Create our virtual networks in two different regions according the instructions files.

1. Log in to the Azure DevOps Demo Generator: https://azuredevopsdemogenerator.azurewebsites.net/

Step 1 – Create Project

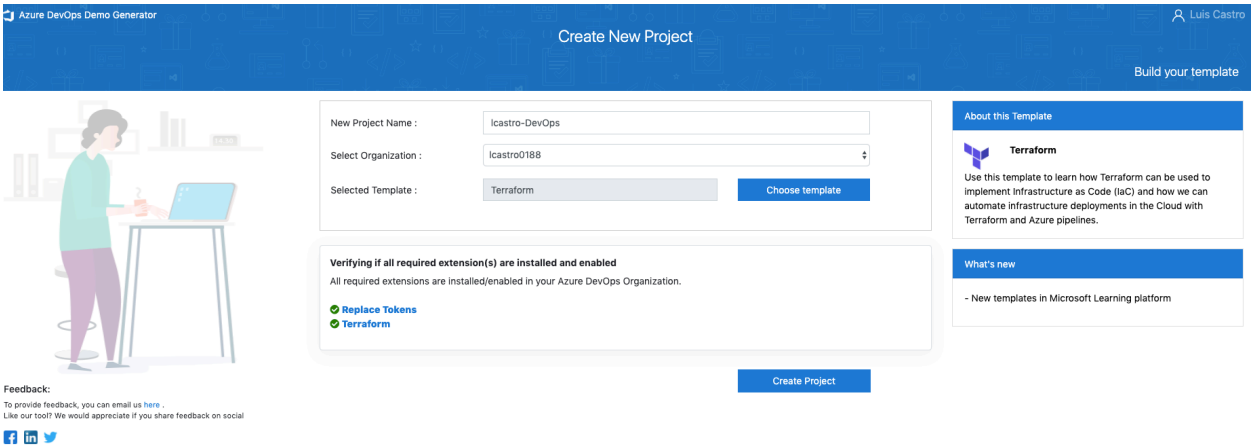
Project Name

username-devops

Select Organization

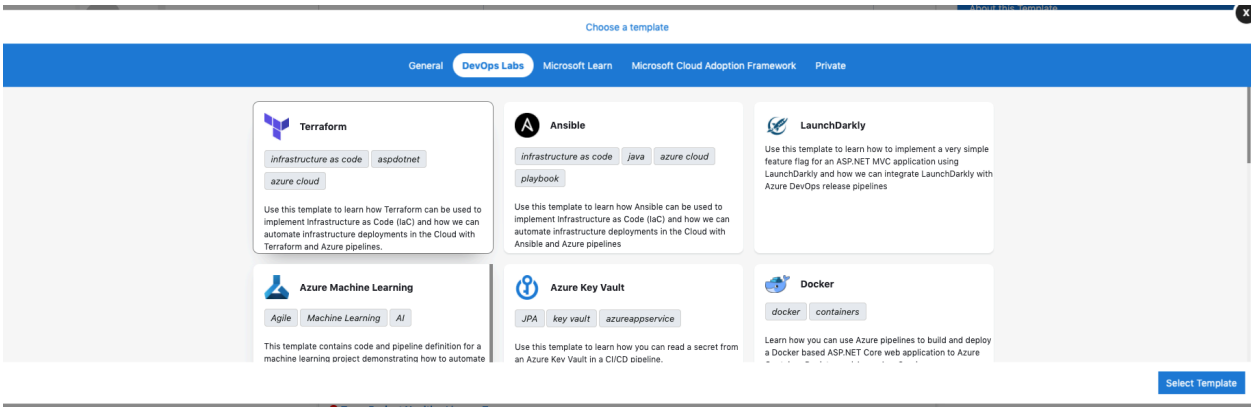
TBD

Choose Template



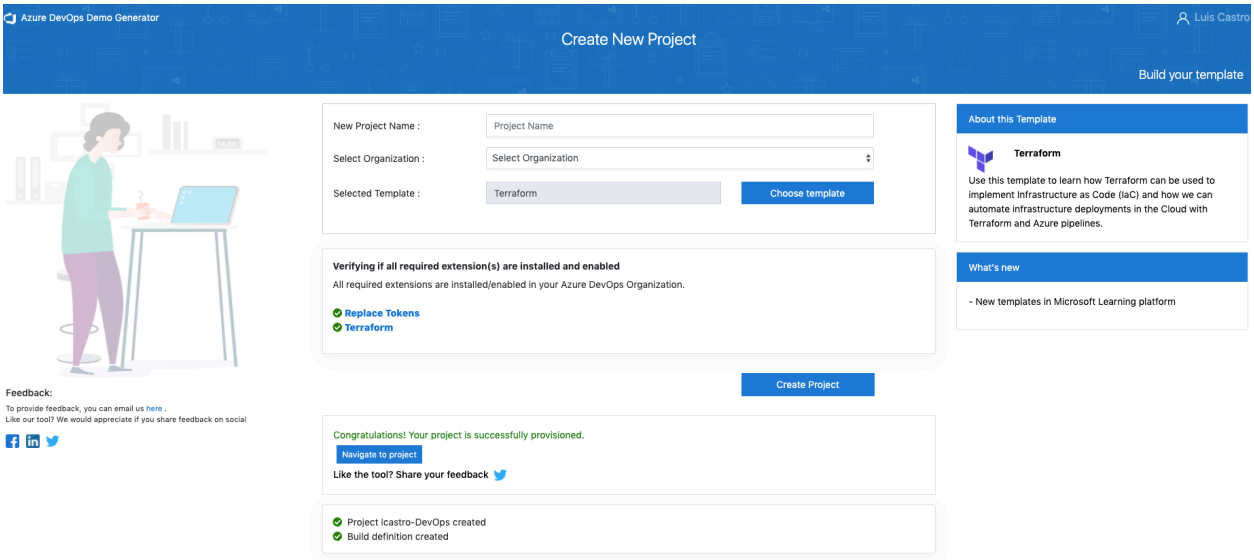
Step 2 - Verify Terraform Version

