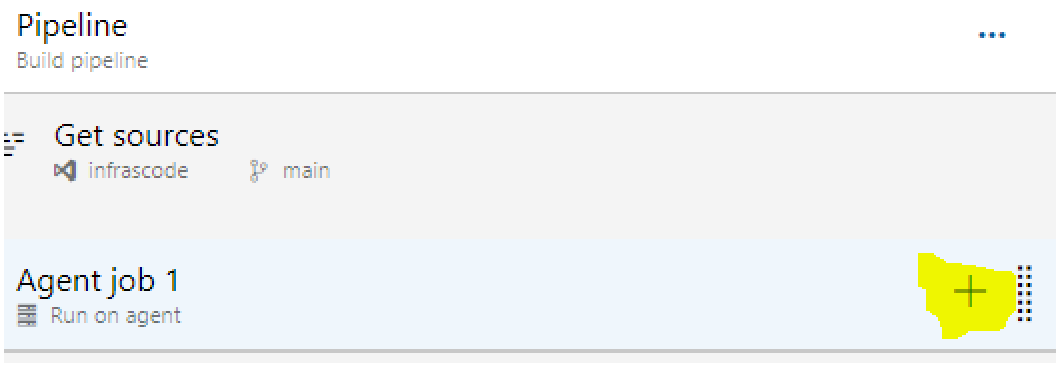
main

⇨ Click on continue

⇨ Select empty job

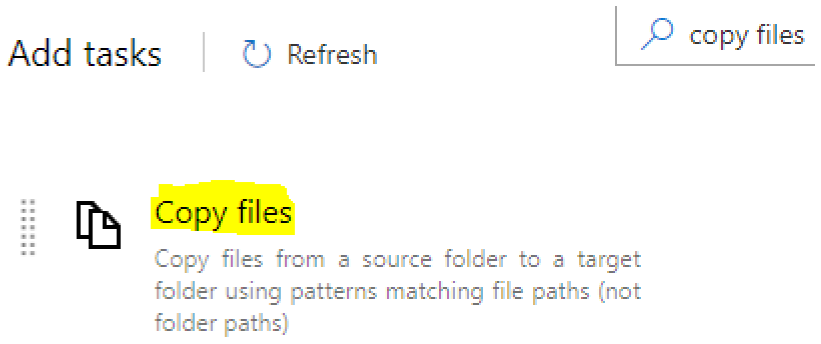
⇨ Click on pipe line and change name to “Terraform-pipeline”

⇨ Click on add task, as below



Pipeline
Build pipeline
Get sources
infrascode main
Agent job 1
Run on agent

⇨ Search as "Copy files" select below and click on add



Add tasks | Refresh
copy files

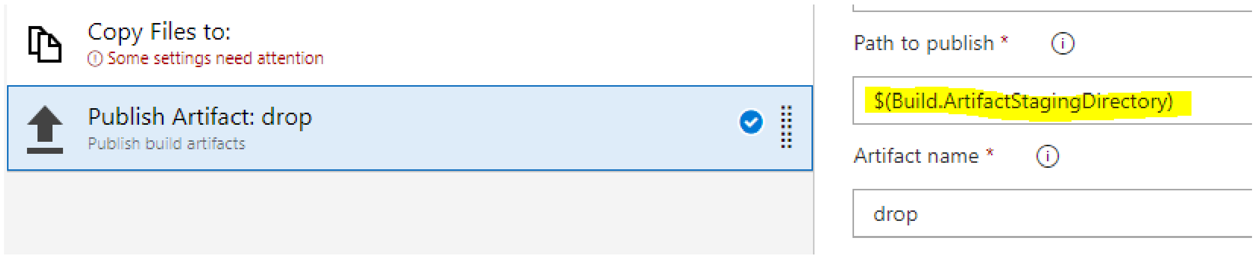
Copy files
Copy files from a source folder to a target folder using patterns matching file paths (not folder paths)

