**Introduction**

Terraform is Infrastructure as Code (IaC) software tool developed by Hashicorp. It’s used to define and provision infrastructure with Hashicorp Configuration Language (HCL). Terraform can be used for deploying infrastructure in the below spaces:

* Azure Cloud
* AWS cloud
* Google Cloud

**Why do we need Terraform?**

Each cloud providers have their method/configuration language for deploying infrastructure, respectively. For Azure, we use JSON, AWS, and GCP as YAML.  Where terraform HCL is cross-platform used to deploy and manage infrastructure as code on any public cloud service. We just need to learn one code language (HCL), and it can be used for any public cloud. Terraform provides Free and paid plans. The free plan offers the majority of its features as free.

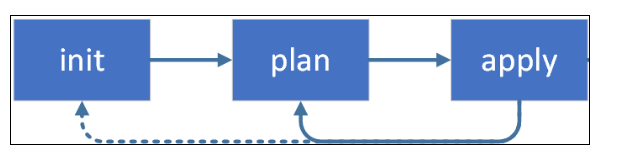
The major difference between Terraform and Azure Resource Manager Templates:

* Terraform uses HCL, whereas ARM used JSON language for defining the infrastructure.
* Terraform has a built-in auto dependency, where the ARM template needs to be explicitly called for dependency. This is required to define the resources which need to deploy in order and has some dependence on another resource.
* Terraform use of variables, in the ARM template, it's been referred to as parameters. The values from the user for deploying the template and defined as a separate file.
* Terraform has local variables, ARM templates named as variables. These values defined within the template itself.
* Terraform uses modules, in the ARM template called Nested template. This is required to call another template for deploying a particular resource.
* Terraform uses modules, in ARM template called Nested template. This is required to call another template for deploying a particular resource.

**Commands to deploy Terraform template:**

 To deploy the terraform file, follow the steps below with exact working directory folder:

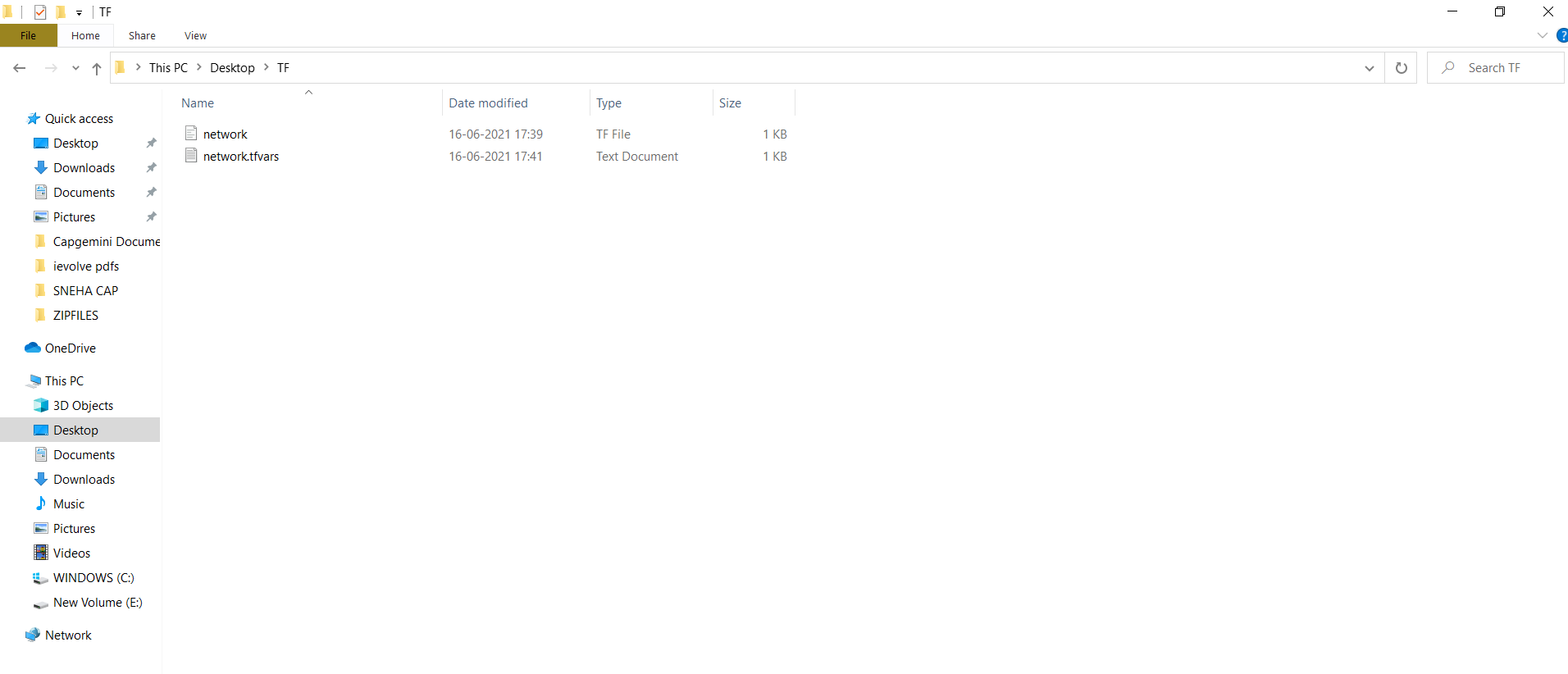
* **Terraform init** -- This command looks through all of the \*.tf files in the current working directory. It automatically downloads any of the providers (Azure or AWS) required for them to provision infrastructure.
* **Terraform plan** -- This command determines what actions are necessary to achieve the desired state specified in the configuration files. This is a dry run and shows which actions will be made.
* **Terraform apply** - auto-approve -- This will create resources as per config file and create Tfstate file which created on the above command, with auto-approval and not prompt for interactive approval for deploy.
* **Terraform destroy** -- This will destroy the configuration created based on the tfstate file.



