**Deploying of Terraform code using Azure DevOps CI/CD pipeline**

**(Using Microsoft Hosted Agent machine)**

Deploying infrastructure as code (IaC) using Terraform is a crucial practice for modern cloud management. Integrating this with Azure DevOps CI/CD pipelines brings automation, consistency, faster deploying and reliability to your infrastructure building.

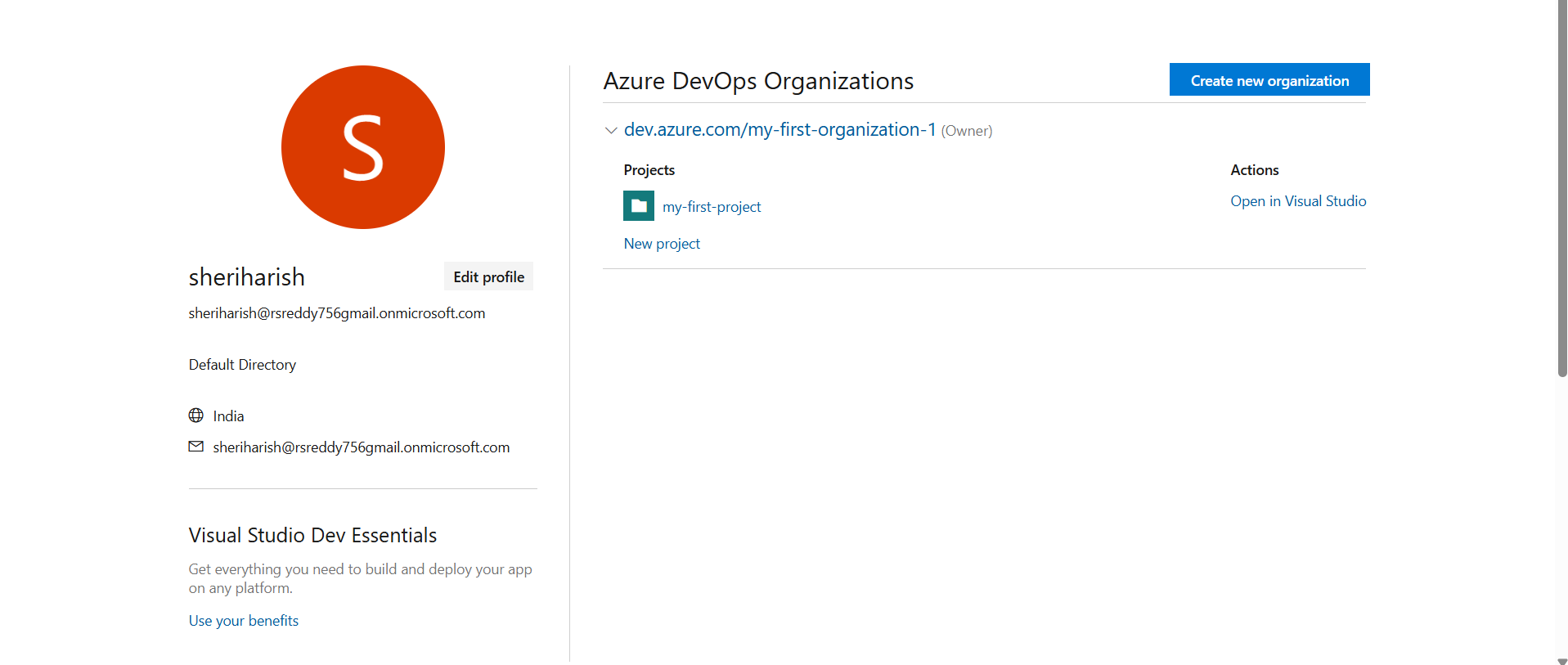
By using Azure pipeline we can easily and fastly deploy our application or infrastructure in the cloud.

Here we firstly push the terraform code (VM creation code) to the Azure Repository (Azure repo) and then we perform the CI and CD.

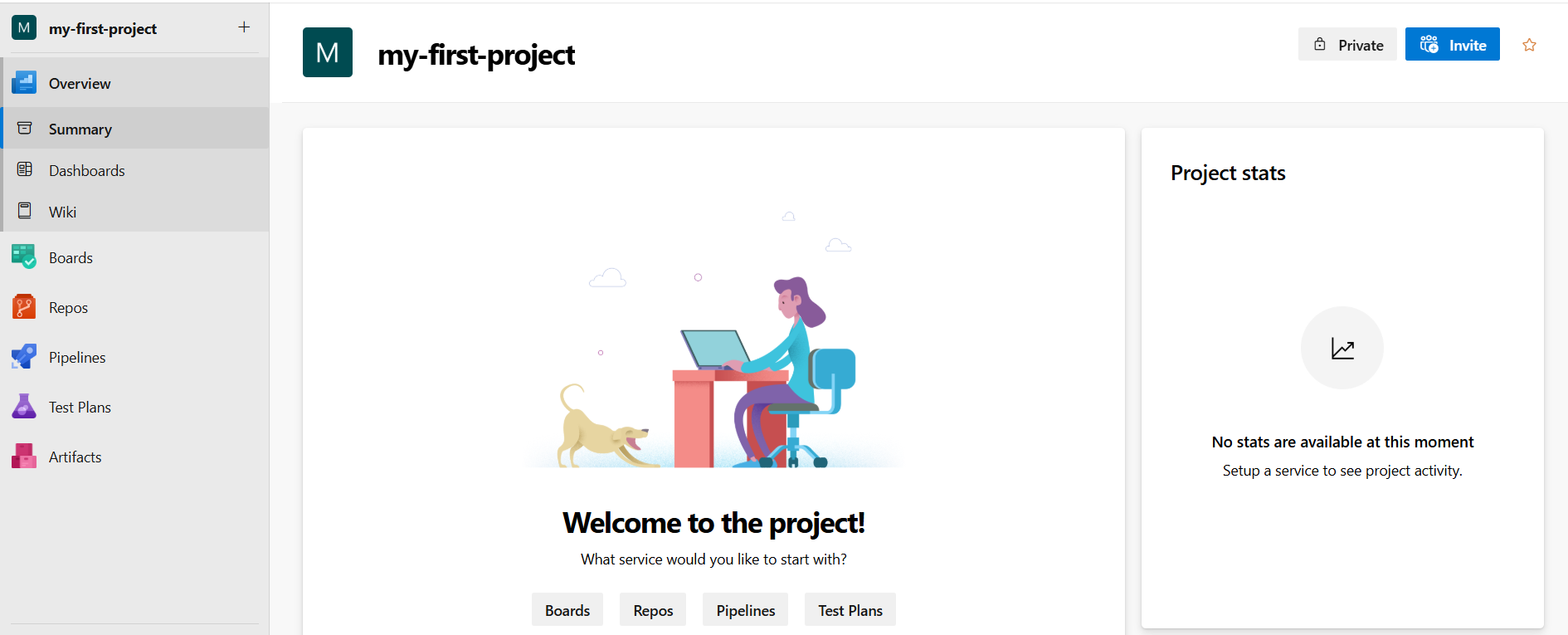
**Note:** We know that terraform code (IAC) does not required any build engine tools like Apache maven, MSBuild, Gradle, So we it does not need CI (continuous integration), it directly build the infrastructure in azure.

**Let’s do it step by step in practically:**

**Step1:** In Azure DevOps create the Organization (my-first-organization-1) within it Create the project (my-first-project).

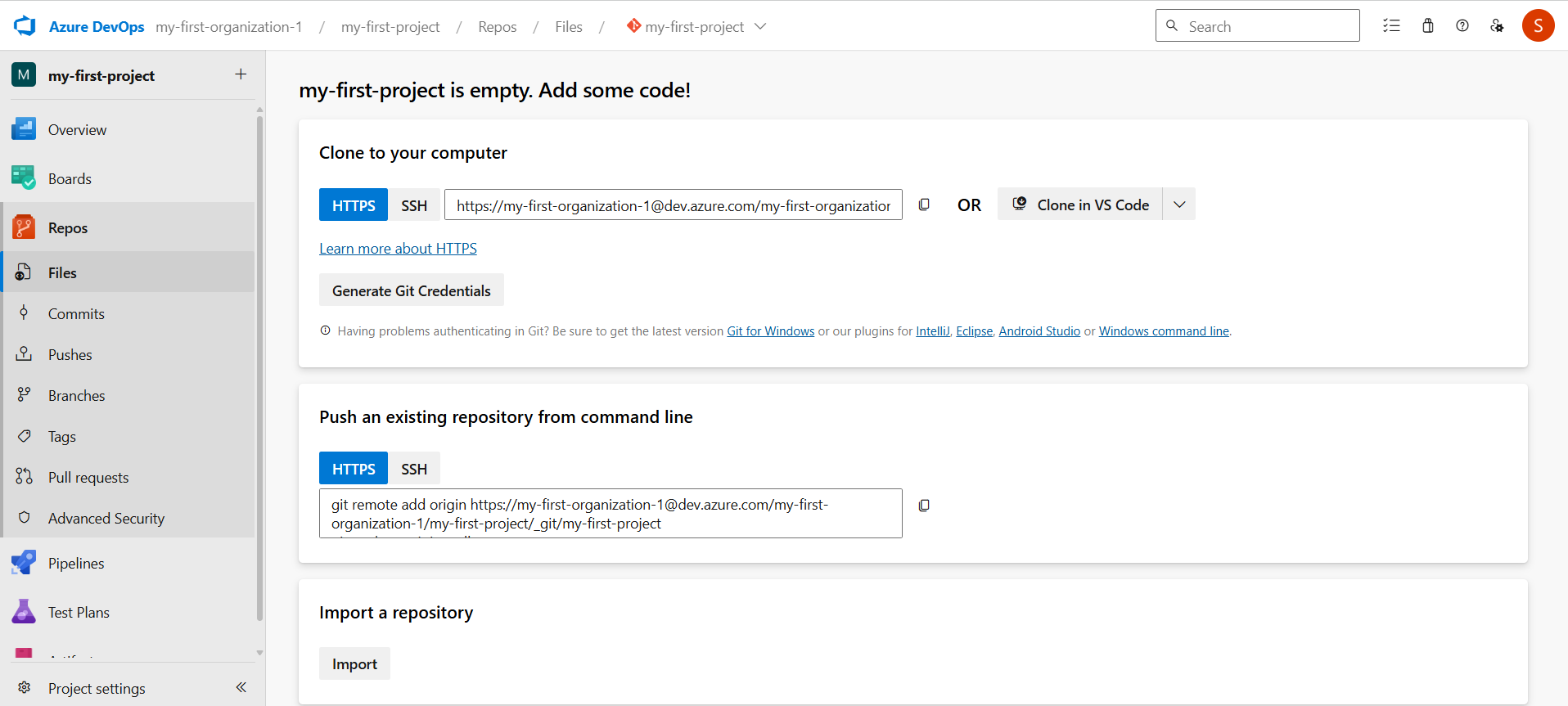
****

Within a project we can find the Azure repository **(Repos)** which is a version control system provided by Microsoft azure. It is same as GitHub here we use similar commands which are used in Git CLI.