Select DevOps Labs and click Terraform>Select Template



Step 3 - Verify Extensions

- **Replace Tokens**
- **Terraform**

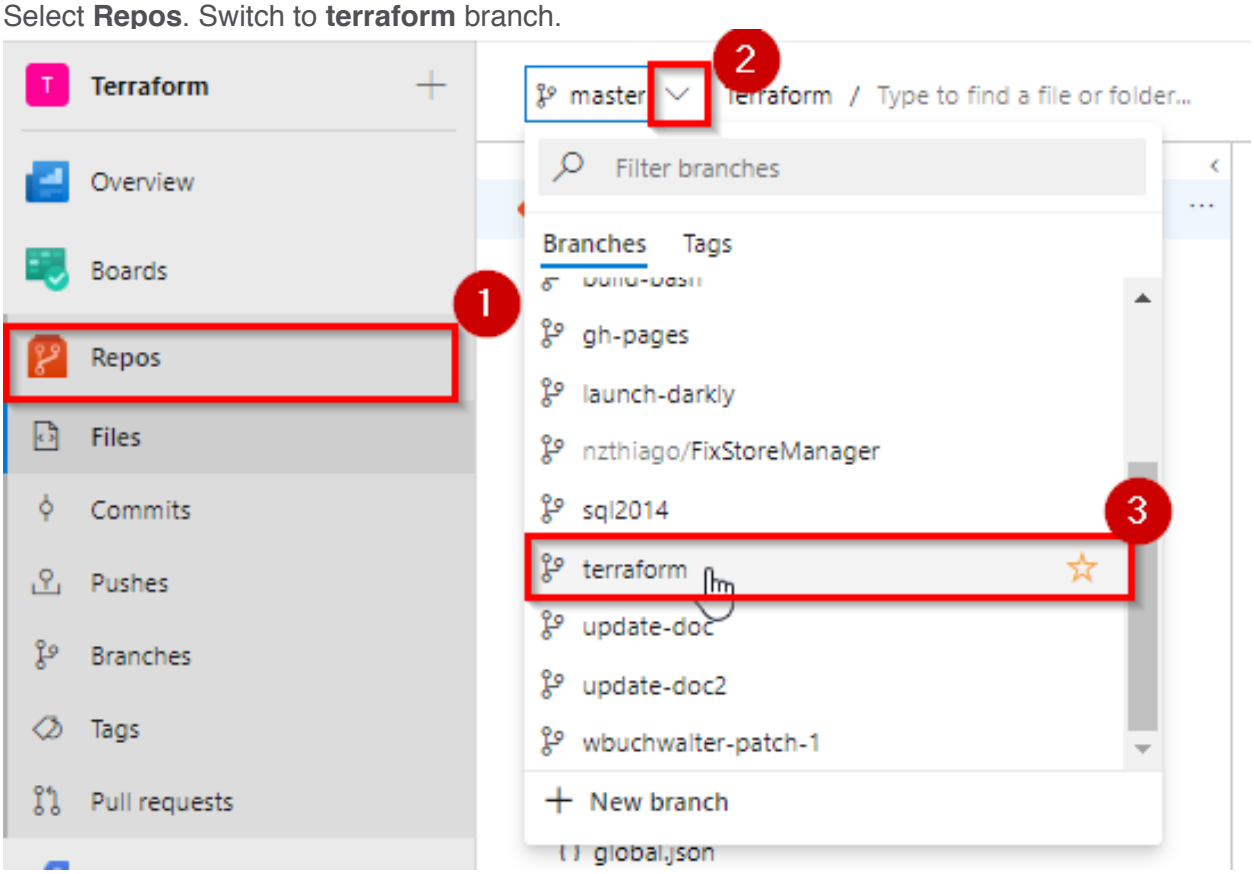


Create Project

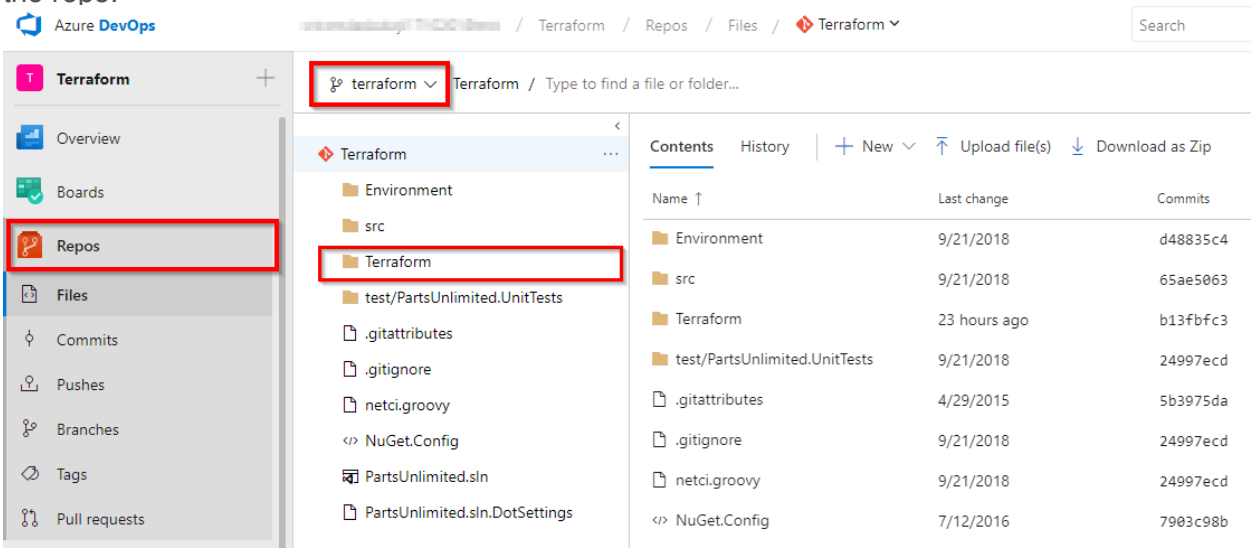
Step 4 - Examine the Terraform file (IaC) in your Source code

1. Navigate to the project you created above using Azure DevOps Demo Generator

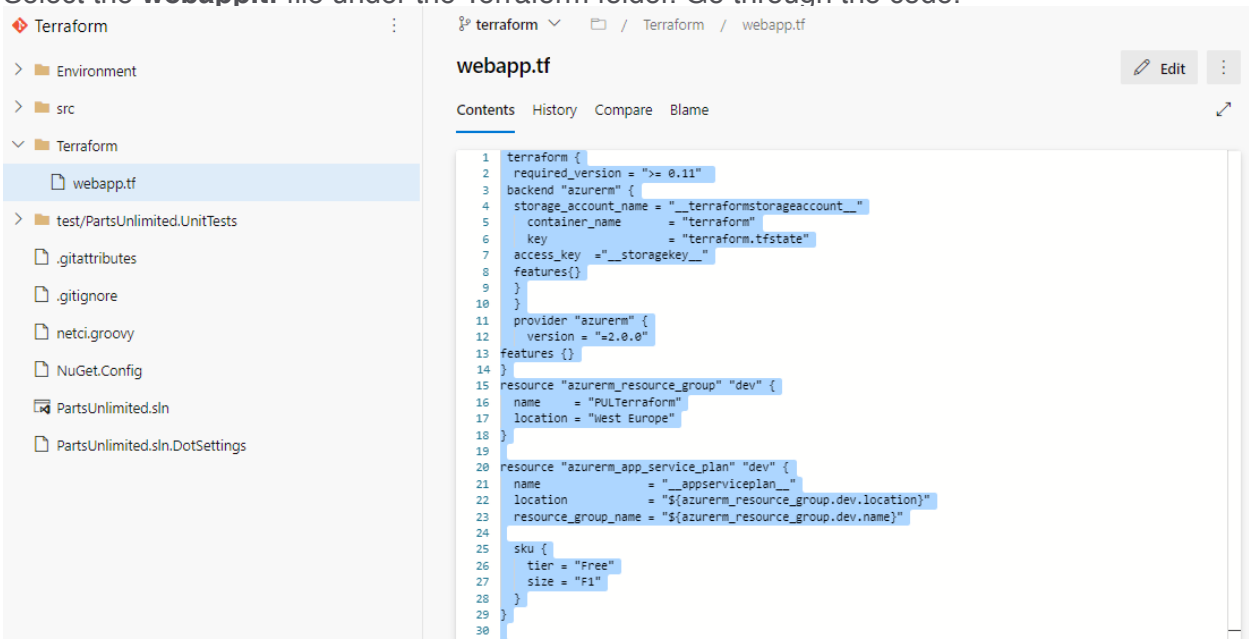
2. Select **Repos**. Switch to **terraform** branch.