**The things needed for  the demo:**

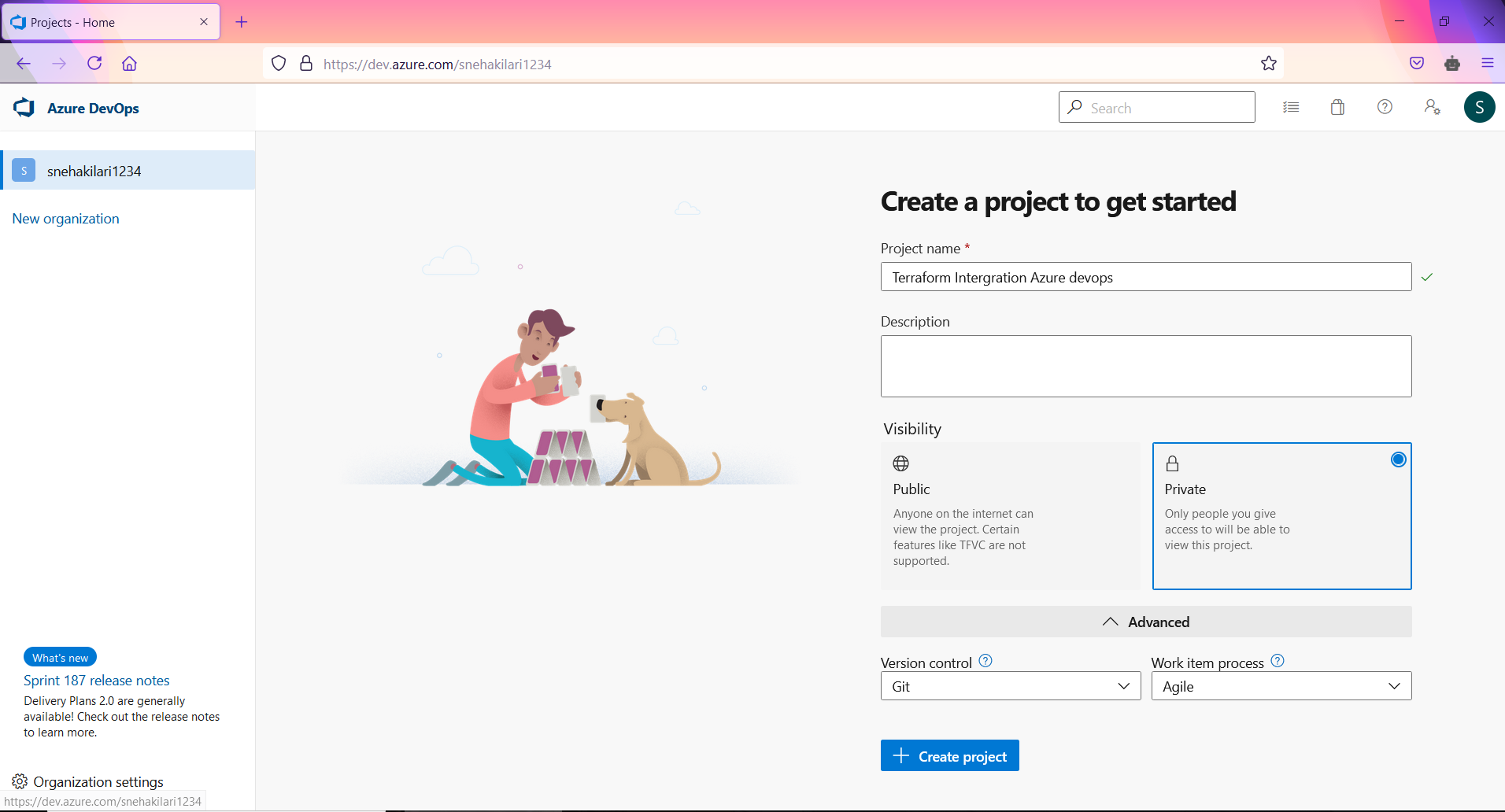
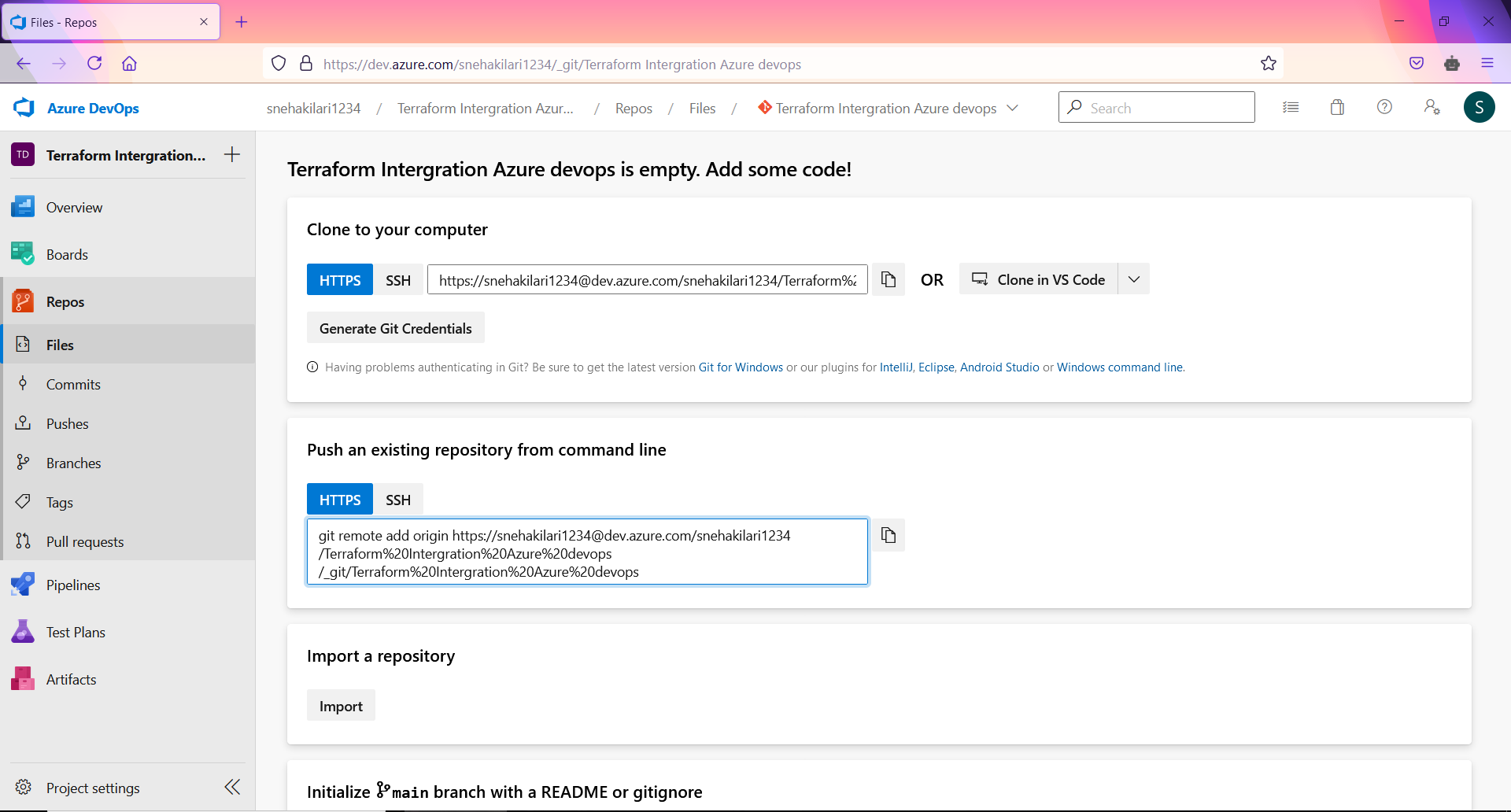
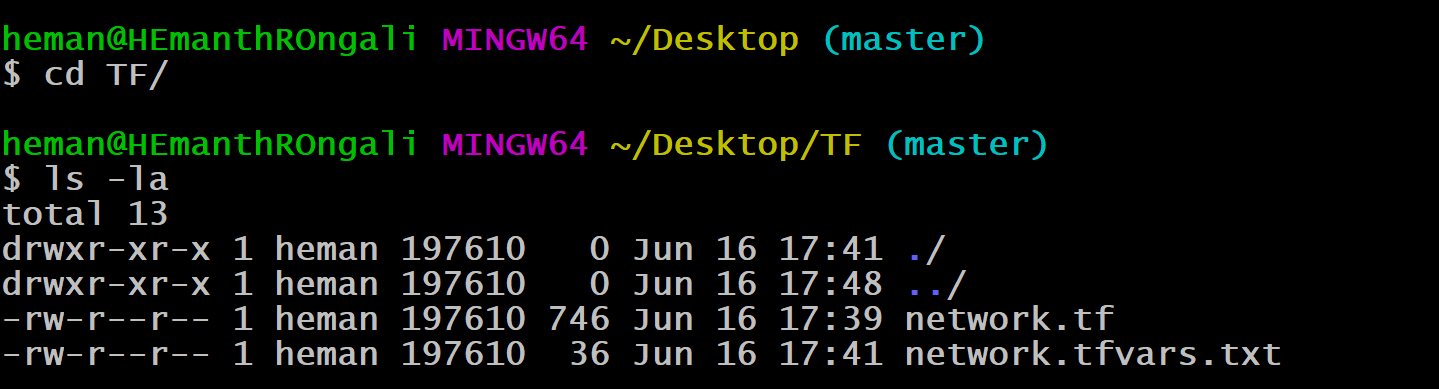
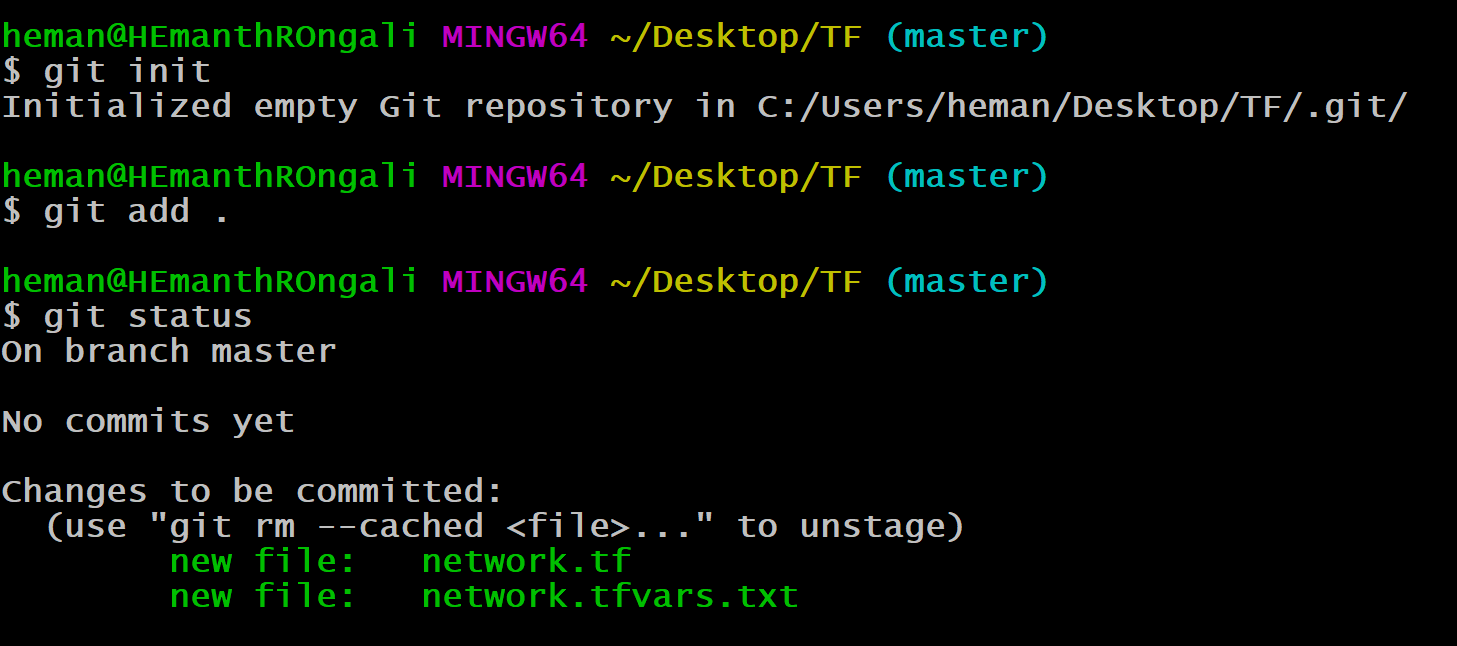
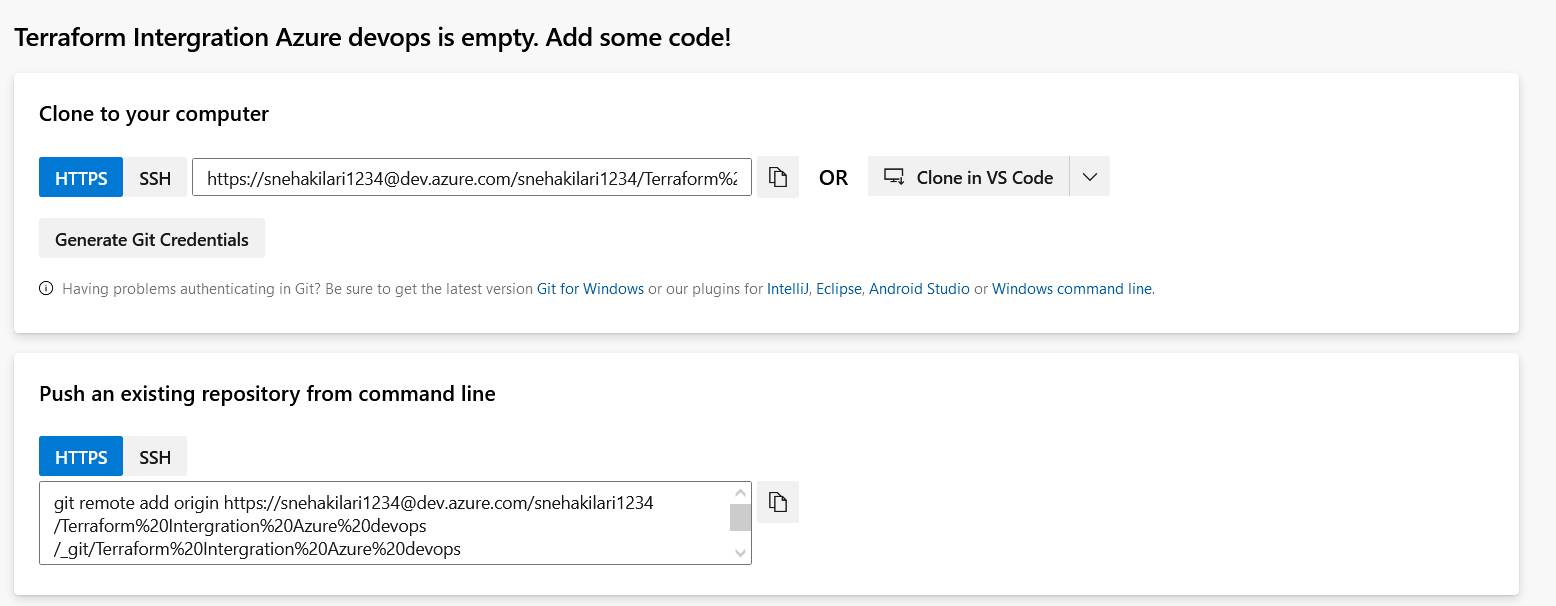
1. Visual Studio code software (Free edition)
2. Azure Subscription access. If not, you can create a free Azure account.
3. An account in Azure DevOps. If not, you can create a new account by log into https://visualstudio.microsoft.com/ and enable Azure DevOps service.  
     