****

**Fig:** Repos (Azure Repository).

Name of the repository is same as the name of the project by default as shown in below figure. We can also change it manually.



**Step2**: Push the Terraform code to the Azure Repository (my-first-project).

**Note:** In this process we no need provide authentication in main.tf file. We only passes the provider information like whether it is Azure or AWS or GCP.

#provider Block

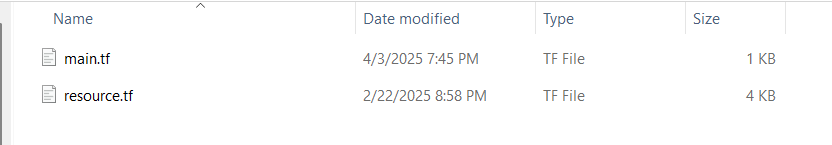
provider "azurerm" {

  features {}

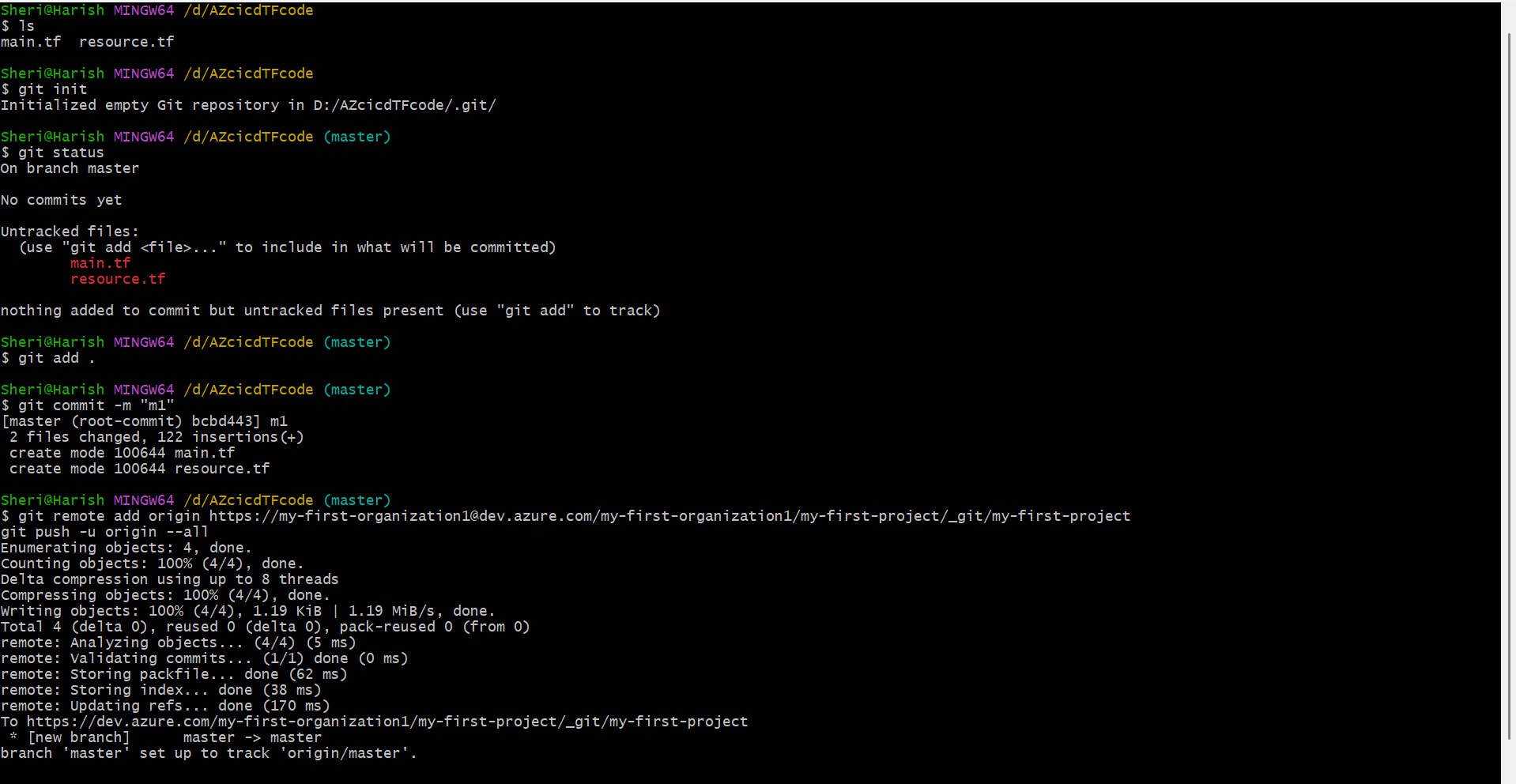
}

Fig: Main.tf file

And the **resource.tf** file consist of VM creation information like resource group, network, subnet, public IP, NSG and so on.



Now Push these two terraform code file (main.tf & resource.tf) to the Azure repository (my-first-project).



**Note:** Here also we use same commands which are used in GitHub in order to push local files to Remote Azure repository (Repos). Both GitHub and Azure Repos are Microsoft products.

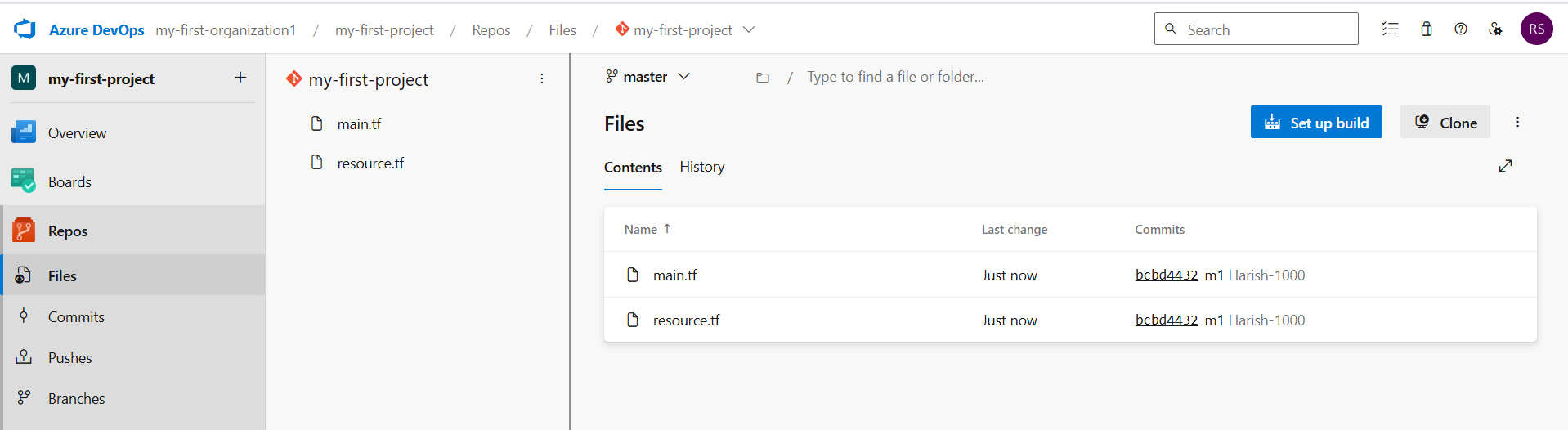


Fig: Two terraform files are pushed successfully.

**Step3:** Now perform the CI (continuous integration).

Terraform code is considered Infrastructure as Code (IaC), and it does not require build tools like Apache Maven to generate packages such as .jar or .war files. Therefore, continuous integration (CI) is not necessary in an Azure pipeline when working with Terraform.

But practically what happen in Agent machine while working with JAVA, Python and other programing languages in CI (continuous integration) is shown below

Source Directory

Workfolder\1\s

**Copy files**

Artifacts Directory

Pushes

Repo

Workfolder\1\a

Publish Artifacts

Agent machine (Linux/windows)

Developer

Fig: Block Diagram of what happen in agent machine while performing CI.

First, the developer pushes the code to the Azure Repos. Then, the Azure DevOps Agent (Agent machine) downloads the source code from the repository and places it into the source directory ($(Build.SourcesDirectory) which is typically **(\\_work\1\s).** After the build or processing steps, the output (known as build artifacts) is saved to the artifacts directory ($(Build.ArtifactStagingDirectory) which is usually **(\\_work\1\a).** Finally, these artifacts are published, so they can be used in release pipelines or deployments.

The output of CI (published artifacts) is the input of CD

CI output=CD input

**Note:** We can’t get artifacts while performing CD until unless we publish artifacts in the CI configuration.

**Step4:** Then perform the CD (continuous delivery/deployment).

We can perform continuous deployment/delivery (CD) directly for the terraform code as it does not require any build engine tools. So we can directly perform CD without configuring CI.

The process which is happened in Agent machine while working with CD is shown below.

**Downloads files**

Terraform install

Terraform init

Terraform validate

Terraform plan

Terraform apply

Repo

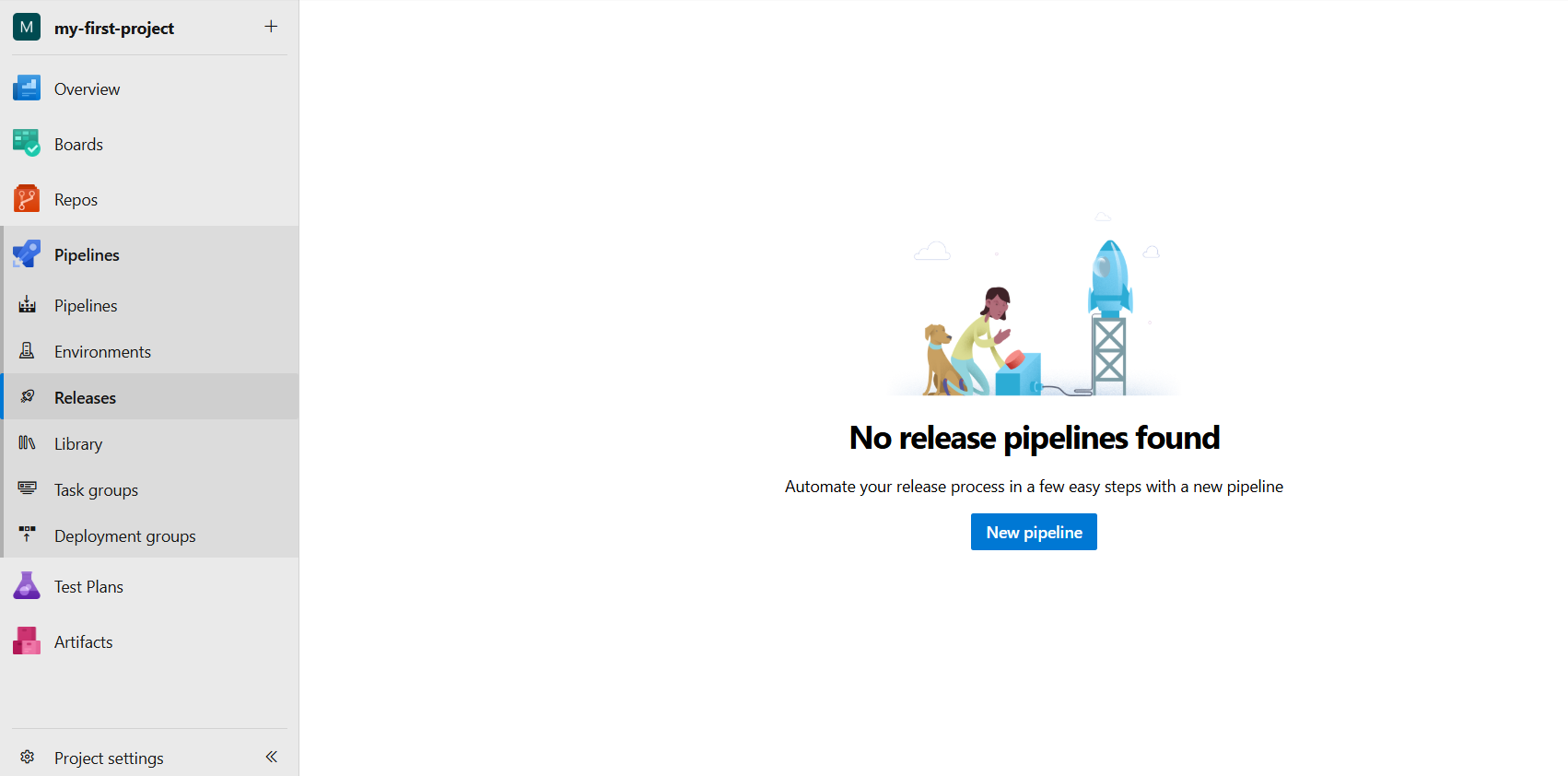
Agent machine (Linux/windows)

