

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

1. Log in to the Azure DevOps Demo Generator:  
<https://azuredevopsgenerator.azurewebsites.net/>

## Step 1 – Create Project

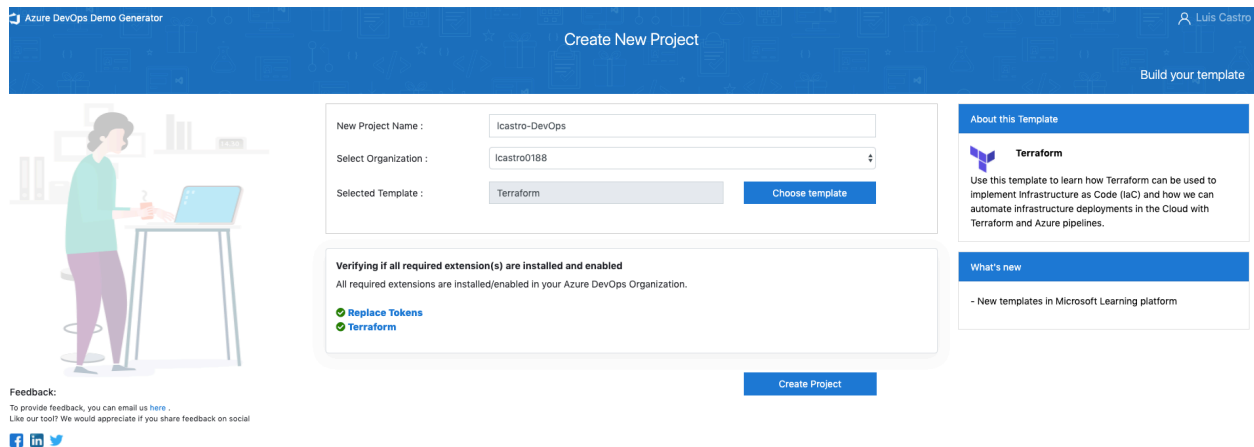
### Project Name

username-devops

### Select Organization

TBD

### Choose Template



Azure DevOps Demo Generator

Create New Project

Luis Castro

Build your template

New Project Name : Icastro-DevOps

Select Organization : Icastro0188

Selected Template : Terraform

Choose template

Verifying if all required extension(s) are installed and enabled

All required extensions are installed/enabled in your Azure DevOps Organization.

Replace Tokens

Terraform

Create Project

Feedback:

To provide feedback, you can email us [here](#).

Like our tool? We would appreciate if you share feedback on social

Facebook LinkedIn Twitter

About this Template

Terraform

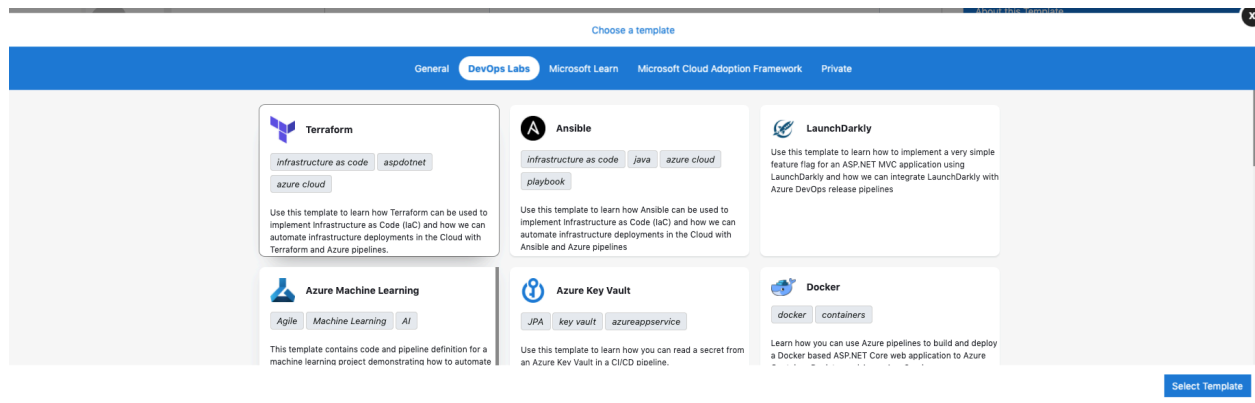
Use this template to learn how Terraform can be used to implement Infrastructure as Code (IaC) and how we can automate infrastructure deployments in the Cloud with Terraform and Azure pipelines.