3. Make sure that you are now on the **terraform** branch and **Terraform** folder is there in the repo.



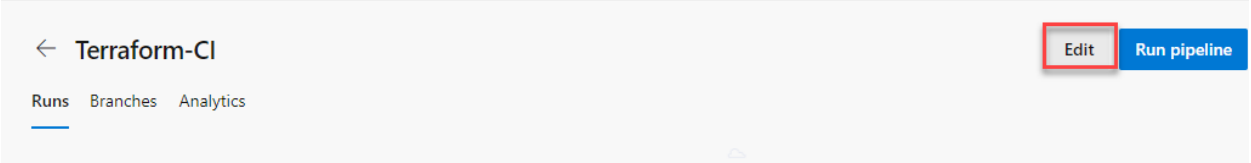
4. Select the **webapp.tf** file under the Terraform folder. Go through the code.



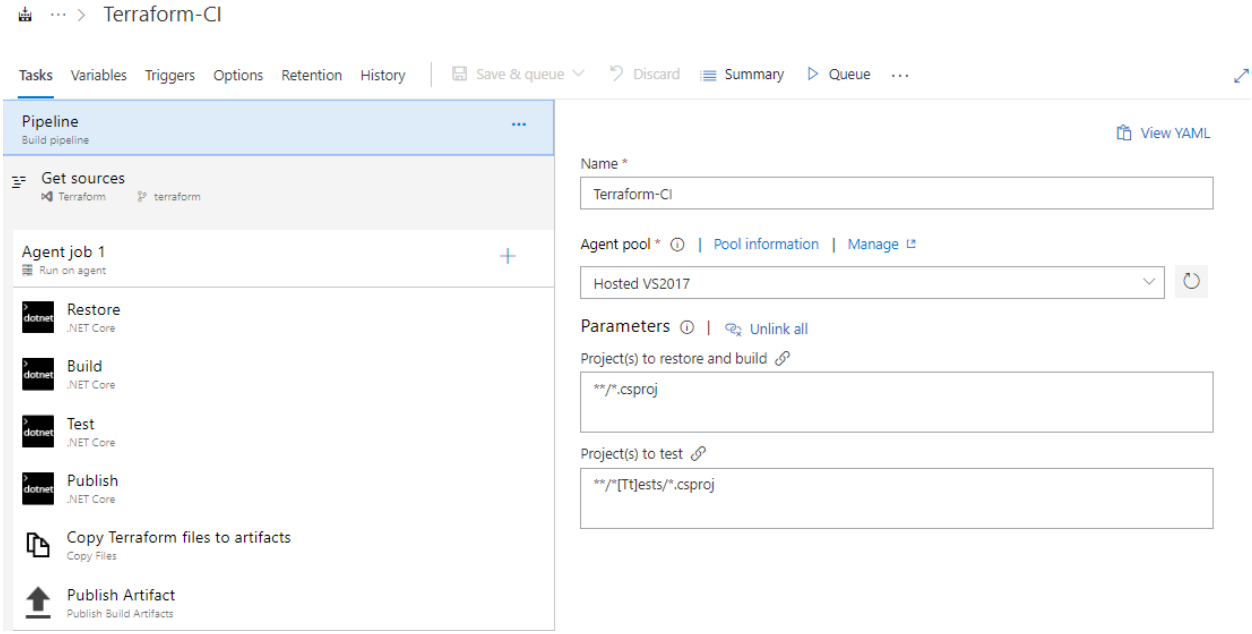
5. **webapp.tf** is a terraform configuration file. Terraform uses its own file format, called HCL (Hashicorp Configuration Language). This is very similar to YAML.
In this example, we want to deploy an Azure Resource group, App service plan and App service required to deploy the website. And we have added Terraform file (Infrastructure as Code) to source control repository in your Azure DevOps project which can deploy the required Azure resources.
If you would like to learn more about the terraform basics click here.

Step 5 - Build your application using Azure CI Pipeline

1. Navigate to **Pipelines –> Pipelines**. Select **Terraform-CI** and click **Edit**.

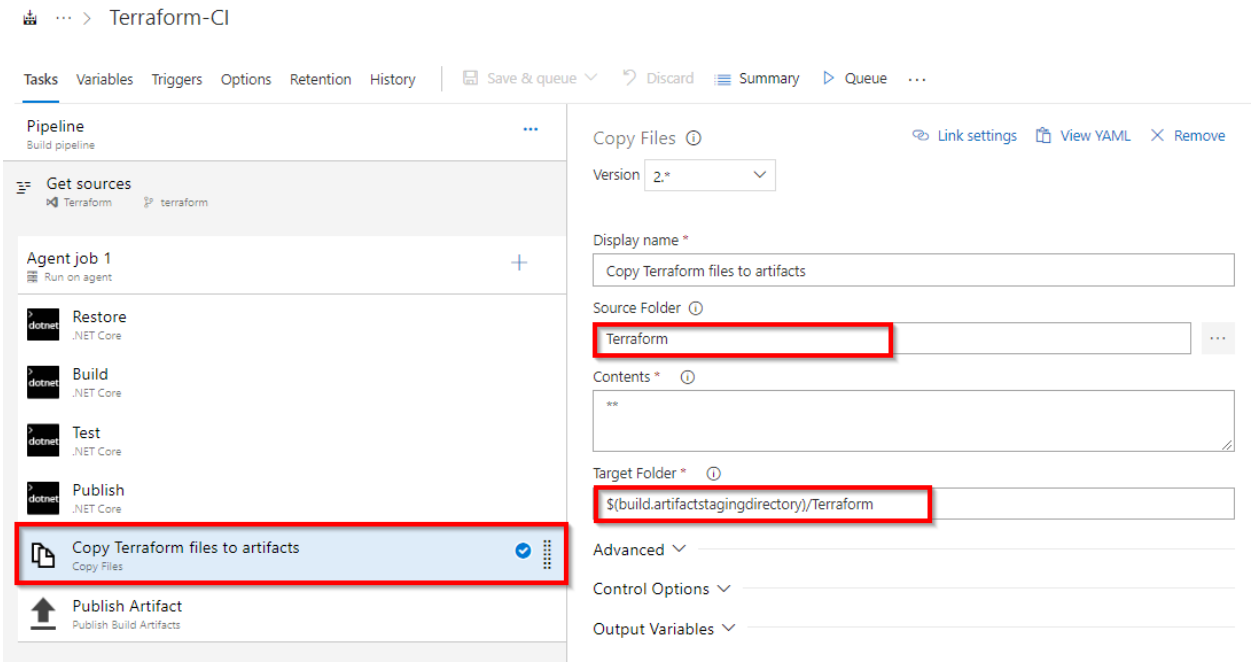


2. Your build pipeline will look like as below. This CI pipeline has tasks to compile .Net Core project. The dotnet tasks in the pipeline will restore dependencies, build, test and publish the build output into a zip file (package) which can be deployed to a web application.