   **Terraform File creation :**

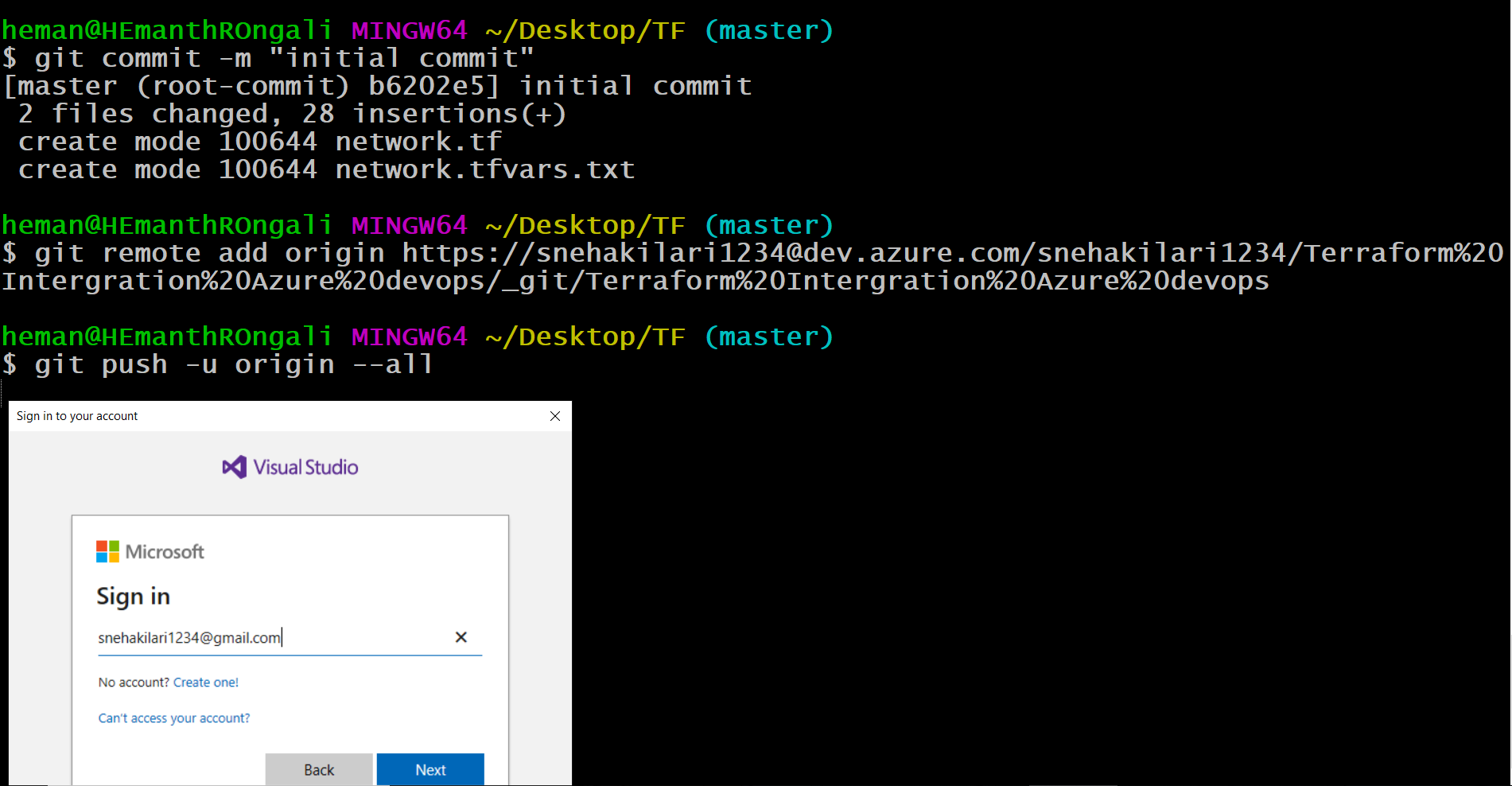
* I will use the Azure Virtual network. In my case terraform file called network.tf.    
    
  **# VARIABLES**
* **variable**
* **"resource\_group\_name" {**
* **type = string**
* **}**
* **provider "azurerm" {**
* **version = "=2.0.0"**
* **features {}**
* **}**
* **resource**
* **"azurerm\_resource\_group" "rg" {**
* **name = var.resource\_group\_name**
* **location = "Australia East"**
* **}**
* **# Create a virtual network within the resource group**
* **resource "azurerm\_virtual\_network" "terraform" {**
* **name = "terraform-network"**
* **resource\_group\_name = azurerm\_resource\_group.rg.name**
* **location = azurerm\_resource\_group.rg.location**
* **address\_space = ["10.10.0.0/24"]**
* **}**
* **resource "azurerm\_subnet" "app-subnet" {**
* **name = "appsubnet01"**
* **resource\_group\_name = azurerm\_resource\_group.rg.name**
* **virtual\_network\_name = azurerm\_virtual\_network.terraform.name**
* **address\_prefix = "10.10.0.0/25"**
* **}**
* Since we define the resource group as variable, we need to create a variable file with the name “network.tfvars”, and its content is as below   
    