What's new

- New templates in Microsoft Learning platform

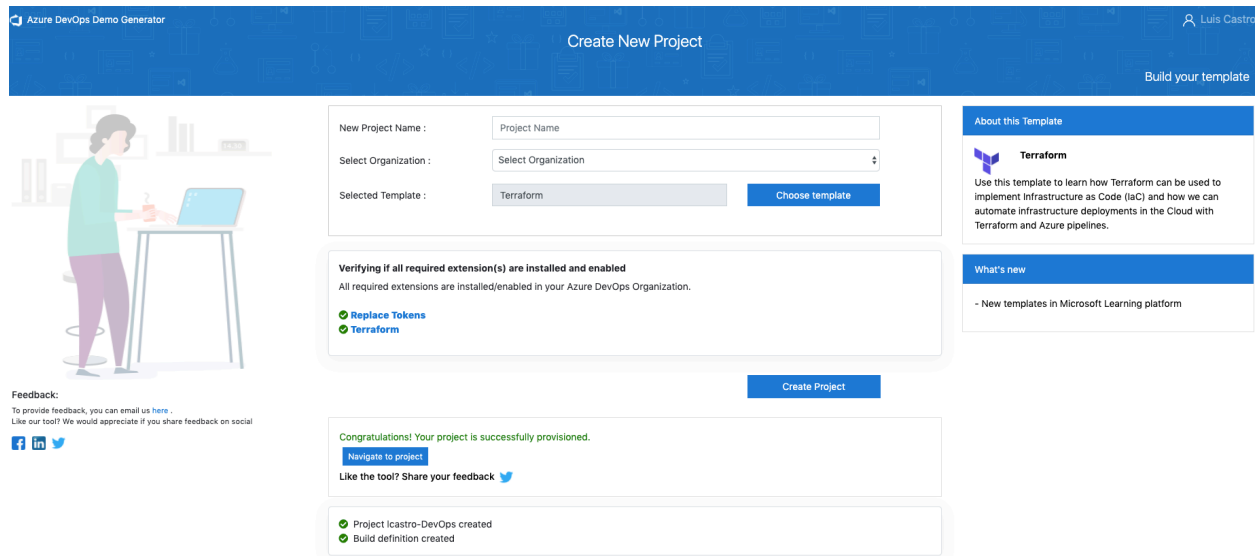
## Step 2 - Verify Terraform Version

## Select DevOps Labs and click Terraform>Select Template



### Step 3 - Verify Extensions

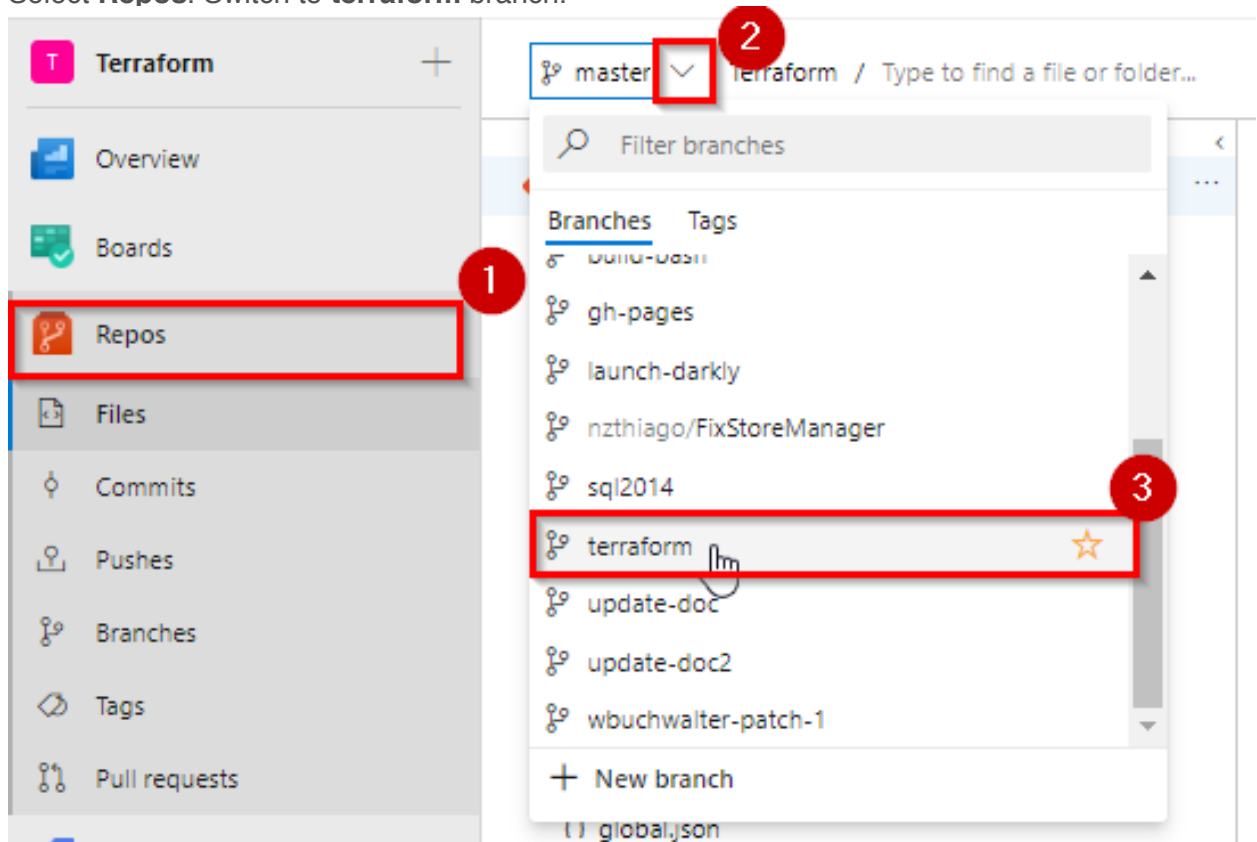
- Replace Tokens
- Terraform



## 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.



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

Azure DevOps interface showing the Terraform repository structure. The 'Repos' tab is selected in the left sidebar. The 'terraform' branch is selected in the top dropdown. The 'Terraform' folder is highlighted in the file explorer. The main pane shows the contents of the Terraform folder, including Environment, src, test/PartsUnlimited.UnitTests, .gitattributes, .gitignore, netci.groovy, NuGet.Config, PartsUnlimited.sln, and PartsUnlimited.sln.DotSettings.

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