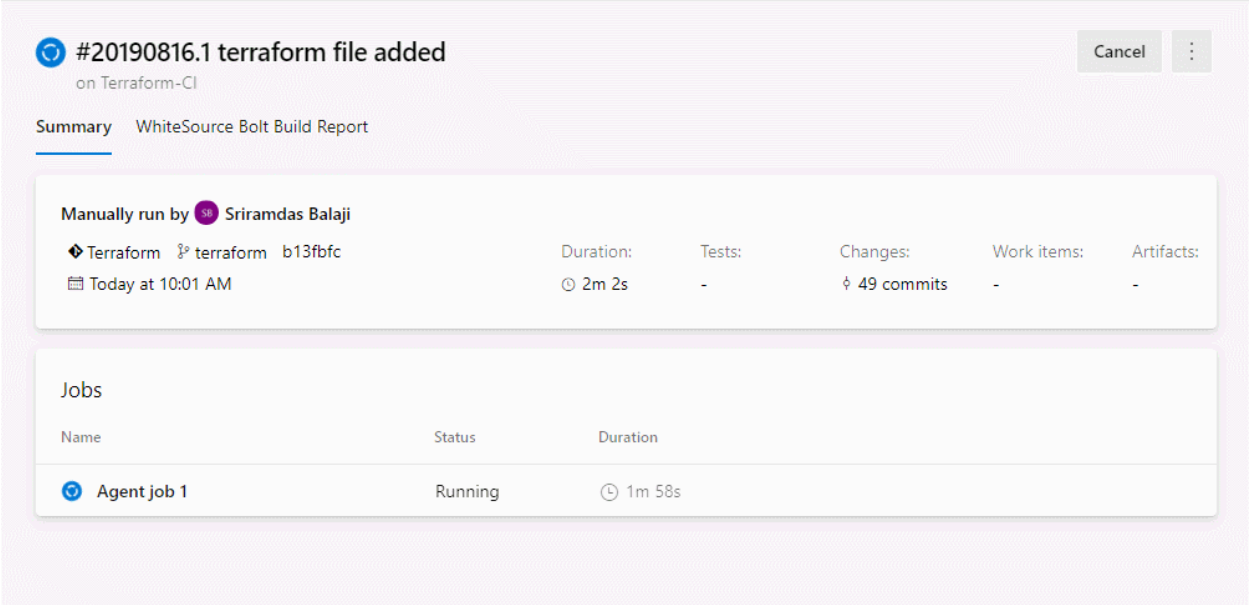
For more guidance on how to build .Net Core projects with Azure Pipelines see here.

3. In addition to the application build, we need to publish terraform files to build artifacts so that it will be available in CD pipeline. So we have added **Copy files** task to copy Terraform file to Artifacts directory.



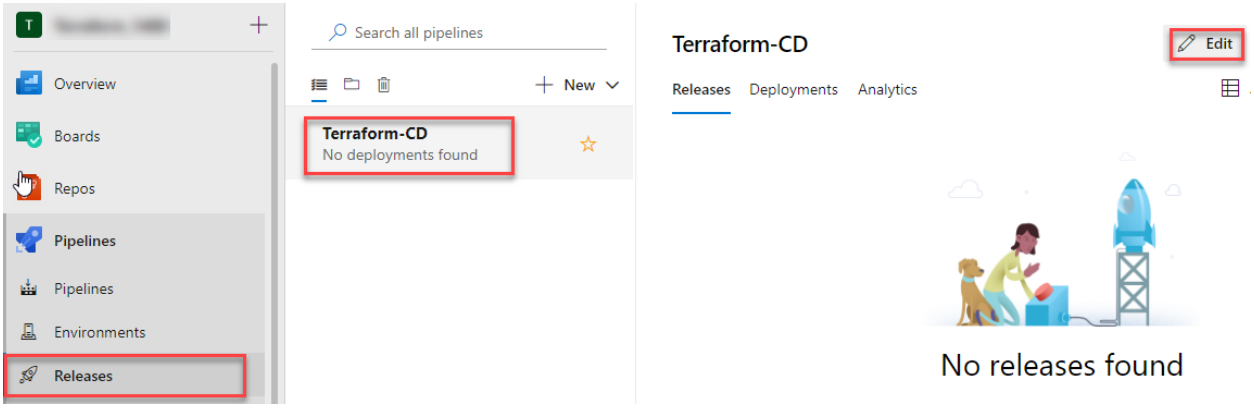
4. Click **Save & Queue** but only Click **Save**

5. Now click **Queue** to trigger the build. Once the build success, verify that the artifacts have **Terraform** folder and **PartsUnlimitedwebsite.zip** file in the drop.