  **:   resource\_group\_name = terraform-rg01**
* The above created two files should be placed in the same folder.

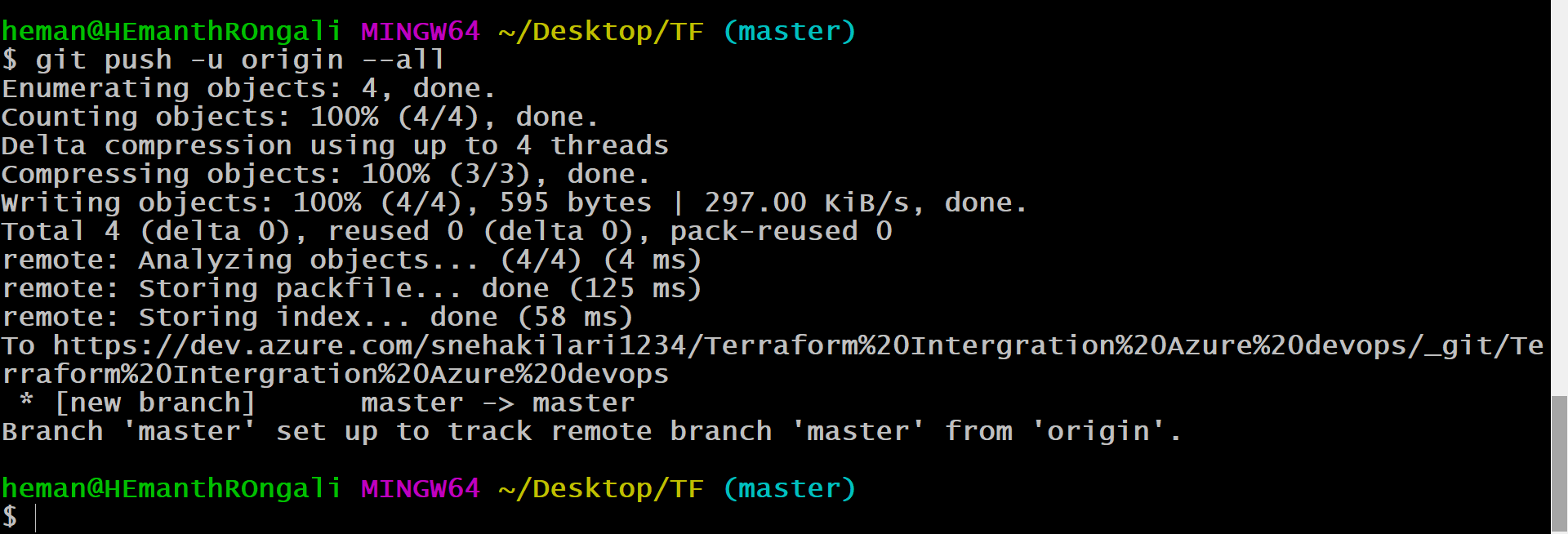


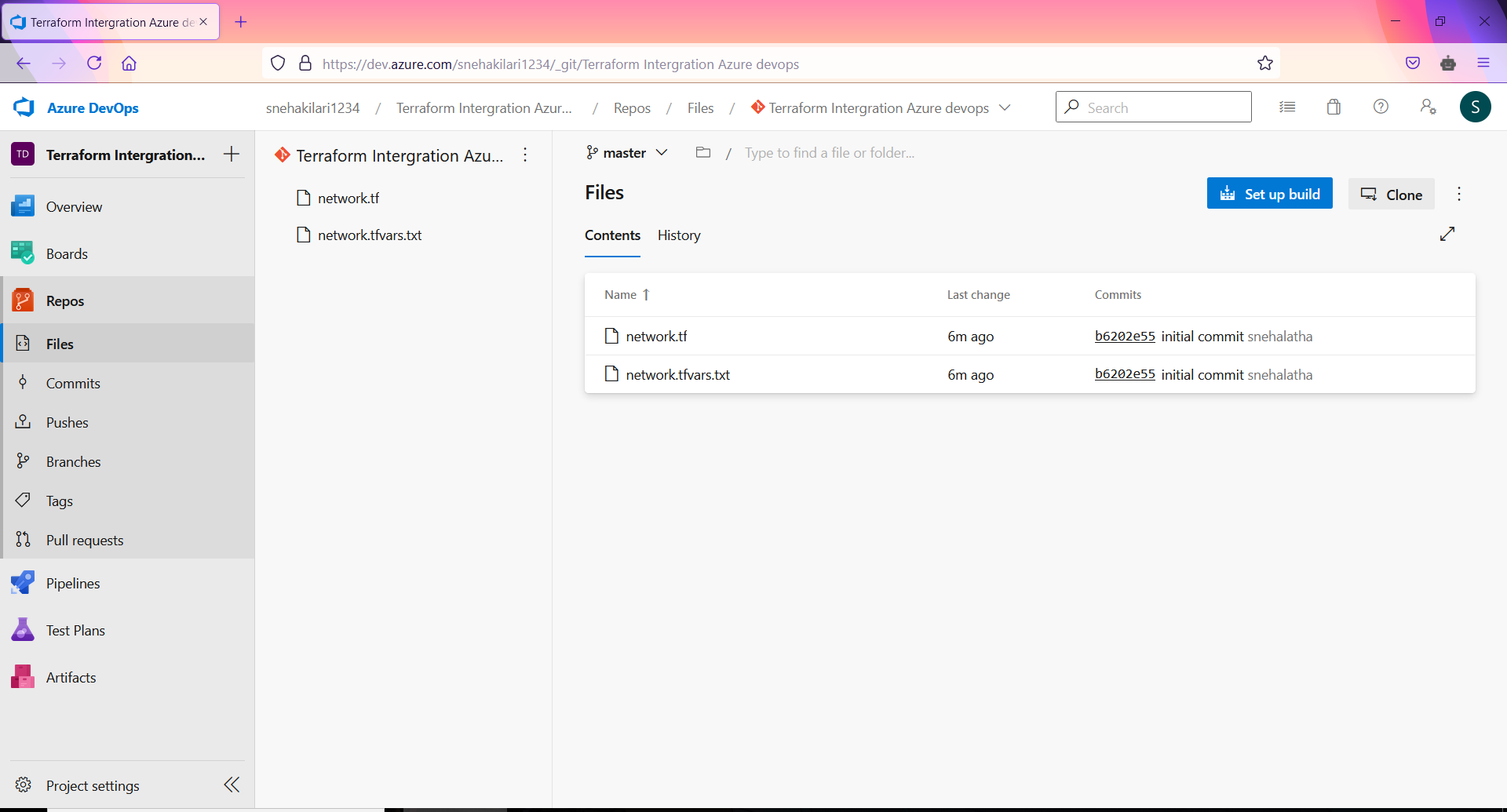
**Deploy resources using Terraform in Azure DevOps**

Follow the below steps with Azure DevOps and its pipelines.

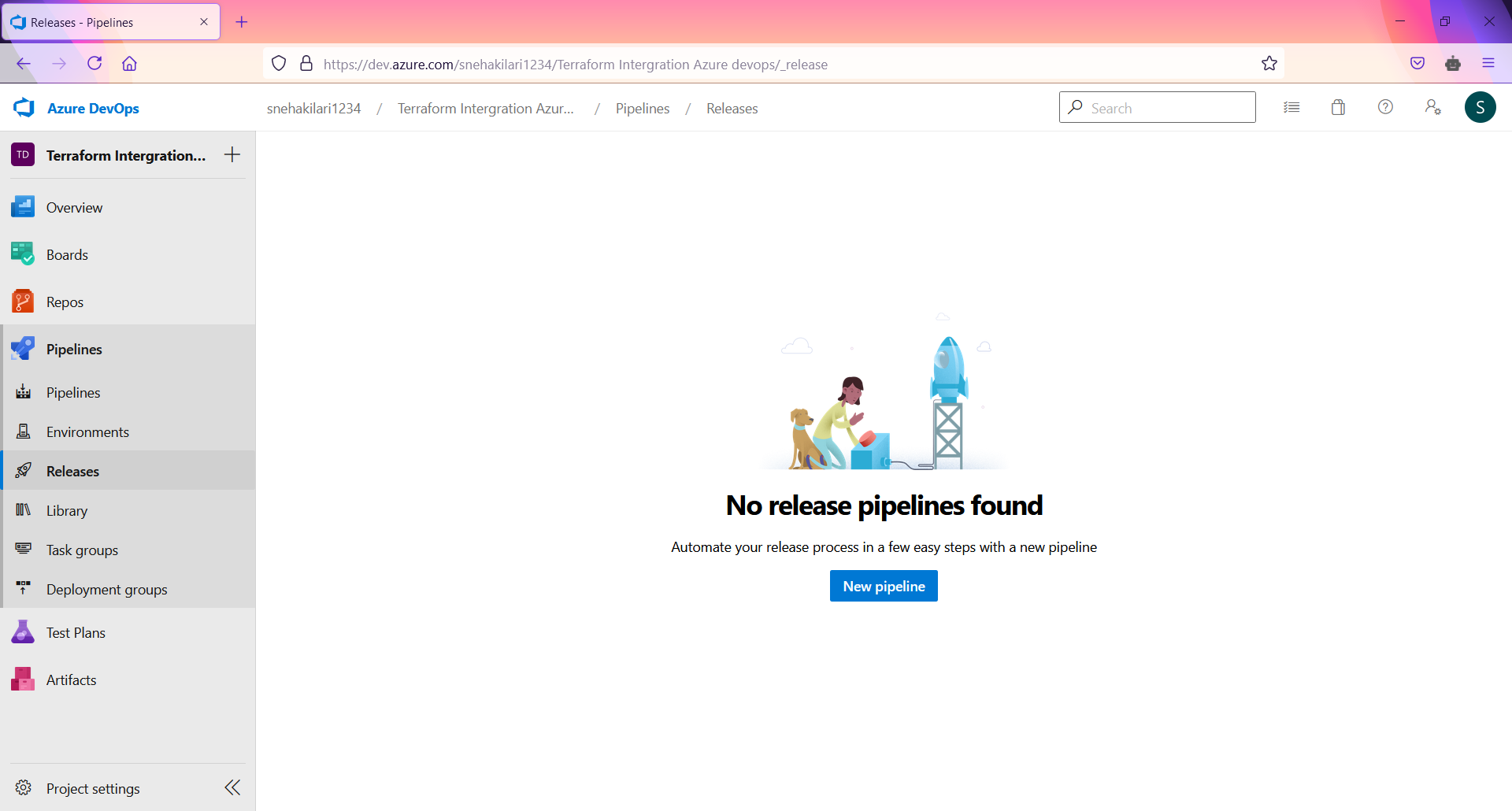
* Login into Azure DevOps and navigate to an existing project in Azure DevOps or create a new one.   
  
* Select Repos and click create a folder and upload the terraform file for deploying resources.   
    
    
    
  We are using “Git Bash” tool for Push my code from local to Azure repos.  
    
    
    
    
    
    
  