⇨ Now add one more task as "Publish build artifacts"

⇨ Go to copy files task

⇨ Click on source folder "…" -> brows and select terraforms folder

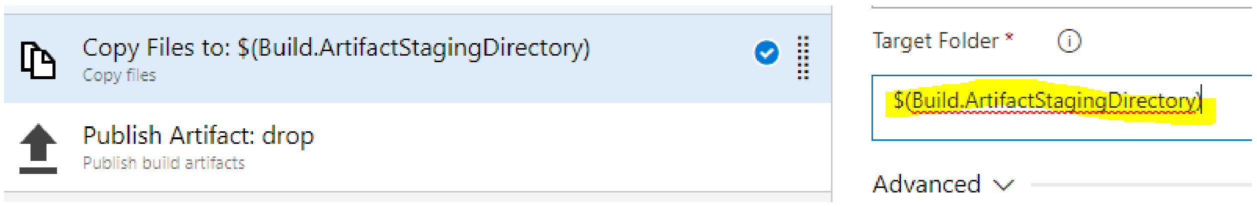
⇨ Now go to publish artifacts folder and copy "path to publish" as below



Copy Files to:
Some settings need attention
Publish Artifact: drop
Publish build artifacts

Path to publish *
$(Build.ArtifactStagingDirectory)
Artifact name *
drop

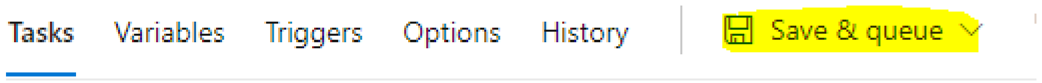
⇨ Now go to copy files task and paste in target folder



Copy Files to: $(Build.ArtifactStagingDirectory)
Copy files
Publish Artifact: drop
Publish build artifacts

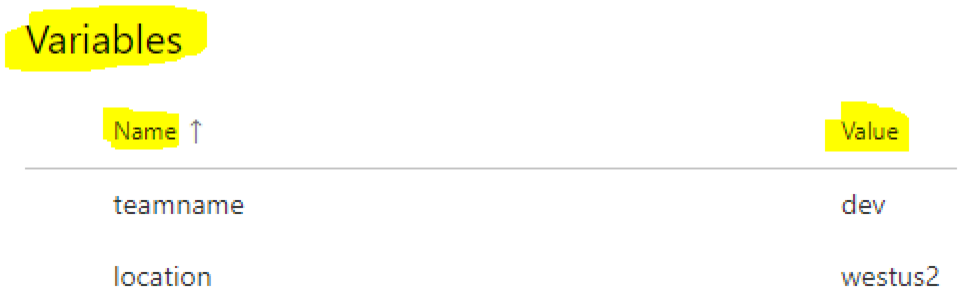
Target Folder *
$(Build.ArtifactStagingDirectory)
Advanced

⇨ Click on save & que as below and run



⇨ Now it will run the agent and create artifact folder

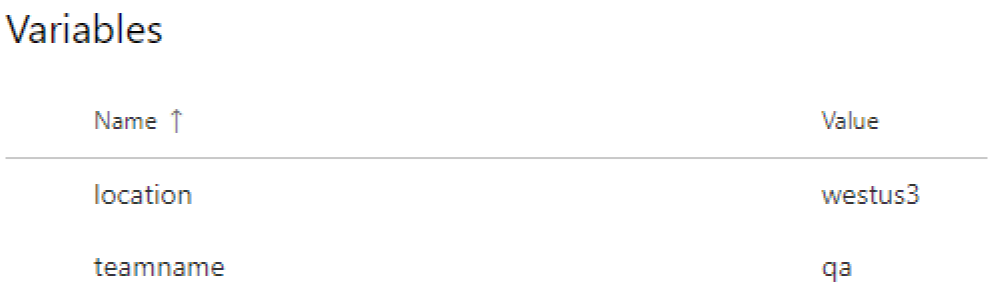
- Now go to Libraries in Pipeline
- Click on +Variables
- Give Variable group name as "development"
- Add variables as below