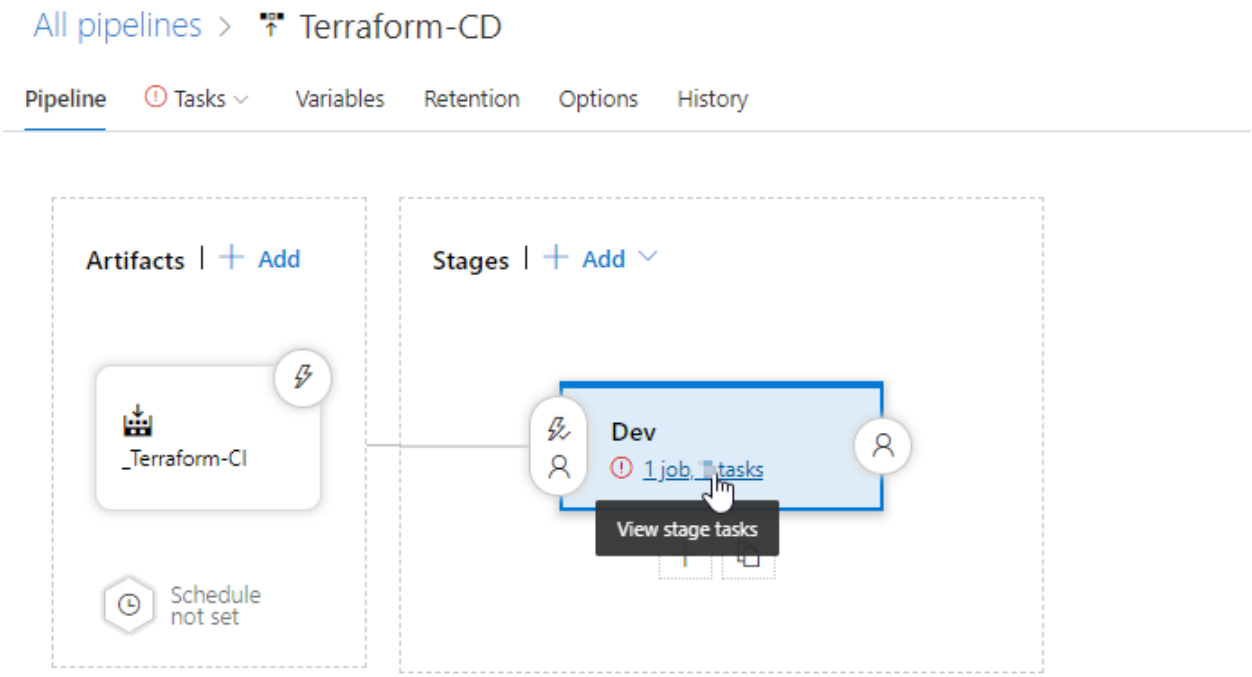


Step 6 - Deploy resources using Terraform (IaC) in Azure CD pipeline

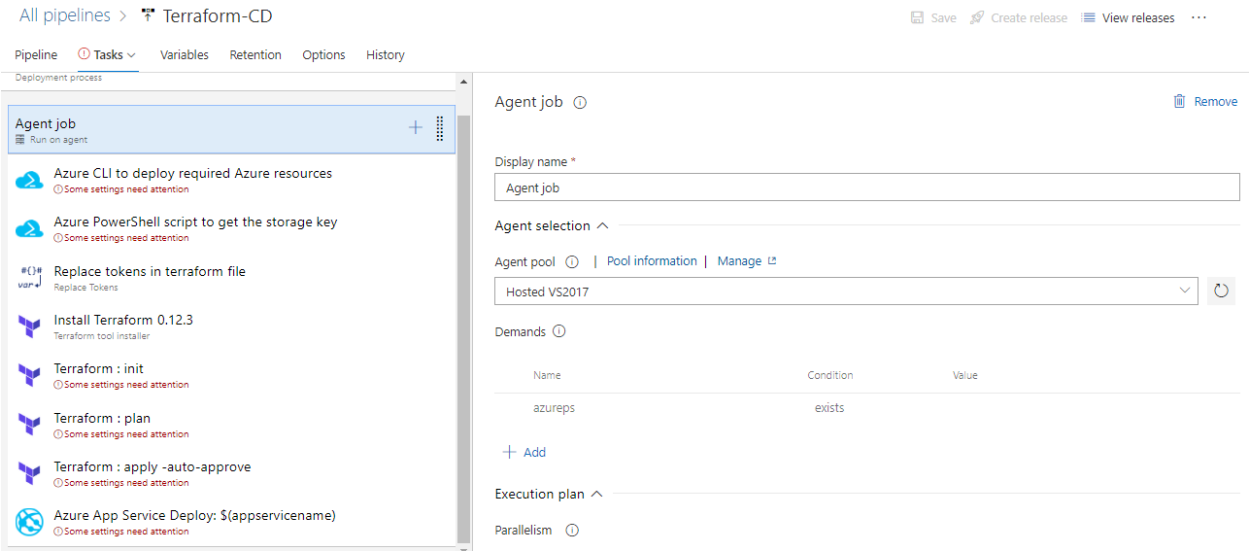
1. Navigate to **Pipelines –> Releases**. Select **Terraform-CD** and click **Edit**.



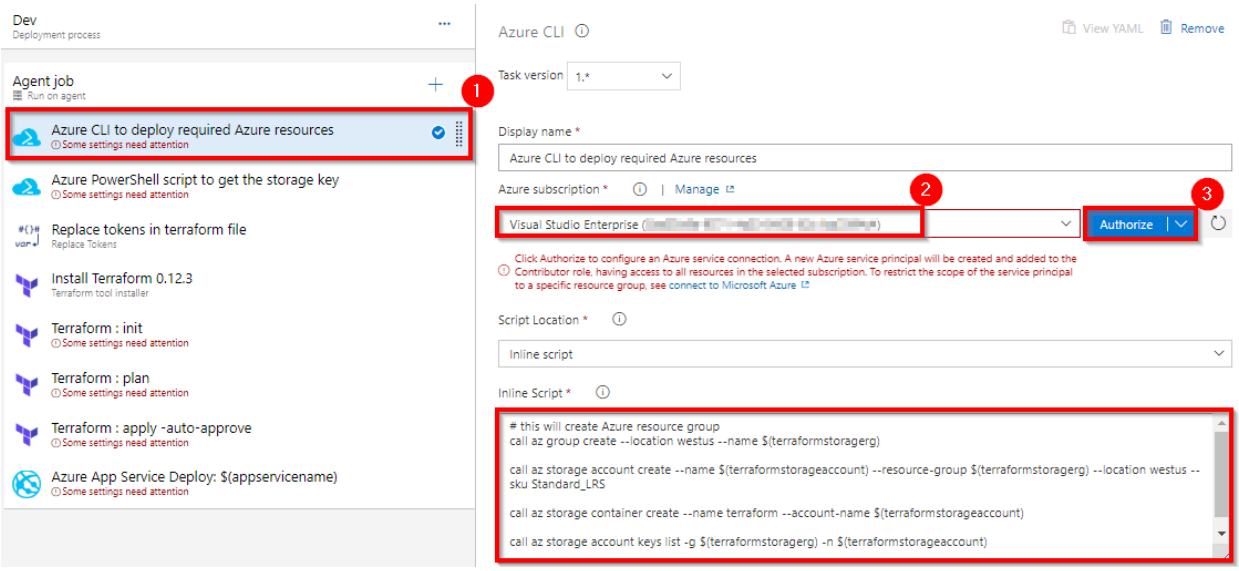
2. Select **Dev** stage and click **View stage tasks** to view the pipeline tasks.



3. Agent Job - **Select Azure Pipelines>Agent Pool: Windows 2019**

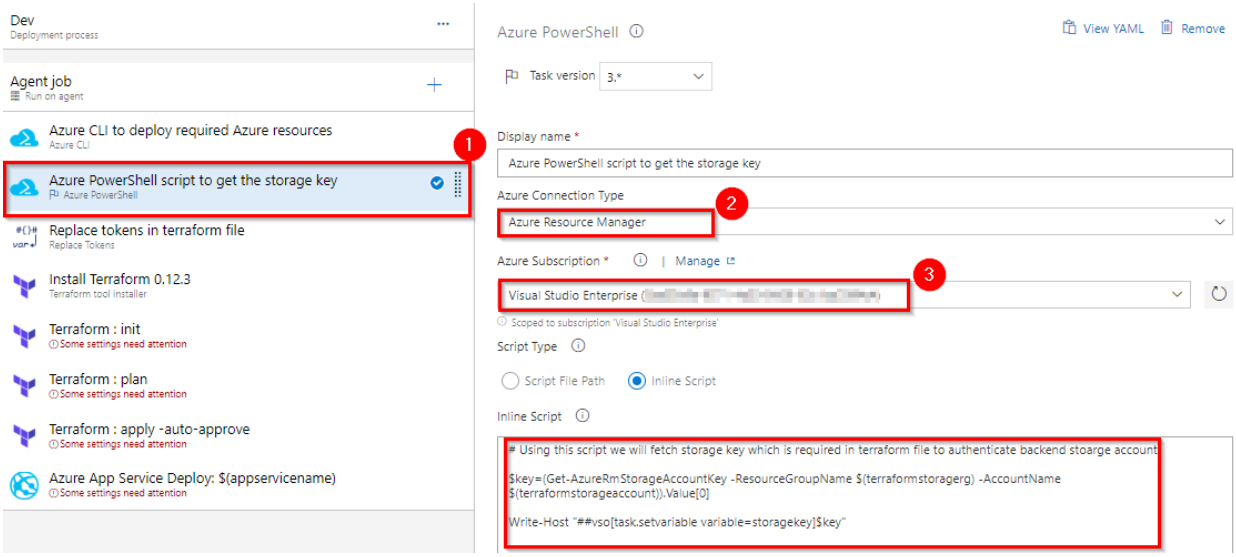


4. Select the **Azure CLI** task. Select the Azure subscription from the drop-down list and click **Authorize** to configure Azure service connection.