Developer

Fig: Block diagram of what happen in Agent machine while performing CD with Terraform code.

**Note:** This Agent machine which is used in above to perform CI & CD is a **Microsoft hosted machine** which is automatically created in background and deleted after performing all the tasks. We no need to take care of it, Microsoft will take care.

Let’s Perform CD by using Terraform code.



Here pipelines are used to perform CI (continuous integration) and Releases are used to perform CD (continuous delivery/deployment).

Pipeline🡺CI

Releases🡺CD

**Case1:** Adding of Artifacts.

Now click on **New pipeline** option and then **Add an artifact**, as we using terraform code so we don’t have any published artifacts so we directly pulls the code from Repository.

My-first-project🡪Pipelines🡪Releases🡪New pipeline🡪Add an artifact.

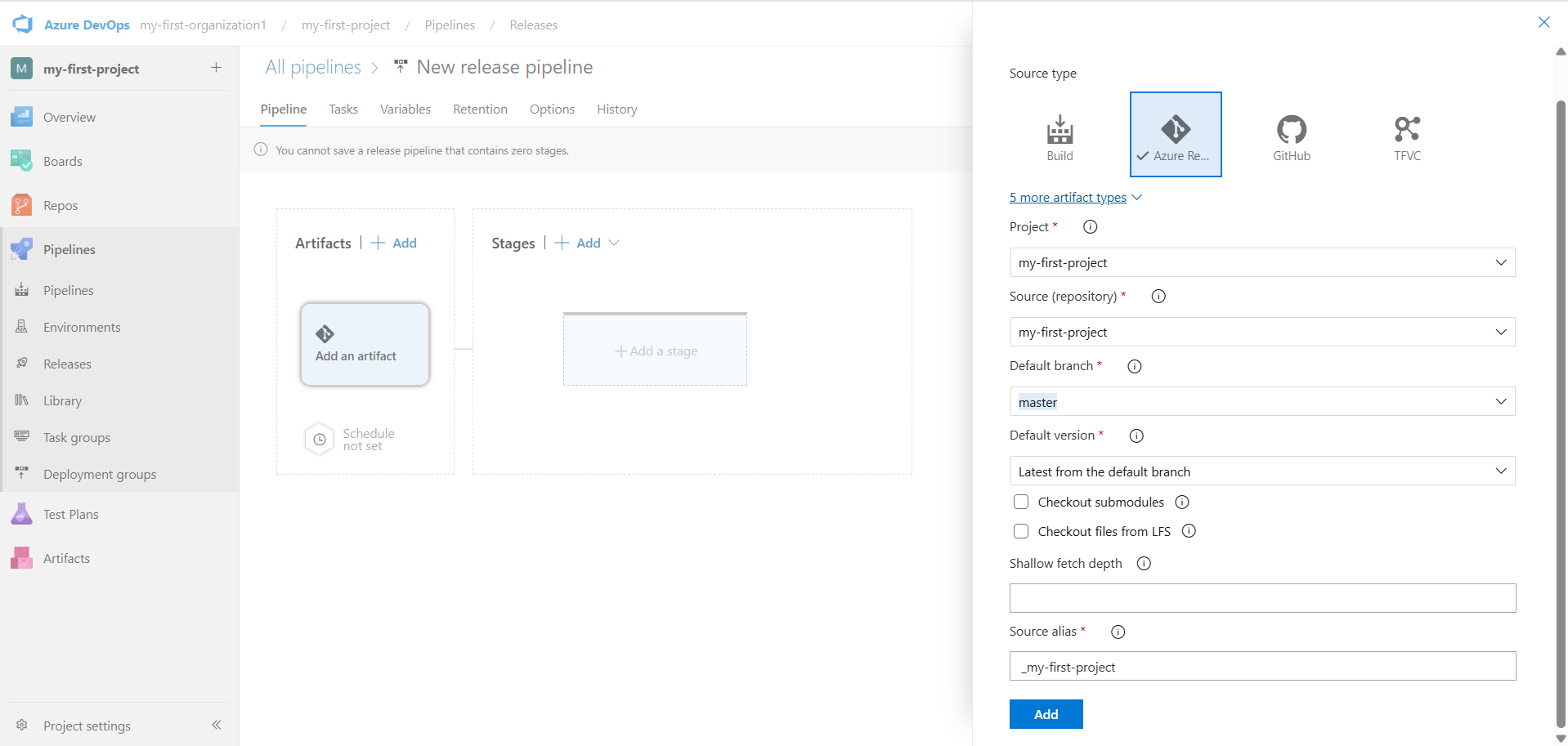
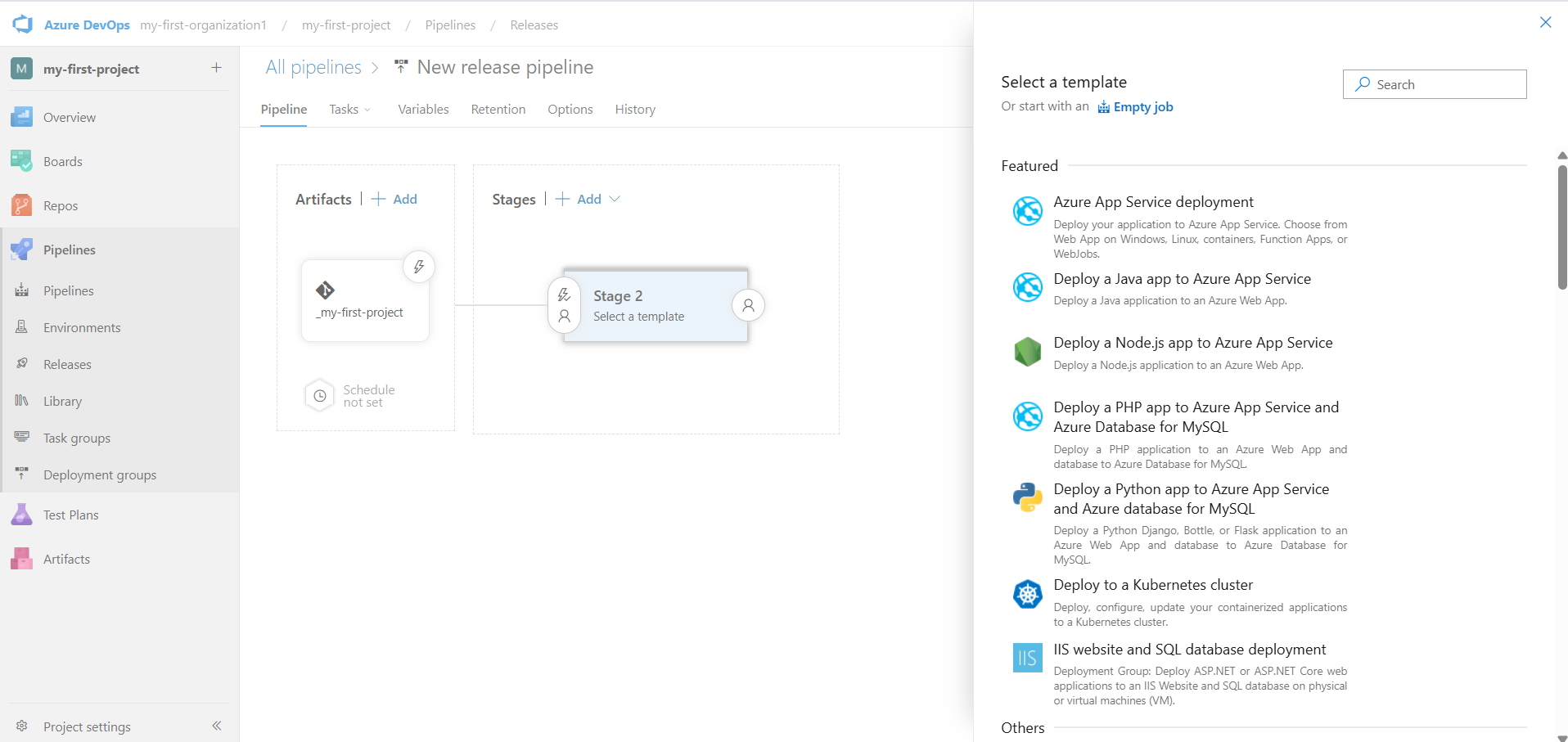


Fig: Adding of artifacts from Azure Repository.