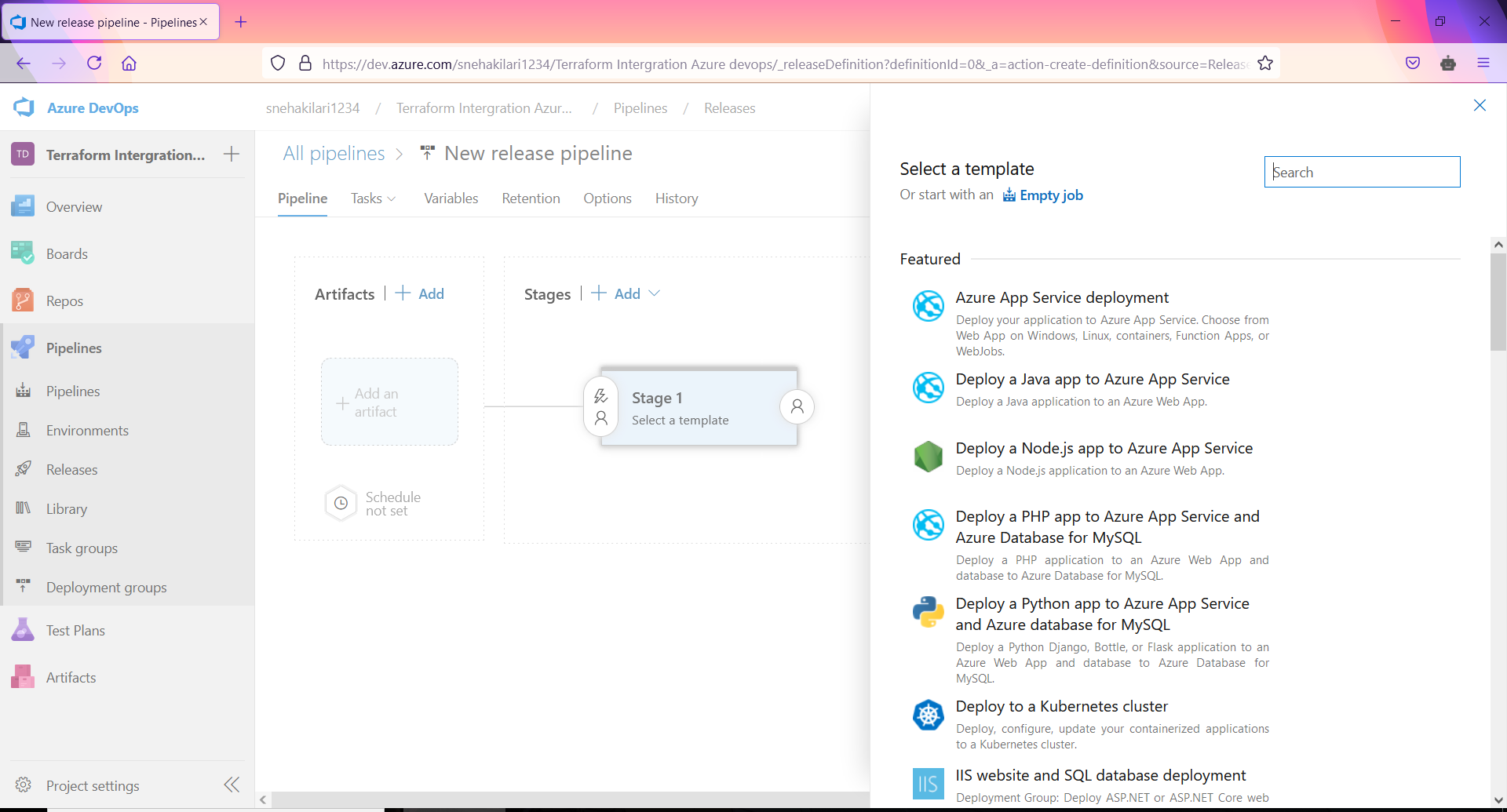




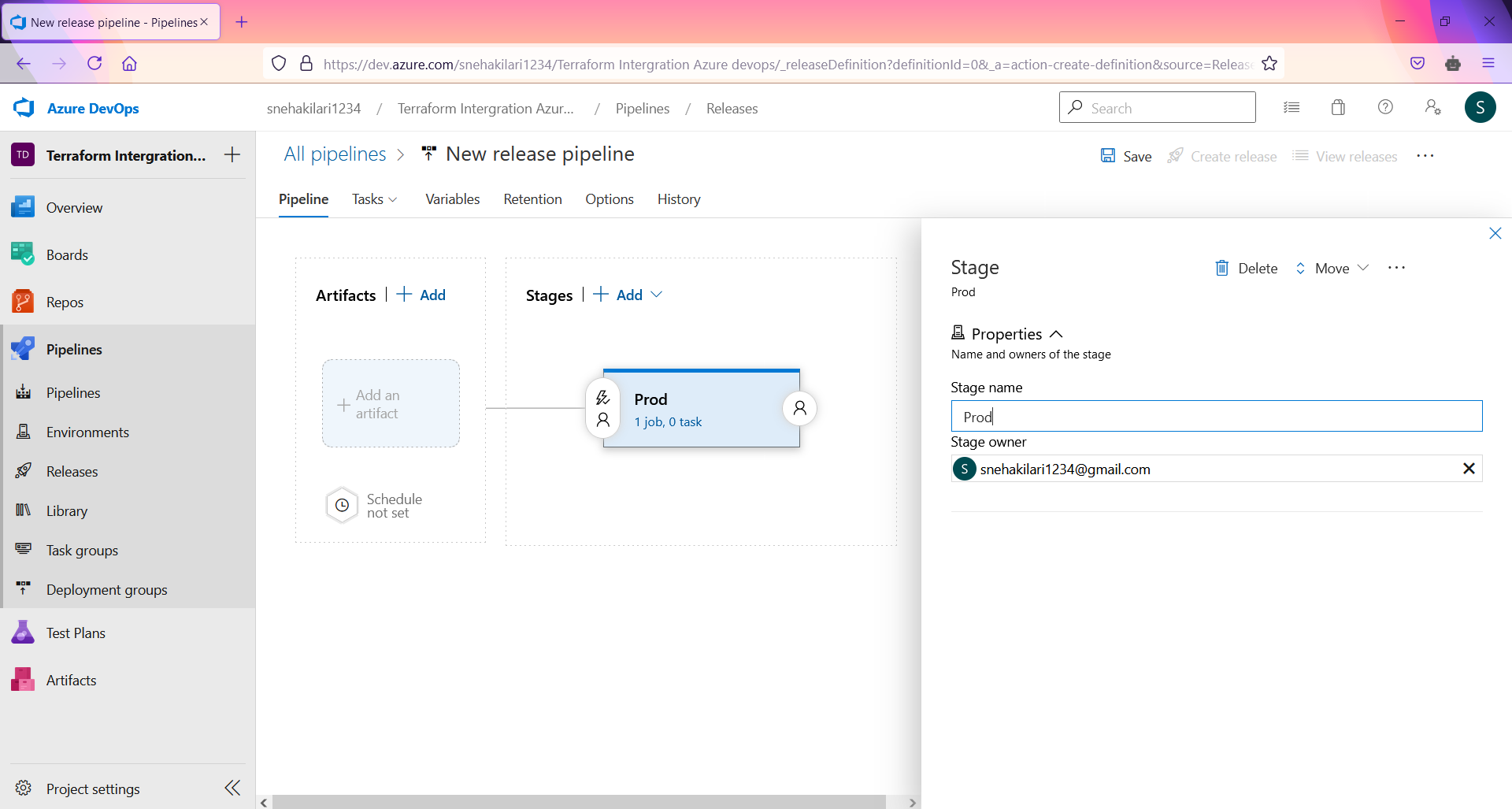
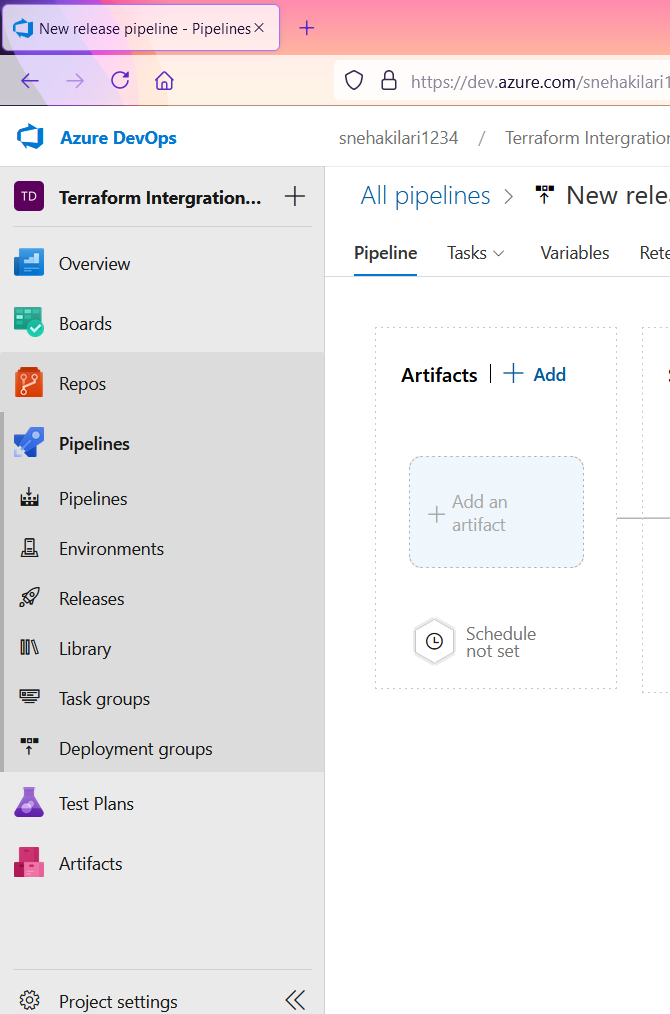
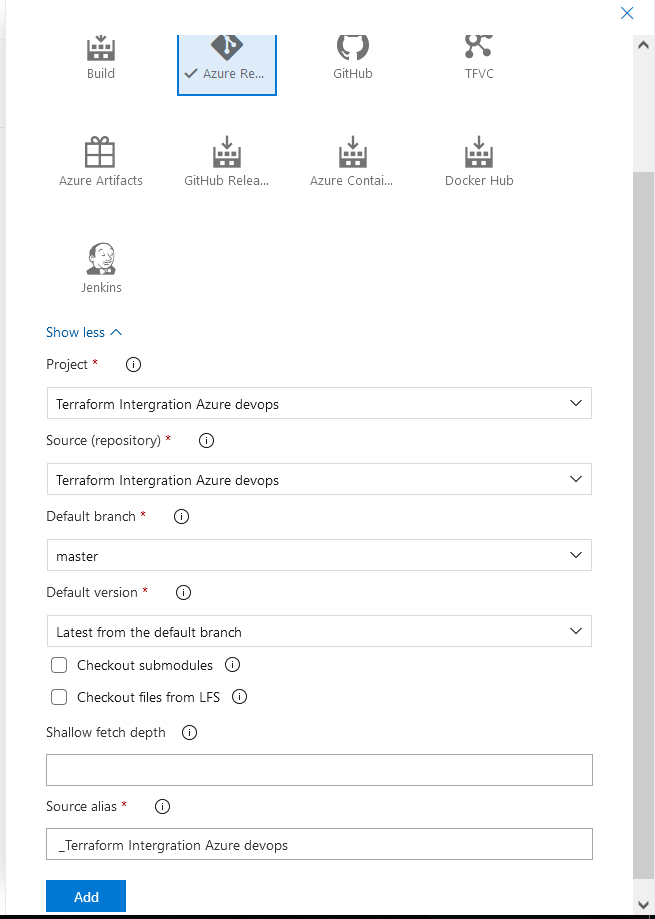
  