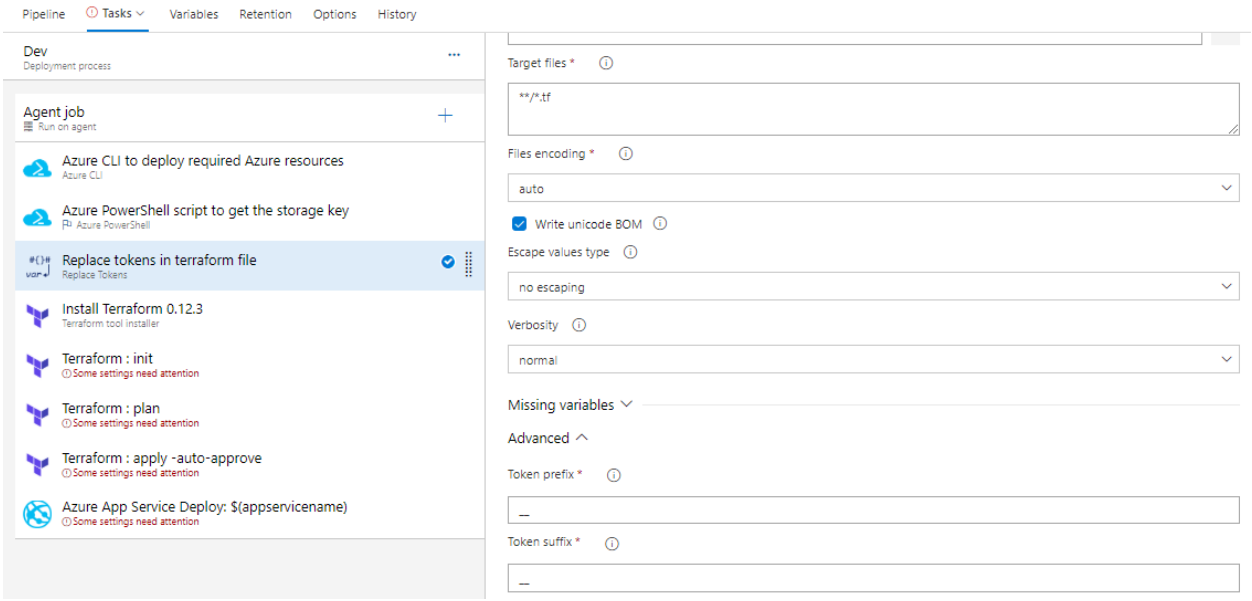
By default, Terraform stores state locally in a file named terraform.tfstate. When working with Terraform in a team, use of a local file makes Terraform usage complicated. With remote state, Terraform writes the state data to a remote data store. Here we are using Azure CLI task to create **Azure storage account** and **storage container** to store Terraform state. For more information on Terraform remote state click here

5. Select the **Azure PowerShell** task. Select Azure service connection from the drop-down.

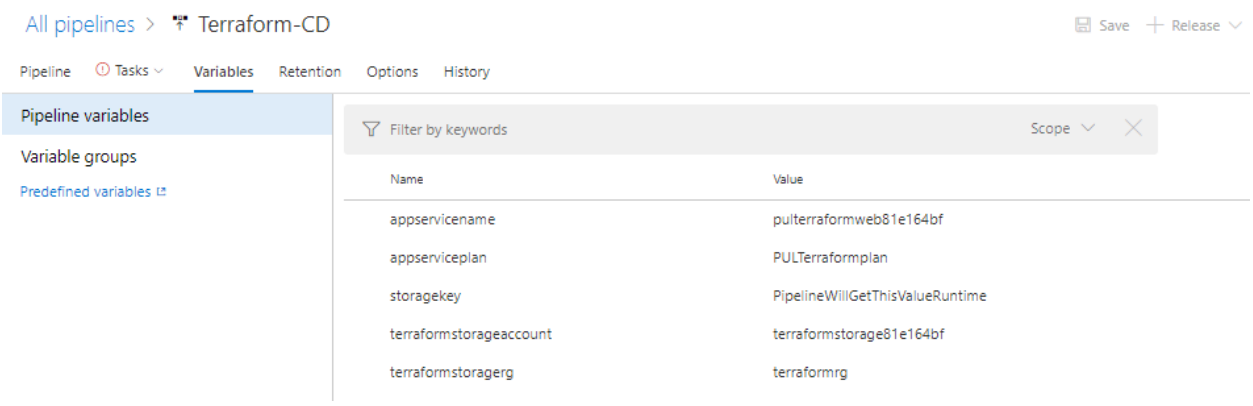


To configure the Terraform backend we need Storage account access key. Here we are using Azure PowerShell task to get the Access key of the storage account provisioned in the previous step.

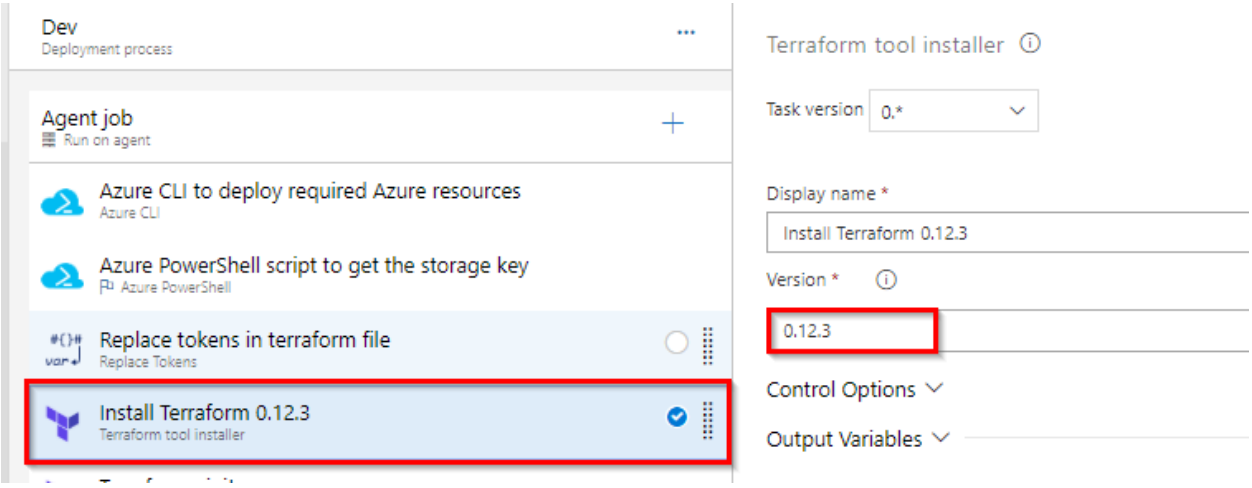
6. Select the **Replace tokens** task.