**Case2:** Adding of stages.

My-first-project🡪Pipelines🡪Releases🡪New pipeline🡪Add a stage🡪Empty job🡪stage name🡪save



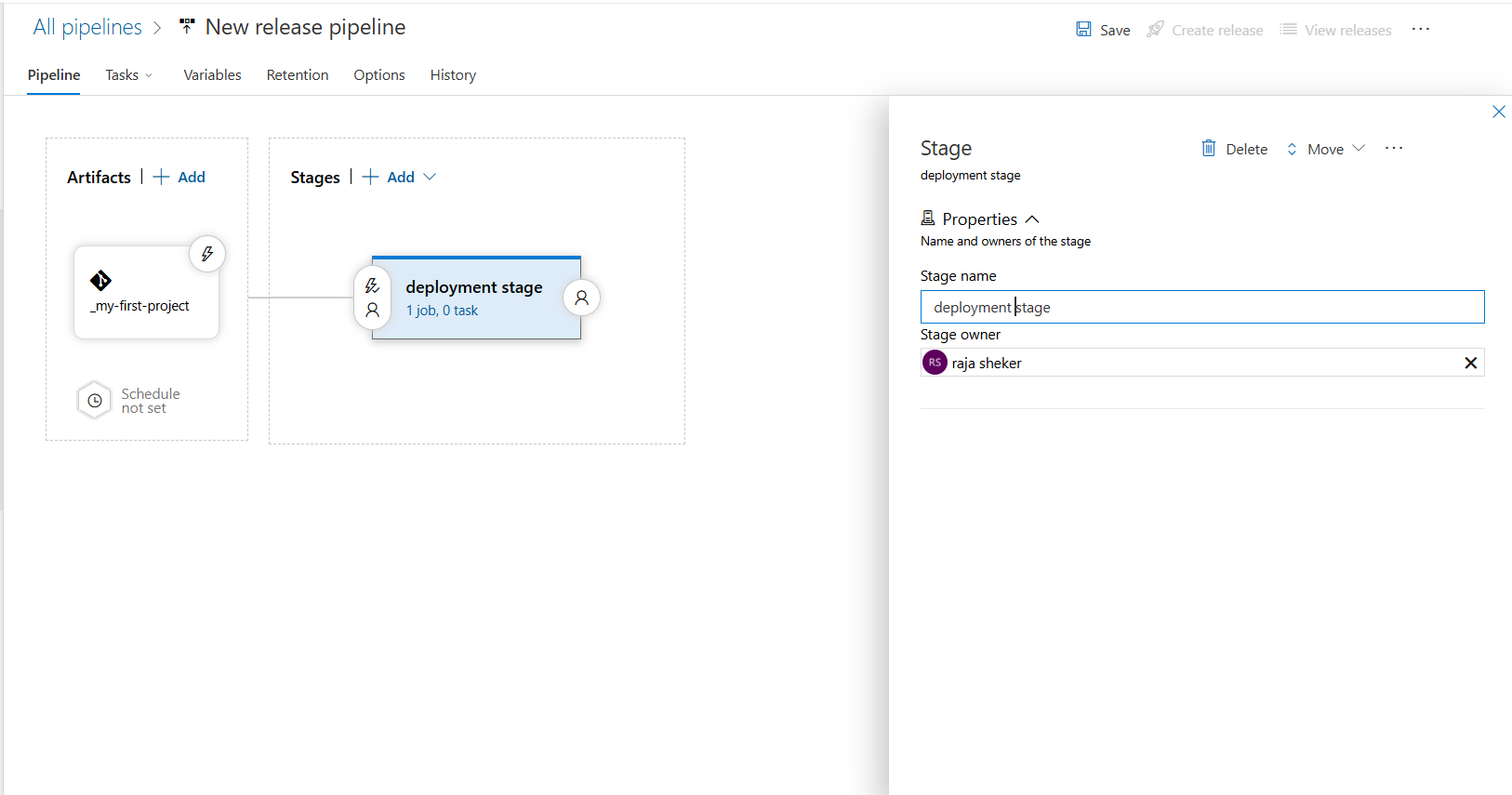


Fig: Renaming of staging name as “deployment stage”.

**Case3:** Add the Task’s to the deployment stage.

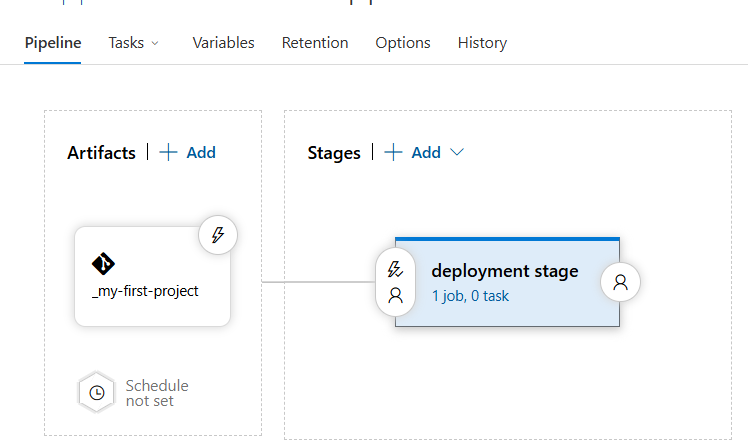


Fig: By default it has Zero tasks.

By clicking on “**1 job 0 task”** we can add the multiple task’s under Agent job.

1 job 0 task🡪” +”🡪search the task what you want to add.

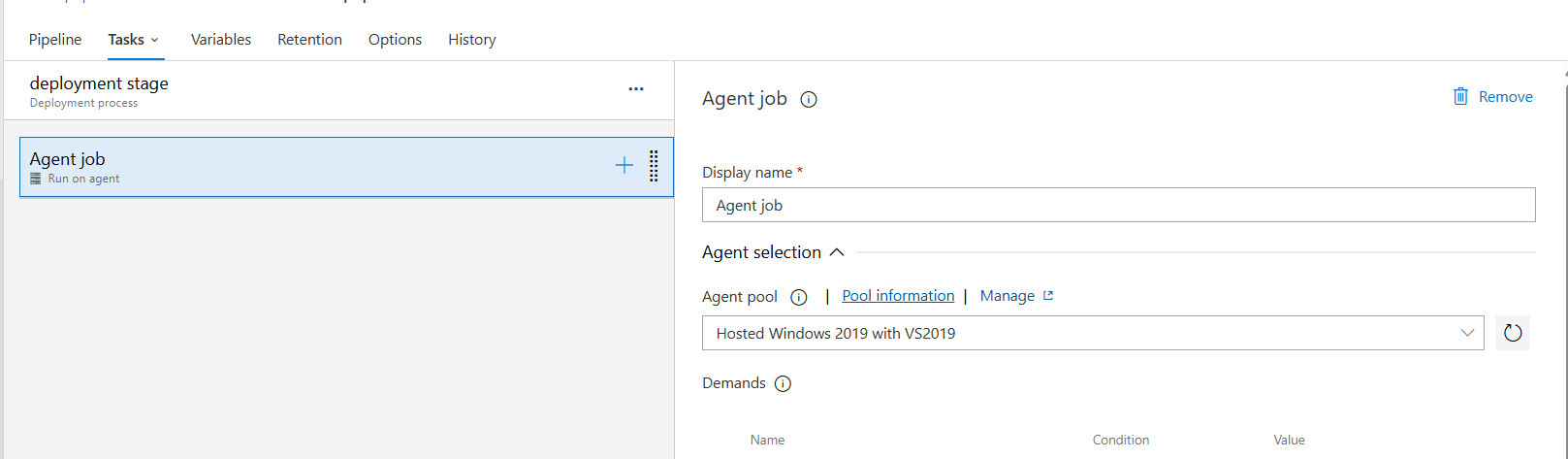


Fig: Agent job.

The above figure shows the Agent machine and its jobs, here the Microsoft is using the Windows machine (Hosted Windows 2019 with VS2019) which is created background. This agent machine firstly downloads the code from Azure Repository and the task’s which are used to execute the terraform code are added under the agent job, after performing the task’s in windows agent machine, the windows agent machine automatically deleted.

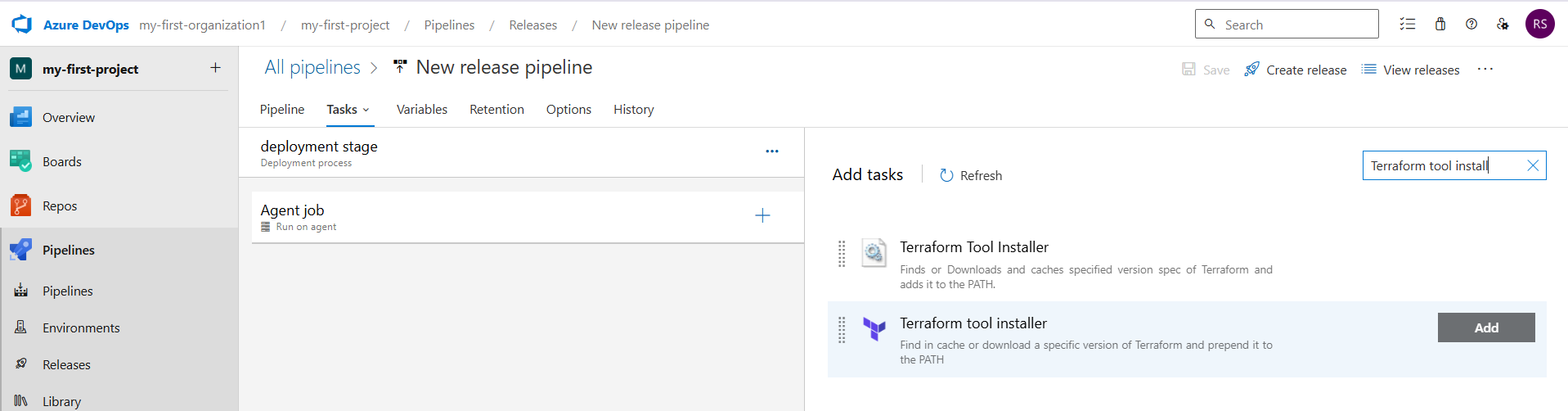
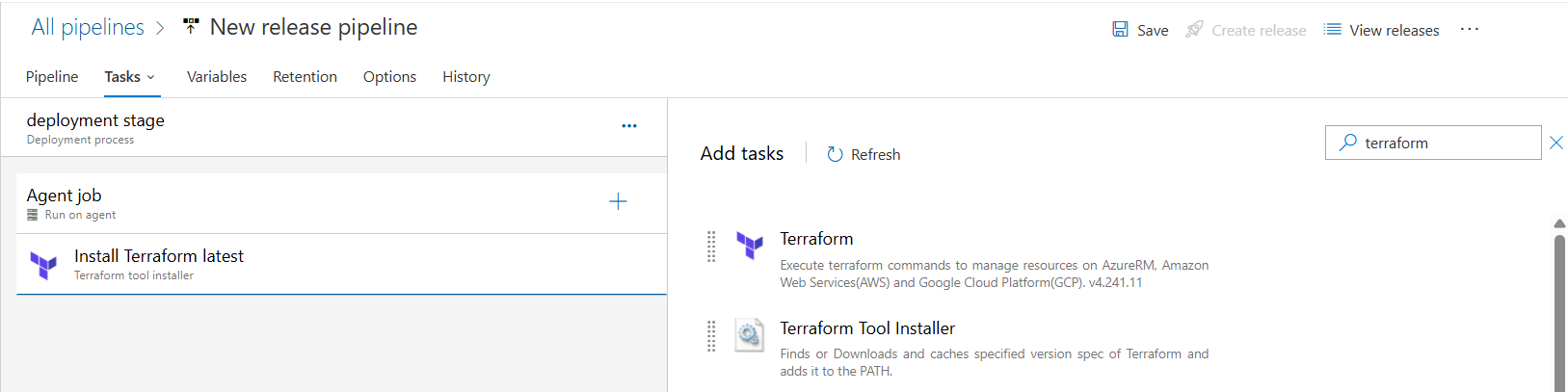


Fig: Adding of “Terraform tool install” task to the agent job.