Variables

Name ↑	Value
teamname	dev
location	westus2

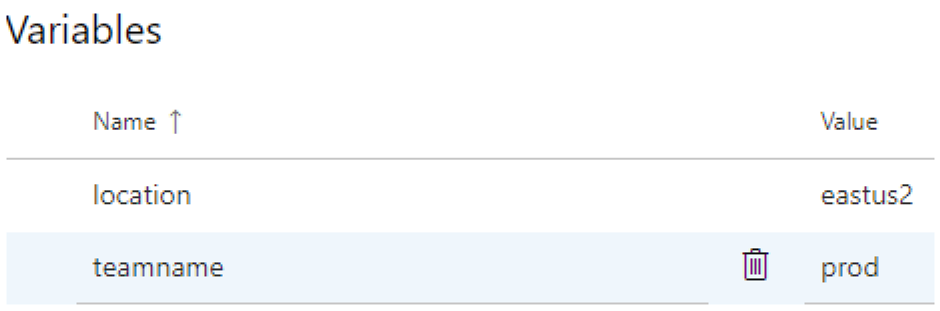
- Now create variable group for qa, name the group as "qa"



Variables

Name ↑	Value
location	westus3
teamname	qa

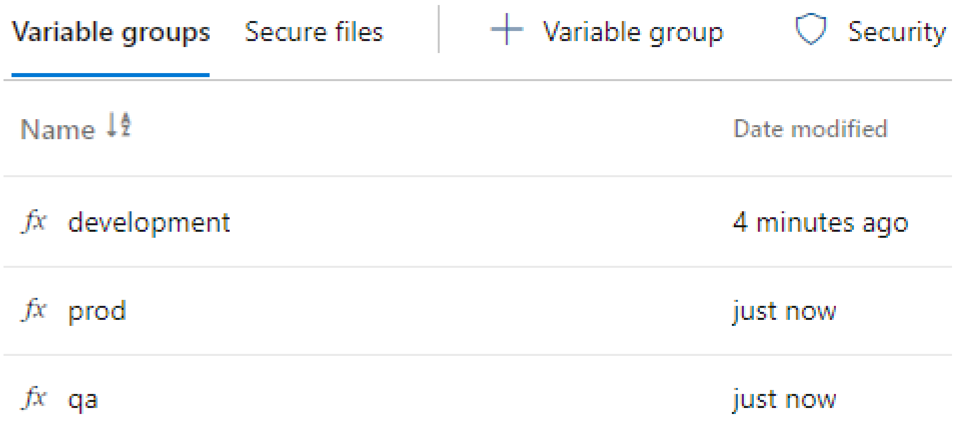
- Now create variable group for prod, name the group as "prod" as same as above



Variables

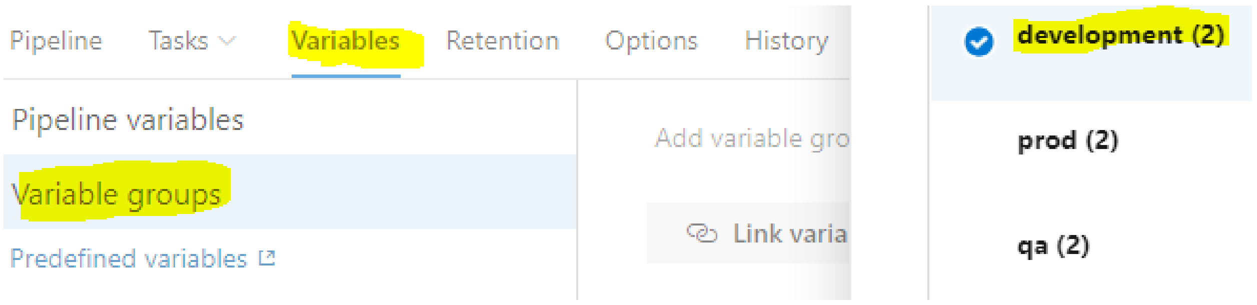
Name ↑	Value
location	eastus2
teamname	prod