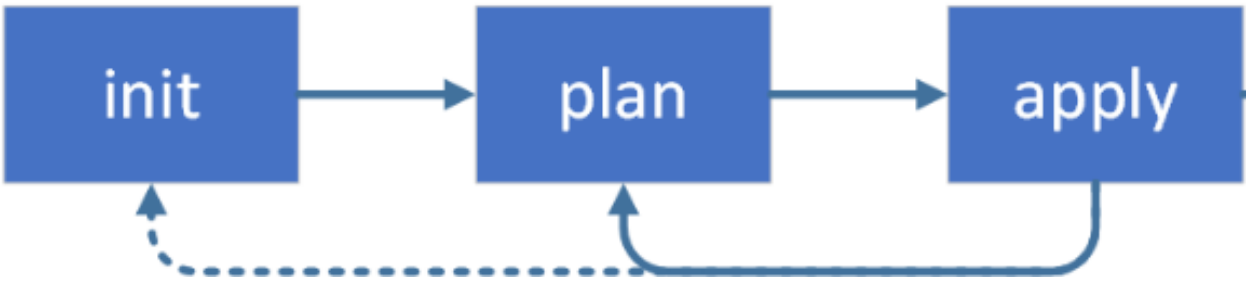
If you observe the **webapp.tf** file in **Exercise 1, Step 3** you will see there are few values are suffixed and prefixed with __. For example **__terraformstorageaccount__**.
Using **Replace tokens** task we will replace those values with the variable values defined in the release pipeline.



7. Terraform tool installer task is used to install a specified version of Terraform from the Internet or the tools cache and prepends it to the PATH of the Azure Pipelines Agent (hosted or private).



8. When running Terraform in automation, the focus is usually on the core plan/apply cycle. The main Terraform workflow is shown below:



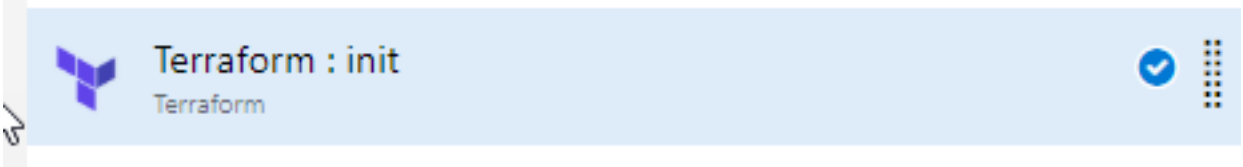
i. Initialize the Terraform working directory.

ii. Produce a plan for changing resources to match the current configuration.

iii. Apply the changes described by the plan.

The next Terraform tasks in your release pipeline help you to implement this workflow.

9. Select the **Terraform init** task. Select Azure service connection from the drop-down. And make sure to enter the container name as **terraform**. For the other task parameters information see here



Terraform ⓘ View YAML Remove

Task version 0.*

Display name *

Terraform : init

Provider * ⓘ

azurerm

Command * ⓘ

init

Configuration directory ⓘ

$(System.DefaultWorkingDirectory)/_Terraform-CI/drop/Terraform

AzureRM backend configuration

Azure subscription * ⓘ | Manage

Visual Studio Enterprise

Scoped to subscription 'Visual Studio Enterprise'

Resource group * ⓘ

$(terraformstoragerg)

Storage account * ⓘ

$(terraformstorageaccount)

Container * ⓘ

terraform

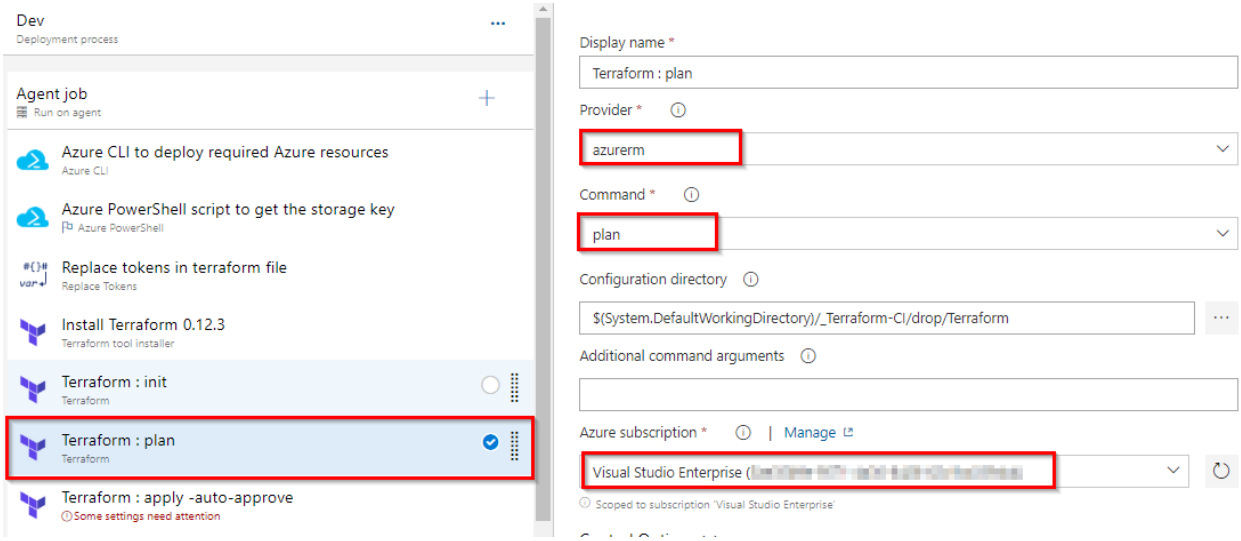
Key * ⓘ

terraform.tfstate

This task runs terraform init command. The terraform init command looks through all of

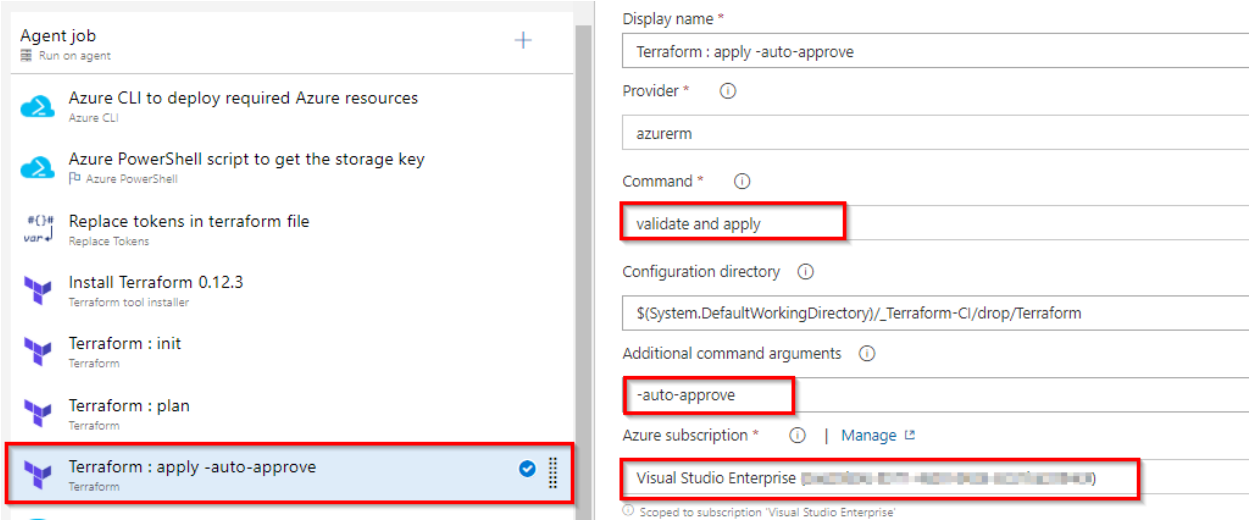
the *.tf files in the current working directory and automatically downloads any of the providers required for them. In this example, it will download Azure provider as we are going to deploy Azure resources. For more information about terraform init command click here