Similarly add terraform executable command such as init, validate, plan and apply task under the agent job as shown below figure.



Azure DevOps is a separate service from the Azure cloud (Azure Portal). To connect these two, we need to set up authentication by creating a Service Principal through App Registration in Azure. This authentication setup is also used in the Terraform configuration (typically in the main.tf file provided in above) to allow Terraform to access and manage Azure resources.

Microsoft Entra ID🡪App registration

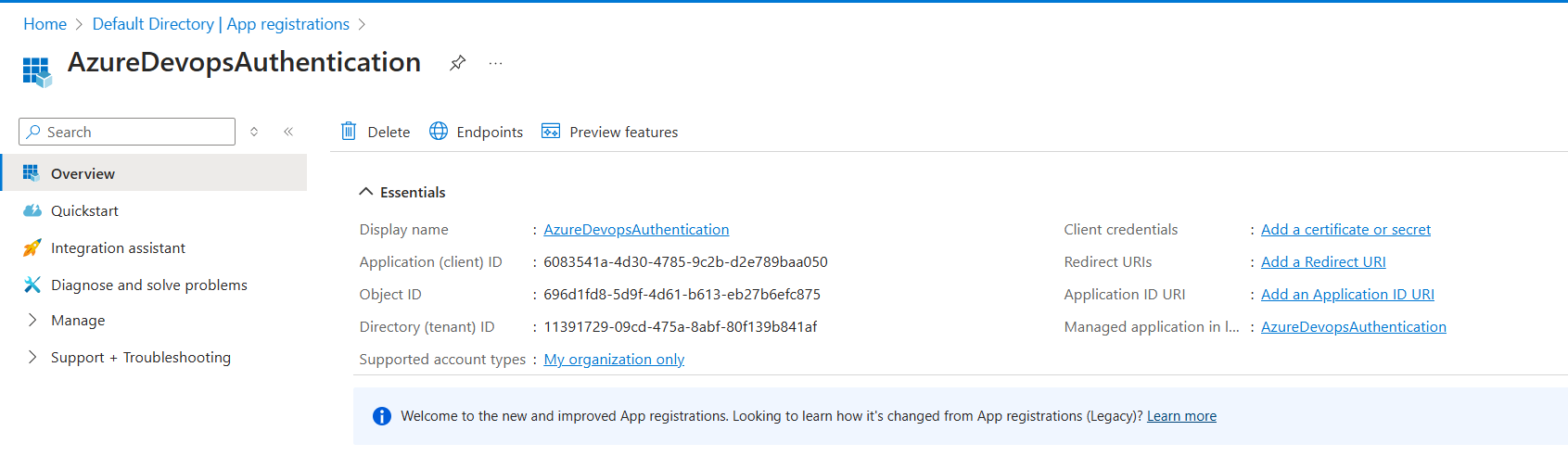
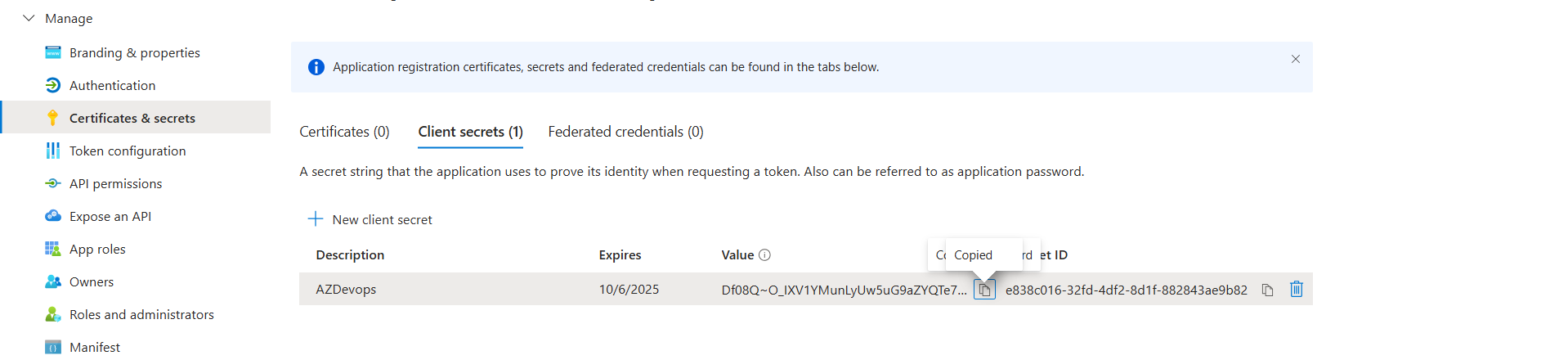


Fig: App registration.



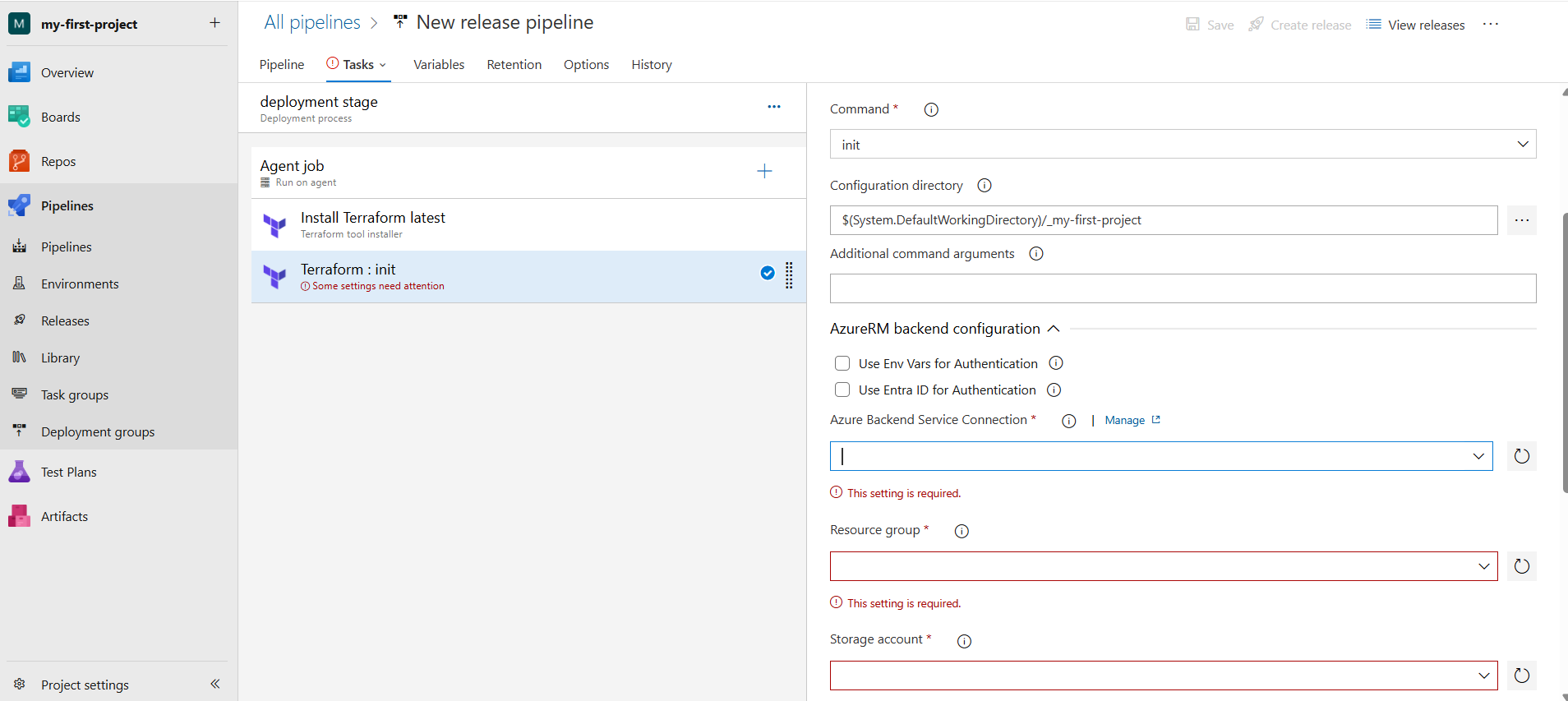
Fig: Secrete value.

Fig: Click on **“manage**” to configure Azure Backend Service Connection.

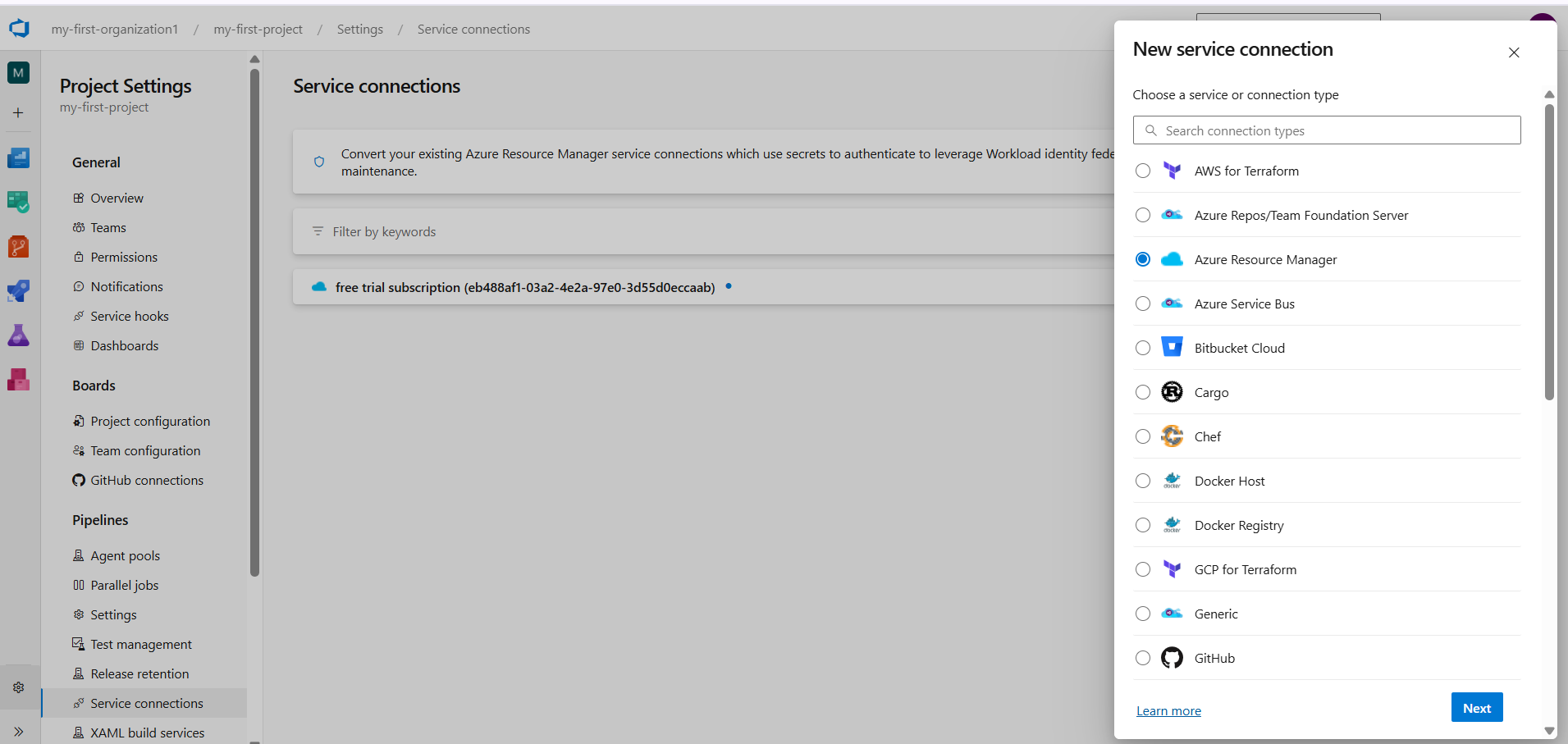


Fig: Azure Resource Manager.