  
**Azure Pipelines :**

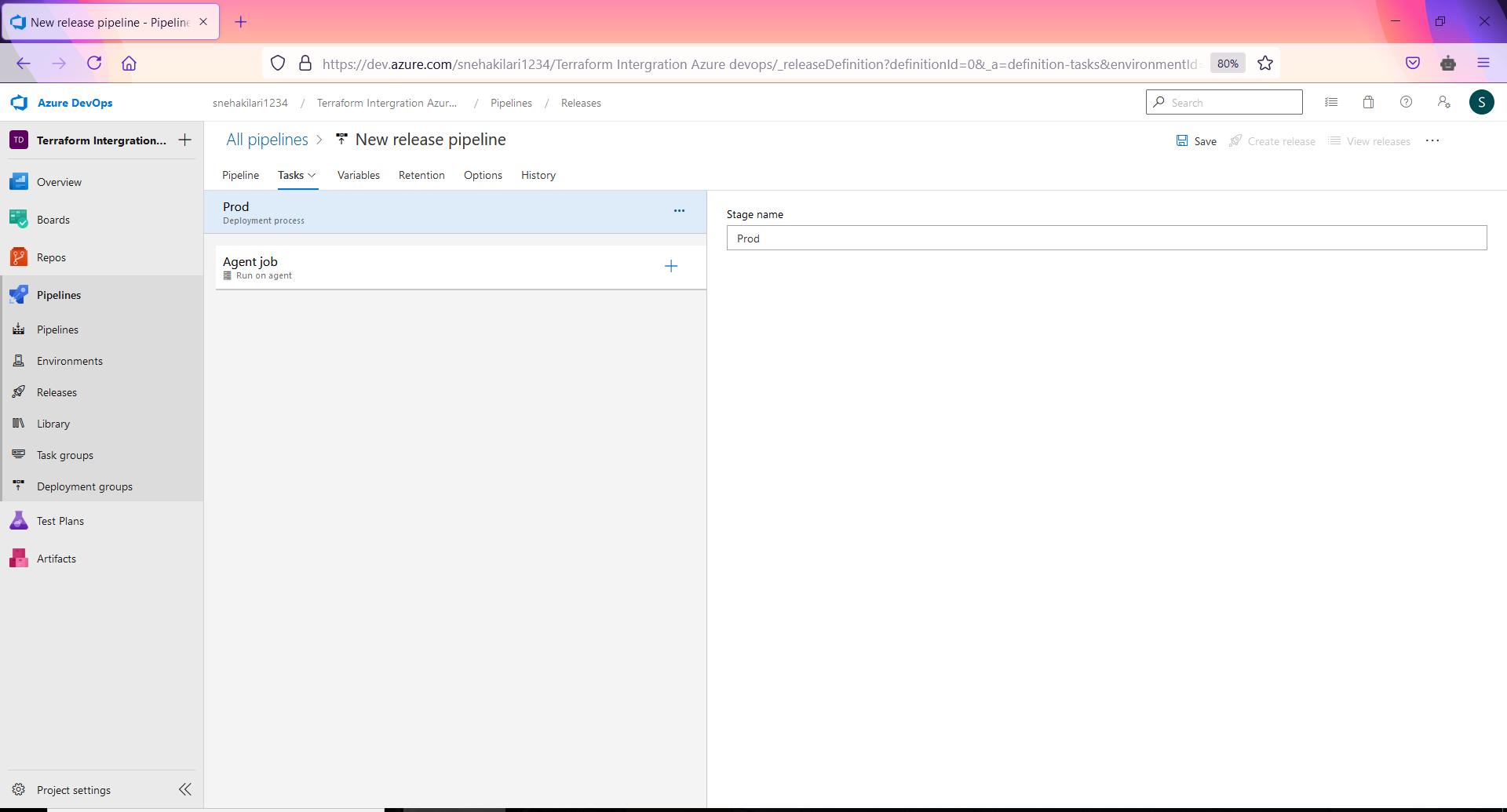
* Navigate to Pipelines –> Releases. Click New and select New release pipeline and click empty job.