- We have created below 3 variable groups

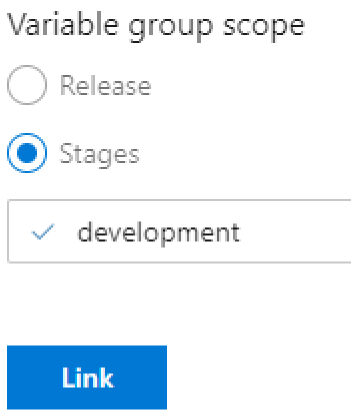

Variable groups | Secure files | + Variable group | Security

Name	Date modified
development	4 minutes ago
prod	just now
qa	just now

- Now go to release pipeline and create new release pipeline
- Close the pop up and click on add an artifact
- Source pipeline : select terraform pipeline and click on add
- Now add stage and name it as "development"
- Now go to variable group from top and link variable goup



-
- Now below variable group scope : select stage and from drop down select development


Variable group scope

Release [✓ Stages]

✓ development

- Now click on link, then it will link to our stage.
- Now go to Task bar on top and click on agent job select the pool and save
- Now click on add task and search for "replace token" add it
    - First time we need to install in other website by clicking on get it free
- Click on Root directory "…" select drop file and add

- Give target file as “generic.tfvars”
- Token pattern select: Custom
- Now give “_” as token prefix and suffix save
- Now add one more task as “terraform tool to installer” add it
- Select latest version
- Now add one more task and search as “Terraform” select and add
- Change display name as “terraform init”
- Provider: azurerm
- Set Configuration directory to “drop” by browsing

Now we need service connection for azure, if we don’t have connection, we need to create new, if we have existing one, we can use that by sharing it. To share existing connection to this project, follow below steps.

  - ⇨ Open other project which has connection in other tab
  - ⇨ Now go to project settings then service connection
  - ⇨ Select connection and click on 3 dots next to edit
  - ⇨ Click on security
  - ⇨ Click on + to add connection in project permissions
  - ⇨ Select our terraform project
- Now the connection is shared we can check terraform project service connections
- Now go to our release pipe line in other tab where we stopped
- Refresh Azure subscription and select our connection
- Now select “demo-rg” resource group as we created storage account and container in that earlier
- Select storage account and container
- Give key as “$(teamname).tfstate”
- Now add one more task search as “terraform” add it
- Change display name to “Terraform plan”
- Select command as “plan”
- Directory drop and select connection
- Additional command arguments : -var-file=.\generic.tfvars -out="$(teamname).tfplan" -state="$(teamname).tfstate"
- Now add one more task as “Terraform”
- Display name as : Terraform apply
- Change the command to : apply
- Select the connection, select the drop as directory
- Additional argument: "$(teamname).tfplan"
- Now save
- Clone development 2 times
- Change names as qa & prod
- Link them to variable groups and save

- Now create and run the pipe line to deploy the script
- Agent will run and deploy the web apps and resource groups for all 3 environments.

Important Notes:

- All most all the organizations secure their storage accounts by connecting Vnet to it. So that storage account and container will be secured, no one can touch or use until they are from storage account.
- Though they are from vnet there will be restriction whom to access who are not.
- When storage account restricted no one will touch our state files saved in container.
- Even Azure devops hosted agent will not access the container because it has secured with vnet.
- So all most all companies will use only Self Hosted Agent installed in VM in same Vnet.
- If the VM is another VM then we need to install the VPN in that VM so it can access to that container.
- No companies expose storage accounts to internet.