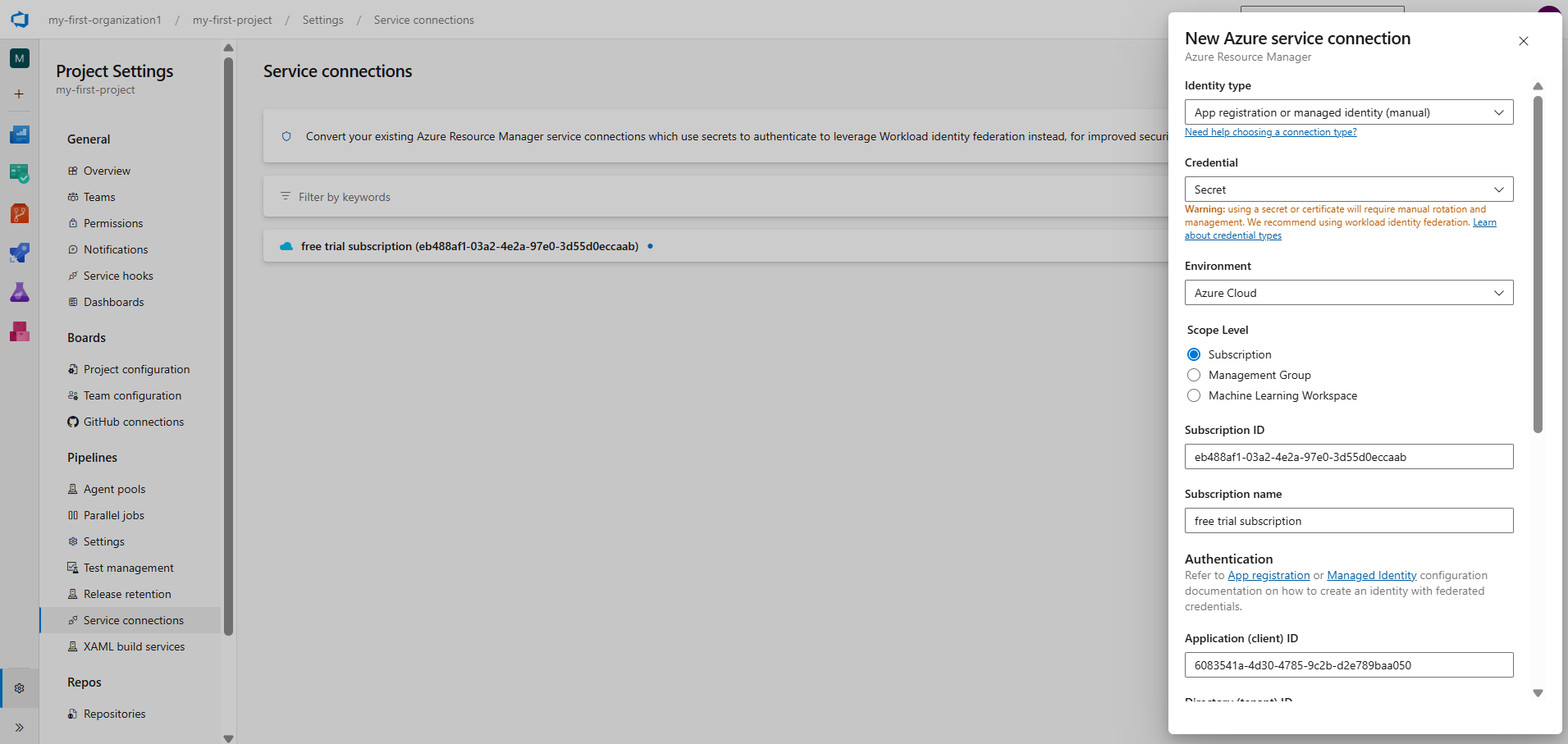


Fig: Defining the tenant ID, client ID, Subscription ID and Secrete Value.

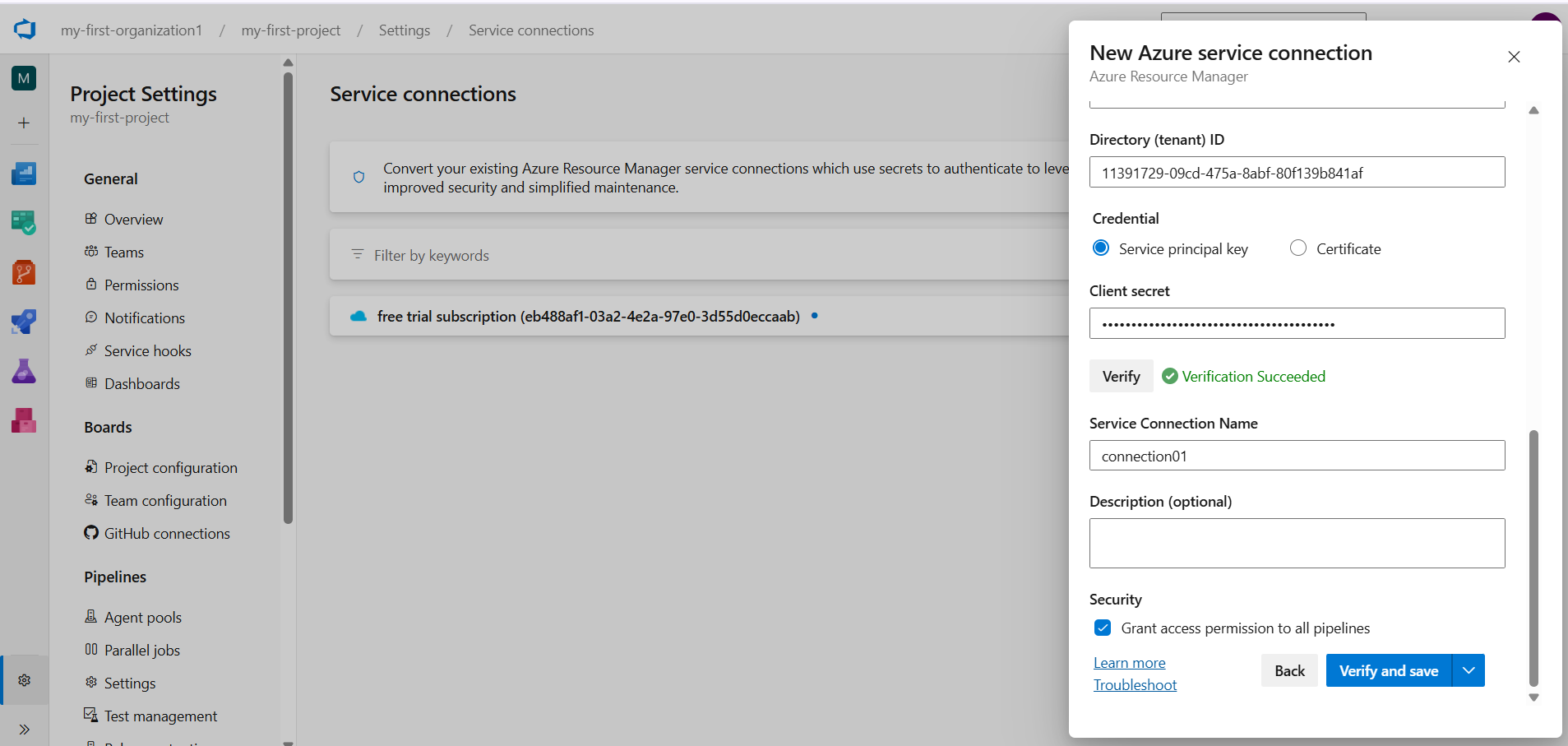


Fig: Enable “Grant access permission to all pipelines” to Service (connetion01).

Service connection is created successful with the name of “connection01”. Through this service connection we can connect Azure DevOps with the Azure cloud.

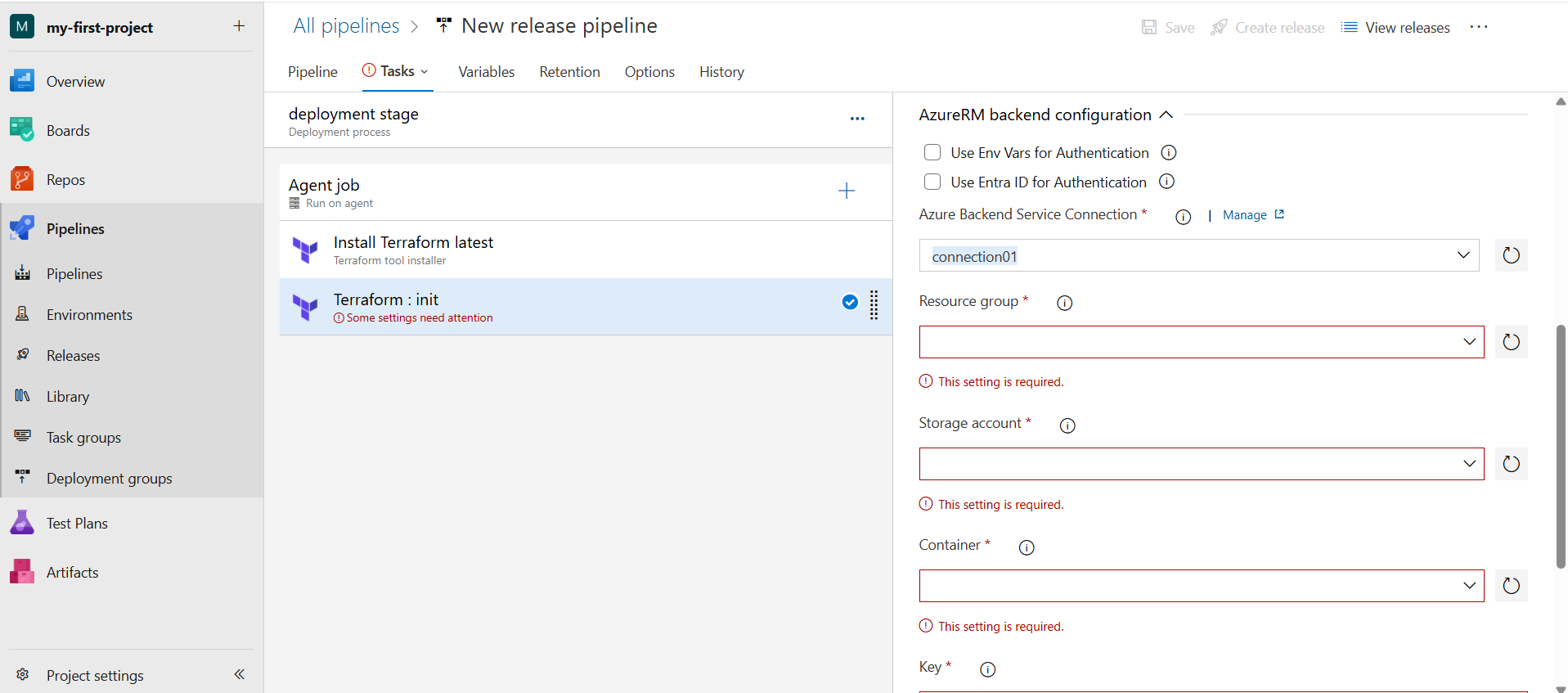


Fig: Selecting of Newly created service connection (connection01).