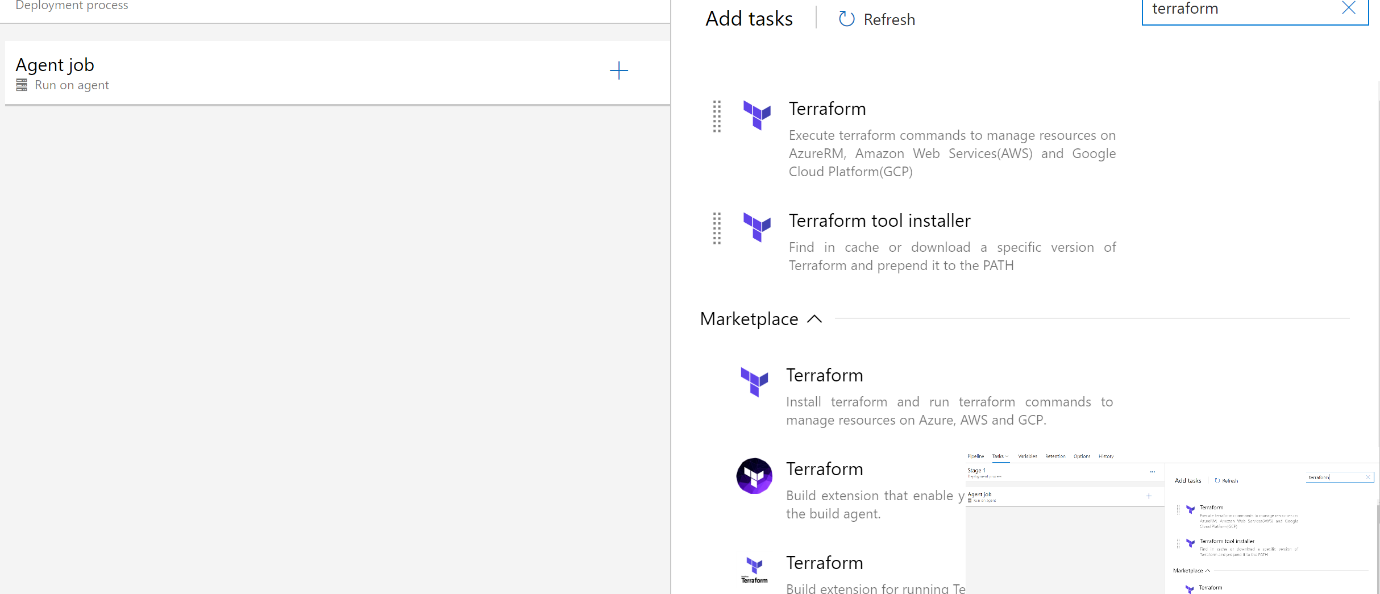
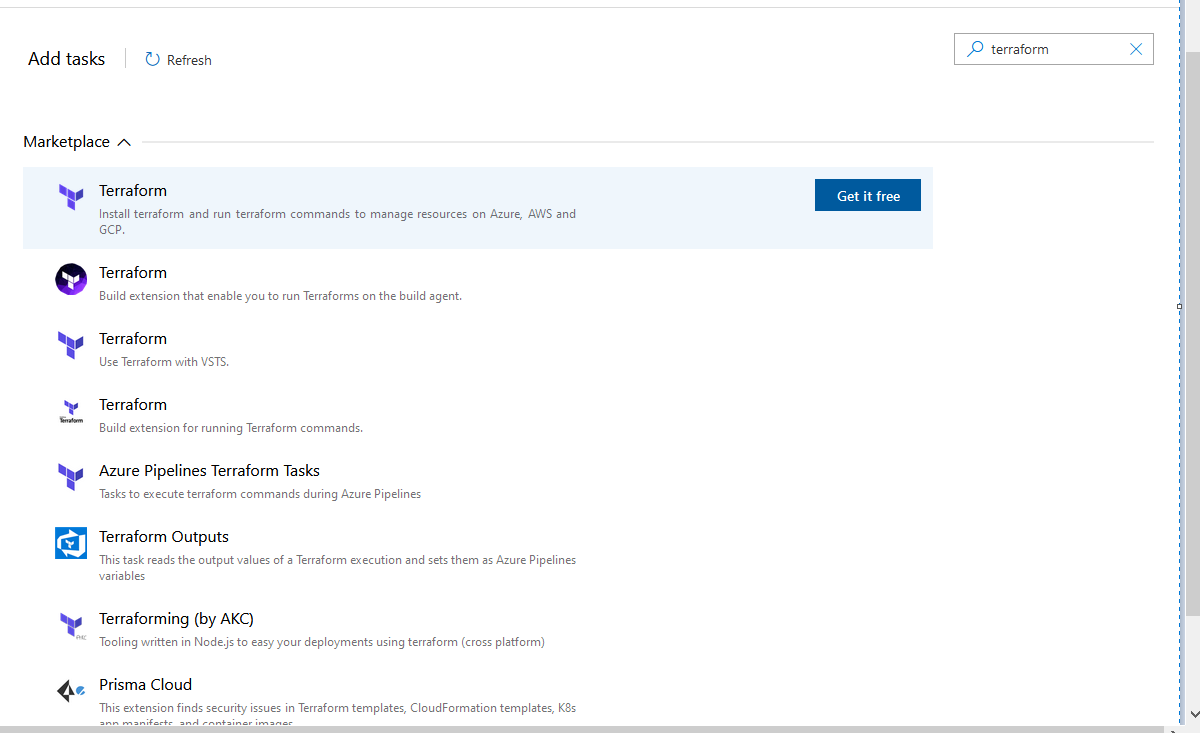
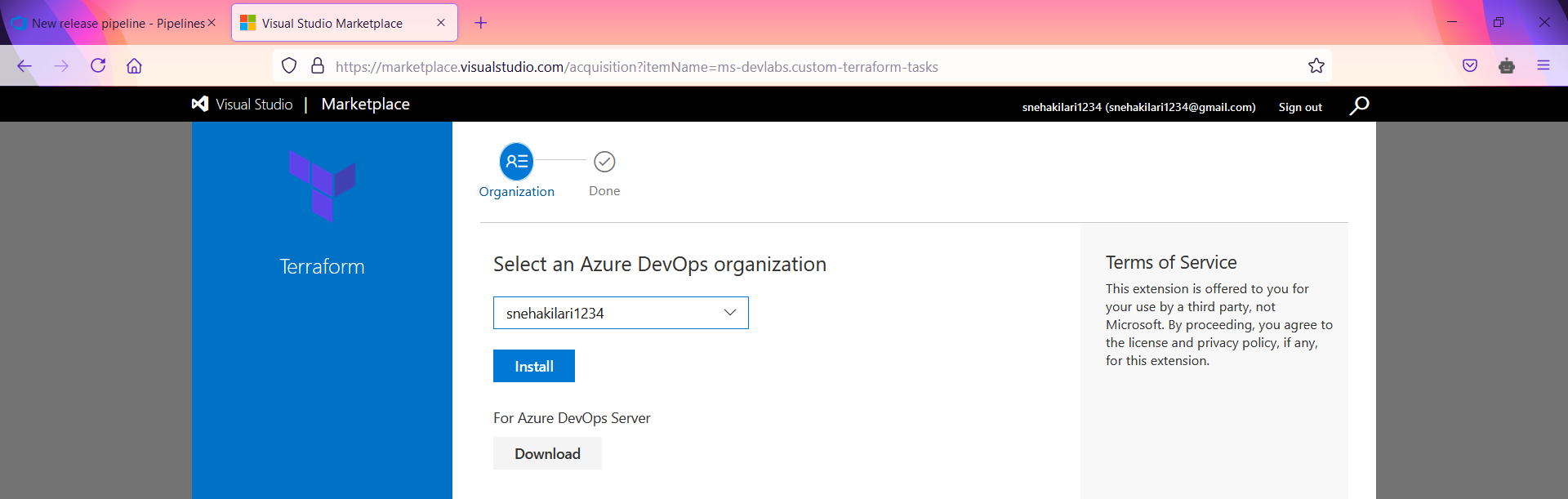
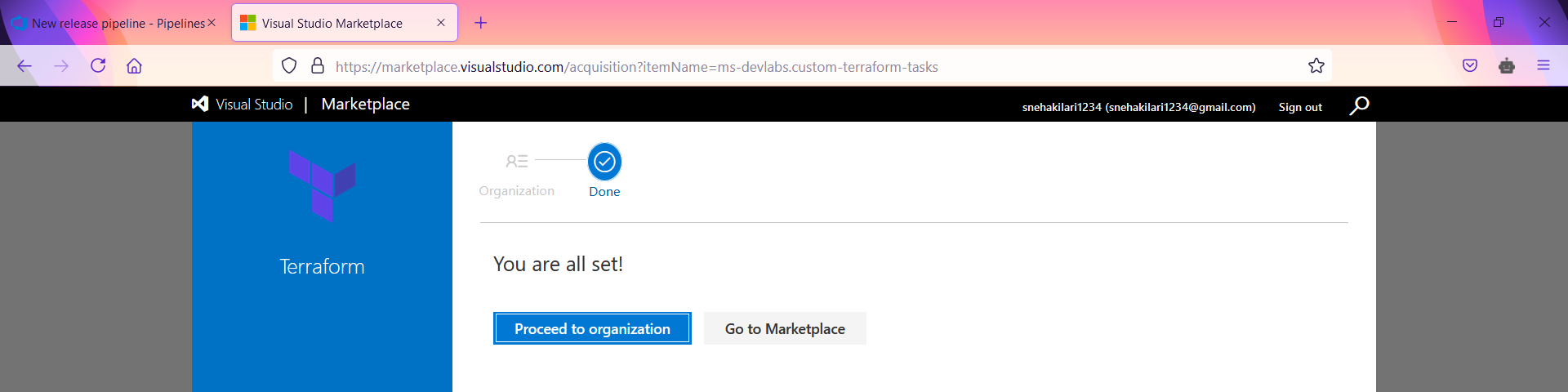
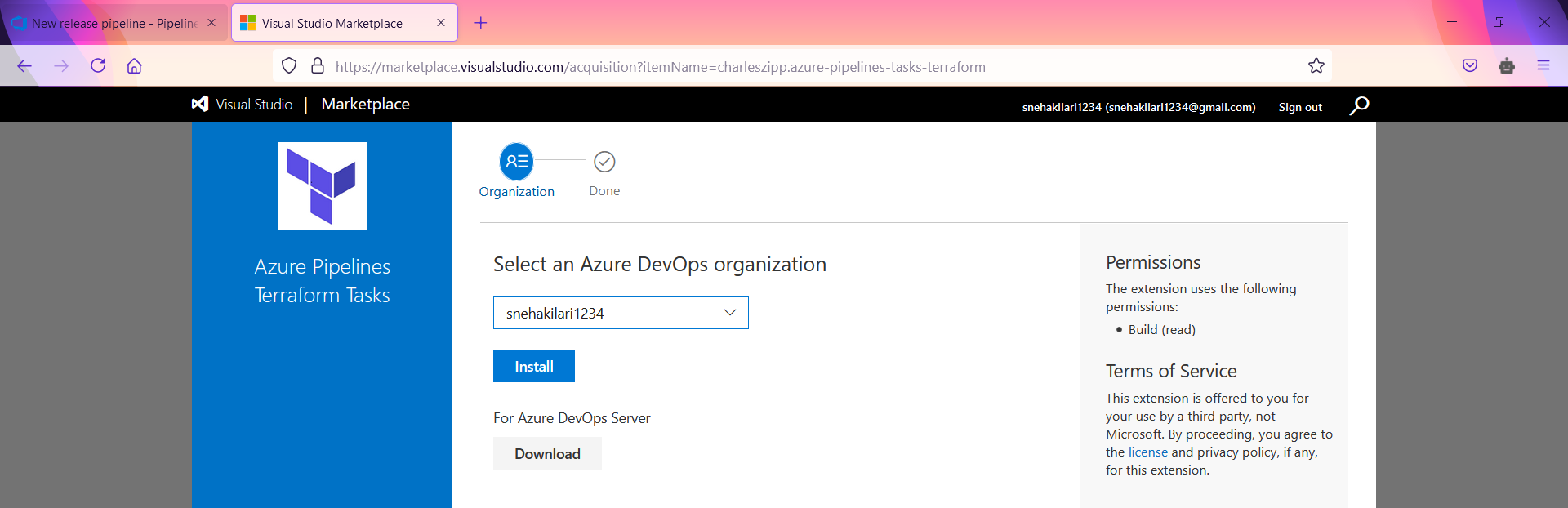
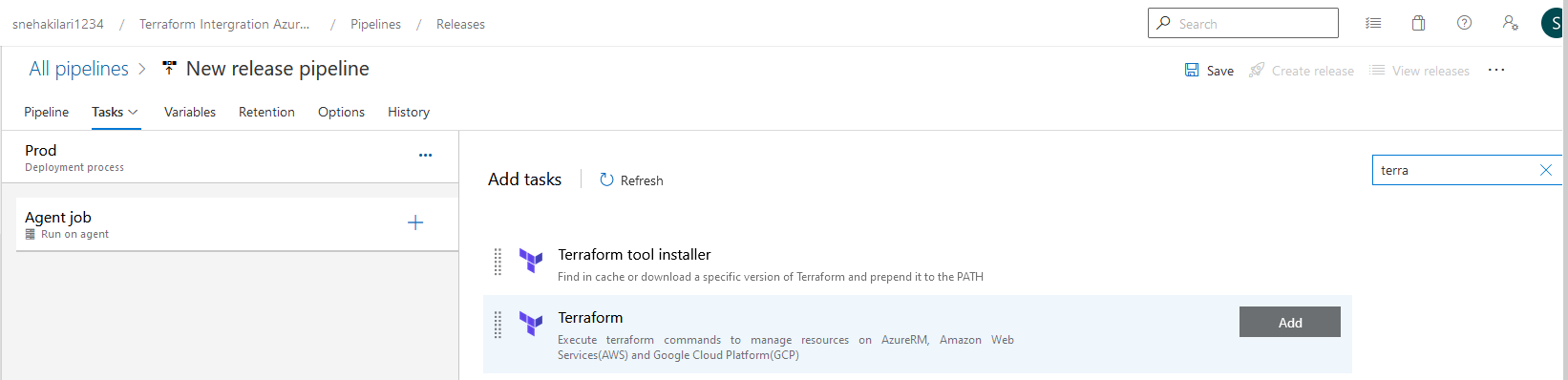
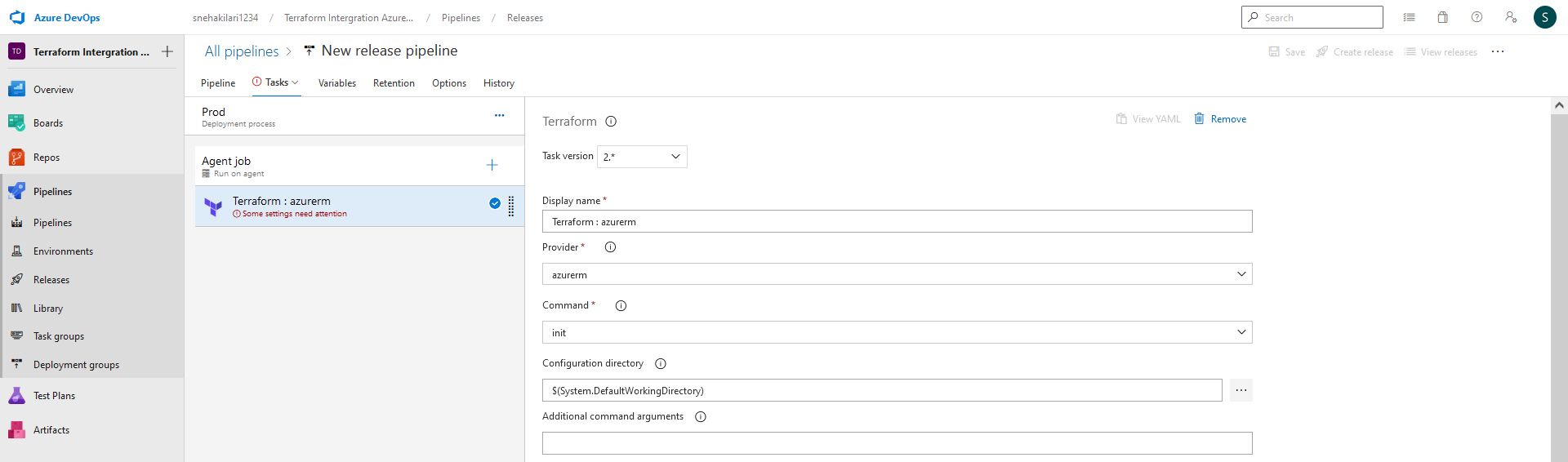
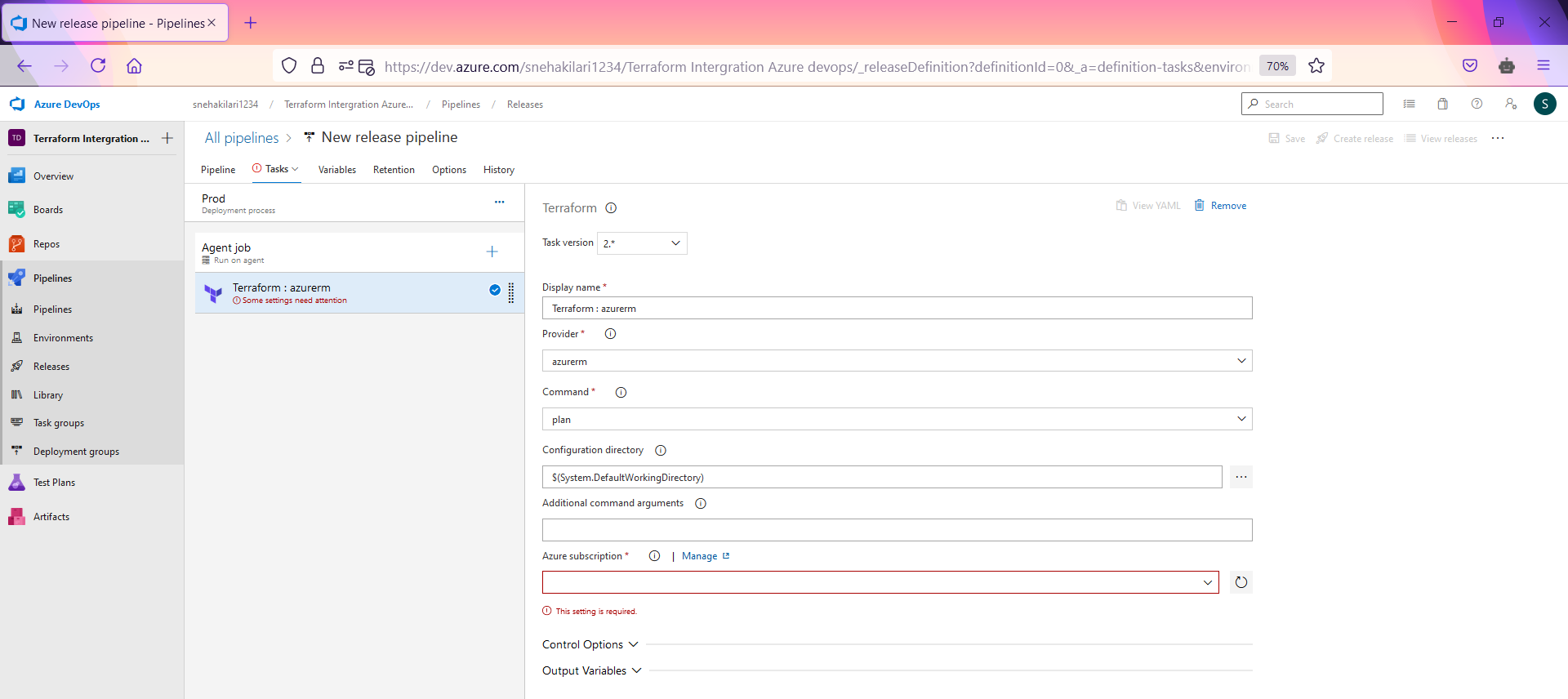
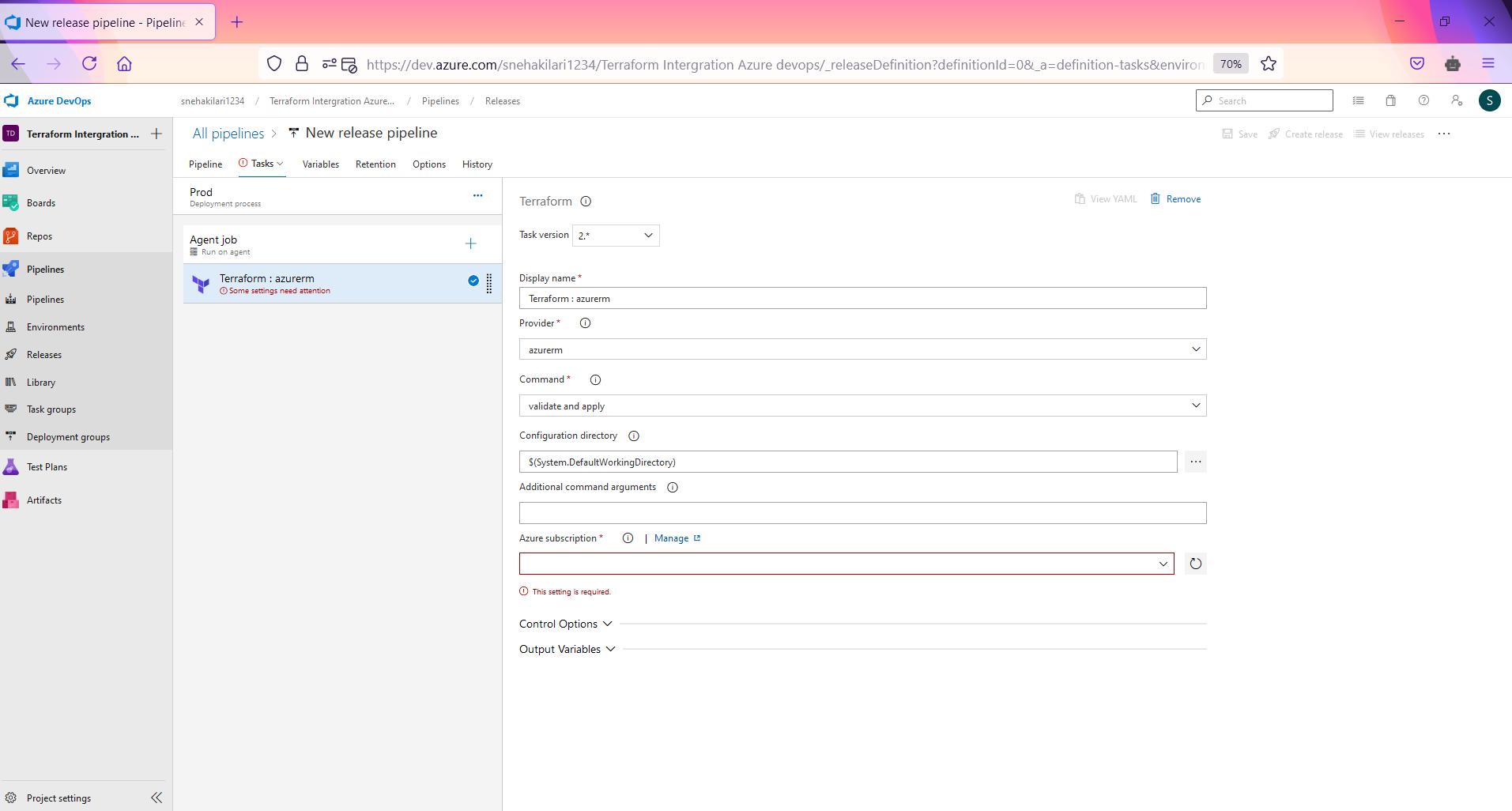