10. Select the **Terraform plan** task. Select Azure service connection from the drop-down.



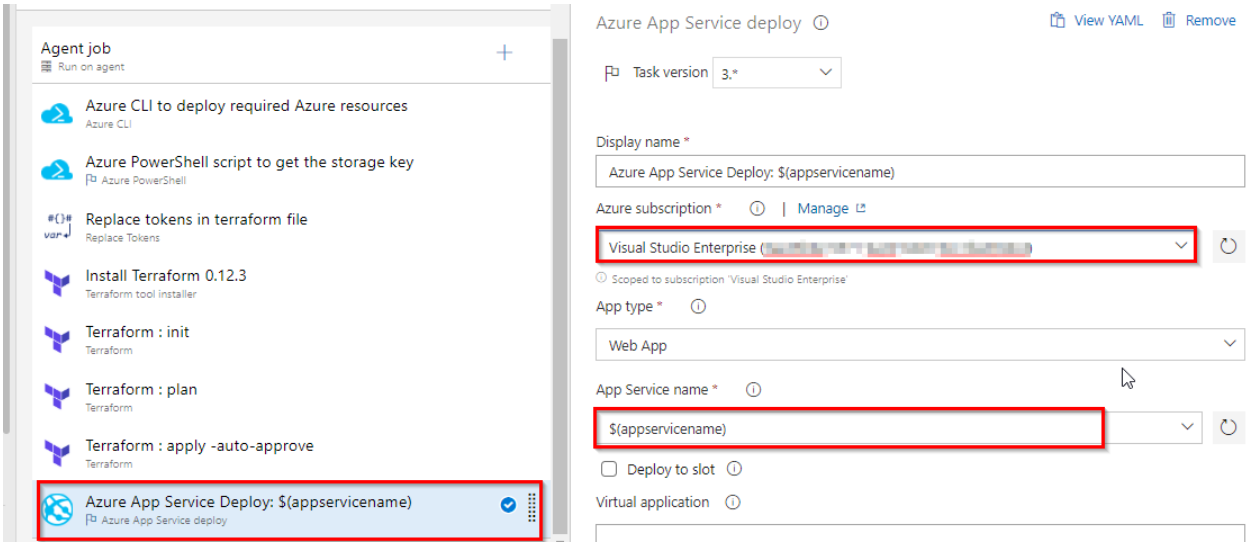
The terraform plan command is used to create an execution plan. Terraform determines what actions are necessary to achieve the desired state specified in the configuration files. This is a dry run and shows which actions will be made. For more information about terraform plan command click here

11. Select the **Terraform Apply** task. Select Azure service connection from the drop-down.



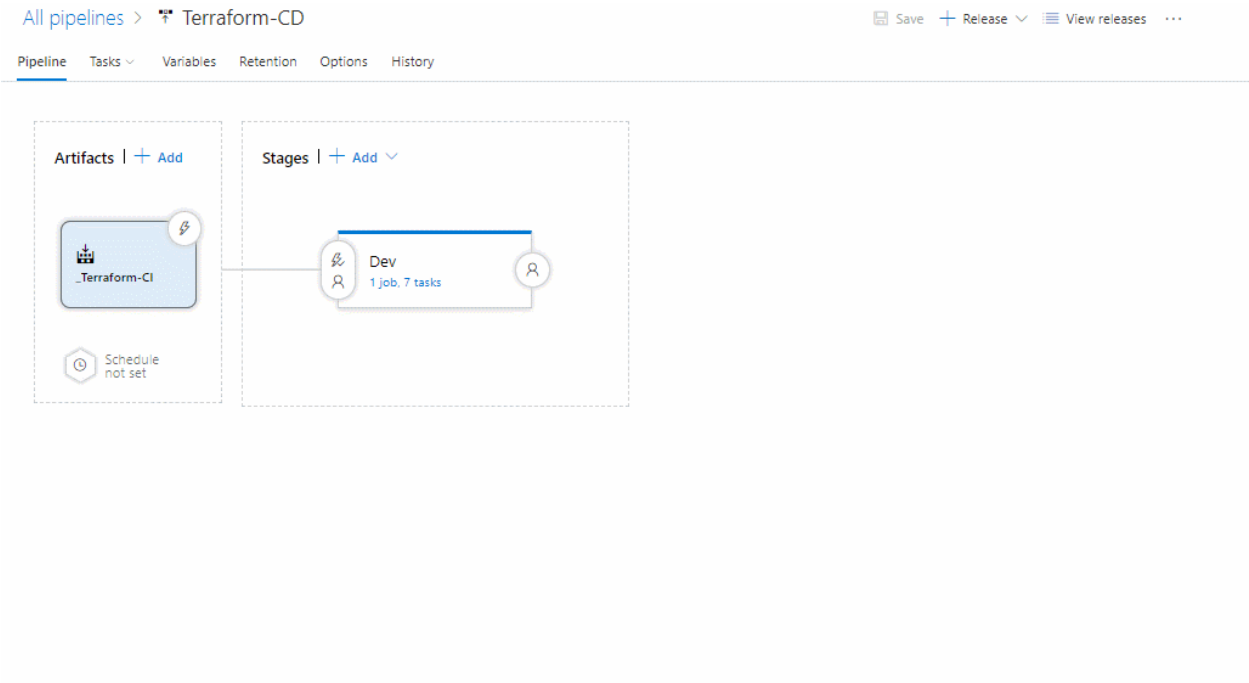
This task will run the terraform apply command to deploy the resources. By default, it will also prompt for confirmation that you want to apply those changes. Since we are automating the deployment we are adding auto-approve argument to not prompt for confirmation.

12. Select **Azure App Service Deploy** task. Select Azure service connection from the drop-down.



This task will deploy the PartsUnlimited package to Azure app service which is provisioned by Terraform tasks in previous steps.

13. Once you are done **Save** the changes and **Create a release**.



14. Once the release is success navigate to your Azure portal.

15. Search for **pulterraformweb** in App services.

16. Select **pulterraformweb-xxxx** and browse to view the application deployed.