Here we have to create manually the resource group, storage account and container which is used to store the **.tfstate** file centralized way.

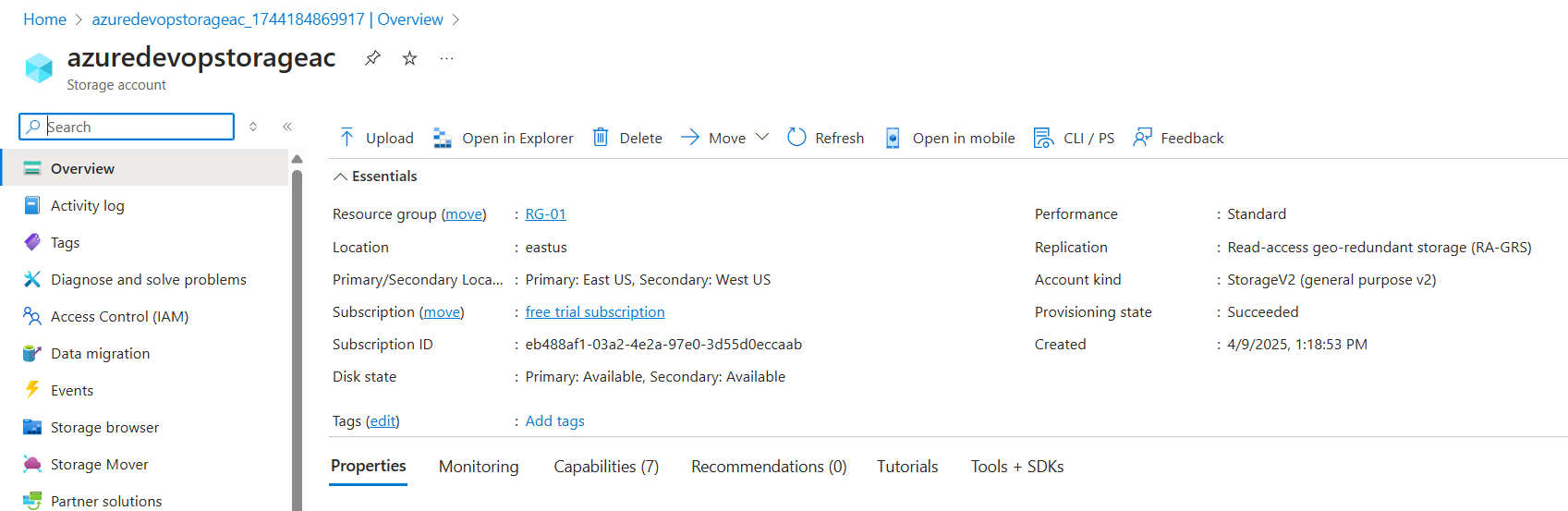


Fig: Storage account in the resource group RG-01.

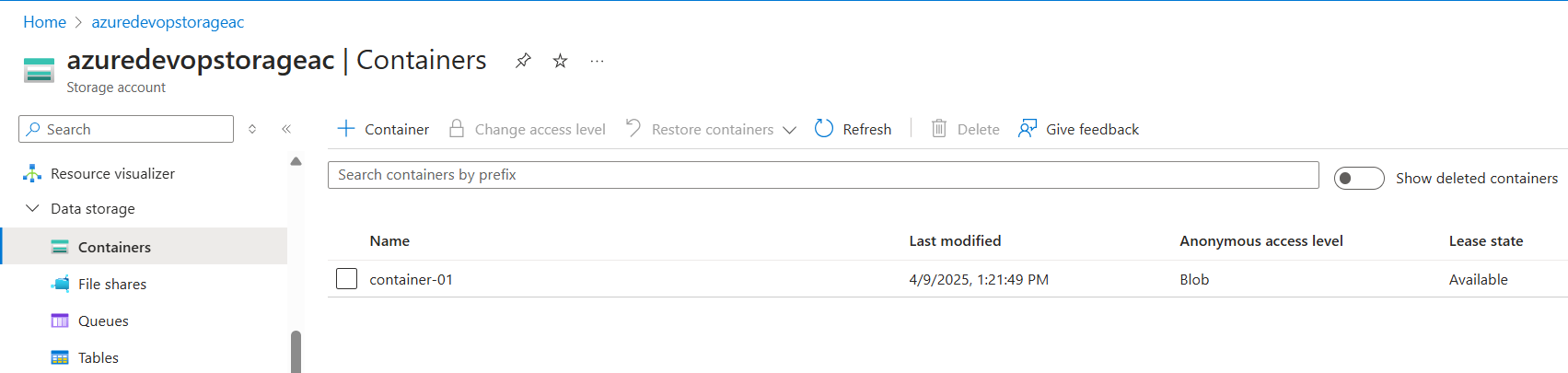
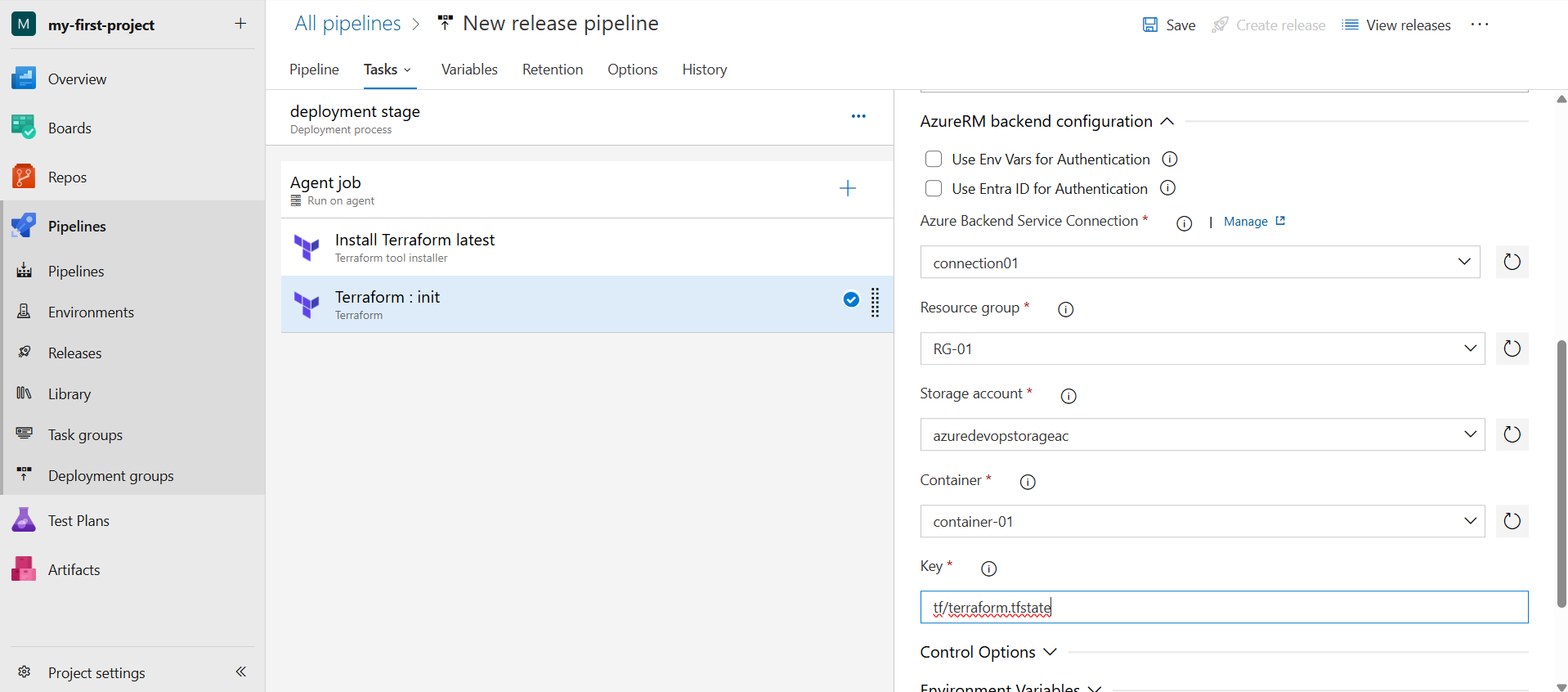


Fig: Container.