  
  
**We selected Empty Job:**



* Provide a name for stage “prod” and click the close button.   
    
    
  
* Select Add an artifact and choose Azure repository.   
    
  
* Choose a project created in Azure DevOps in step 1 and select the repository where network.tf is stored.
* Select the master branch and latest from the default branch in default version and click ADD.  
    
  
* Select prod stage and click View stage tasks to view the pipeline tasks.



* Click + in Agent job, search for terraform and click add on terraform tool installer once and terraform with thrice (need to add three tasks).  
    
    
    
    
    
    
    
    
    
    
    
    
    
    
    
    
  
* Select Terraform tool installer task and change terraform version to 0.12.10
* Select the first Terraform task and change the display name as Terraform init. Ensure provider as Azurerm and command as “init”. Select the configuration directory as network.tf is stored.  
    
  
* Select Azure subscription and authorize it. Choose your resource group, storage account, and blob container for storing terraform state files (terraform.tfstate).
* Select the second Terraform task and change the display name as Terraform plan. Provider as Azurerm and choose command as “Plan”. Select the configuration directory as network.tf is stored. Choose your Azure subscription for deployment.   
    
  
* In additional command arguments type: -var-file="network.tfvars" -out vnettest.tfplan
* Select the last Terraform task and change display name as Terraform apply. Ensure provider as Azurerm and command as “validate and apply”. Select the configuration directory as network.tf is stored.   
    
    
  
* In additional command arguments type: vnettest.tfplan. we reference the terraform state file for execution, which created in the above task. Choose your Azure subscription for deployment.
* Once you are done, Save the changes and Create a release.
* Once the release is a success, navigate to your Azure portal. Search for the name of the resource group which defined in the variable file and check virtual network is deployed.  
    
  **Finally Terraform integration with Azure devops we have 5 tasks.  
    
  Those are   
    
  1. Init  
  2. Plan  
  3. Validate and apply  
  4. Validate  
  5.Destory**