Now add all the Task’s to the agent job:



**Note:** In above figure the key is given manually by us, it is our wish what is to be given.

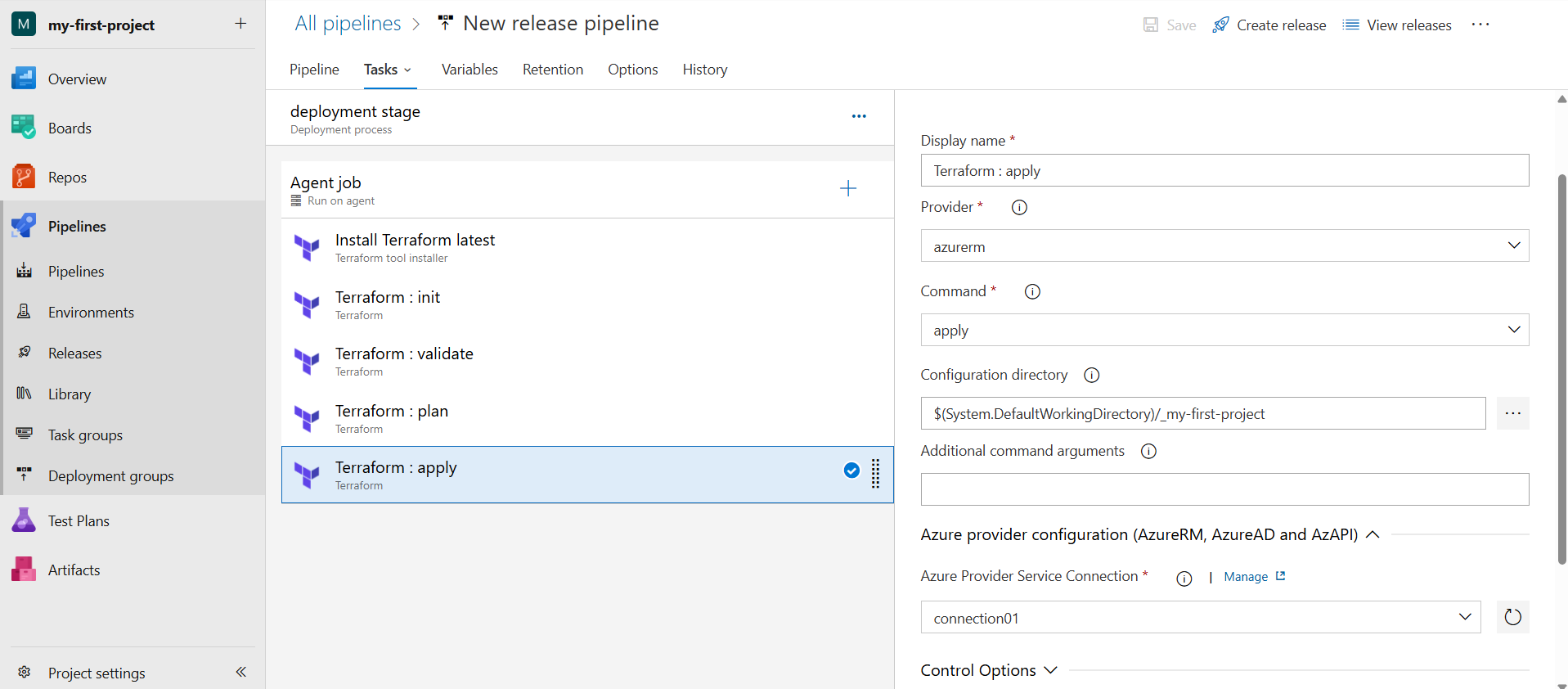
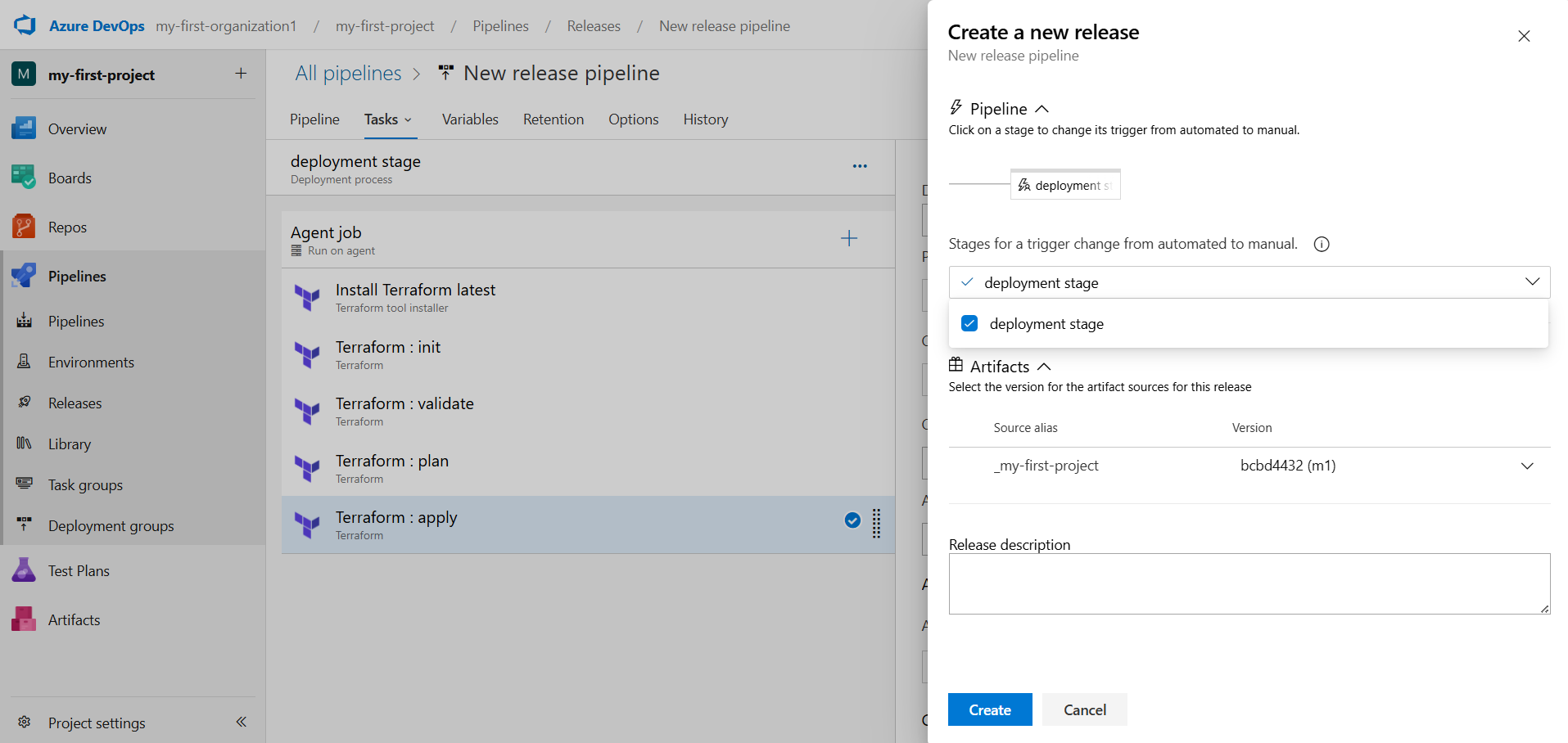
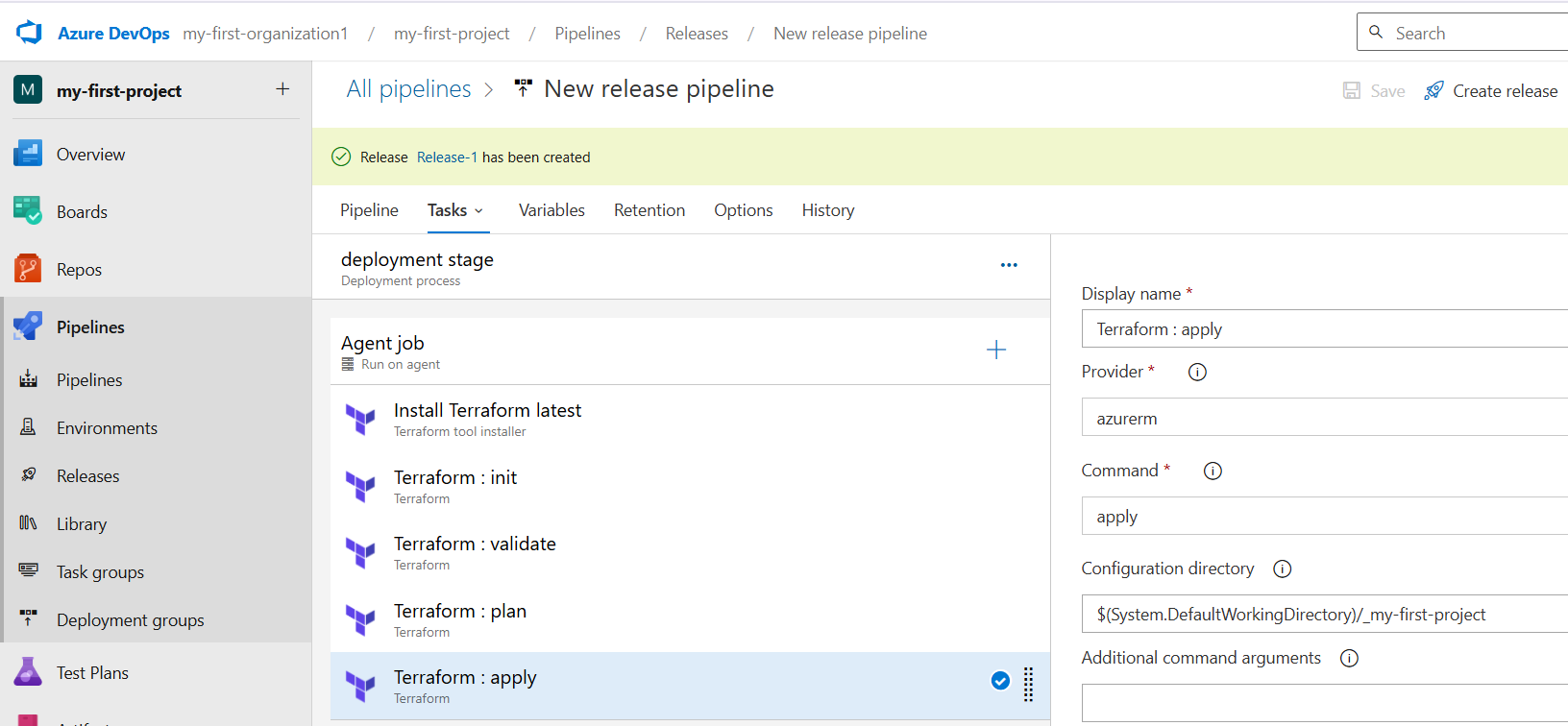


Fig: All the Terraform tasks are added successfully to the agent job.

Now click on the “**create release**”





Click on **Release-1** and the **deploy it.**

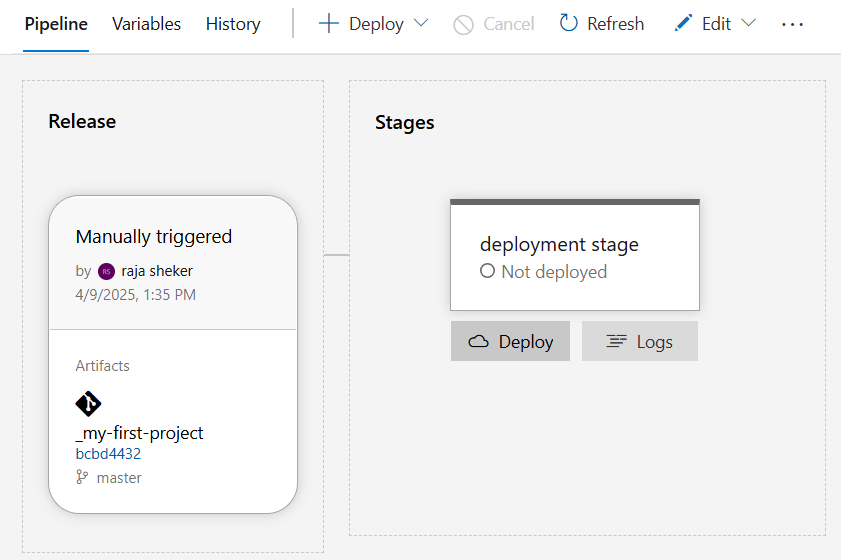


Fig: Deploying of our application (terraform code).

In above figure logs are used to check or view the records of the events while performing the tasks.