Azure DevOps interface showing the **webapp.tf** file content. The file is open in the editor, showing the Terraform configuration for an Azure Resource Group, App Service Plan, and App Service.

```

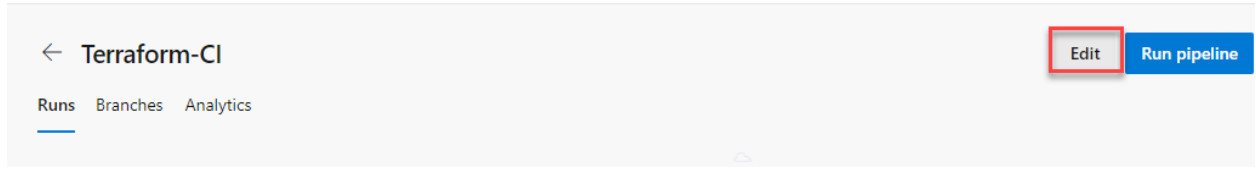
1 terraform {
2   required_version = ">= 0.11"
3   backend "azurerm" {
4     storage_account_name = "__terraformstorageaccount__"
5     container_name = "terraform"
6     key = "terraform.tfstate"
7     access_key = "__storagekey__"
8   }
9 }
10
11 provider "azurerm" {
12   version = "=2.0.0"
13   features {}
14 }
15 resource "azurerm_resource_group" "dev" {
16   name = "PULTerraform"
17   location = "West Europe"
18 }
19
20 resource "azurerm_app_service_plan" "dev" {
21   name = "__appserviceplan__"
22   location = "${azurerm_resource_group.dev.location}"
23   resource_group_name = "${azurerm_resource_group.dev.name}"
24 }
25 sku {
26   tier = "Free"
27   size = "F1"
28 }
29
30

```

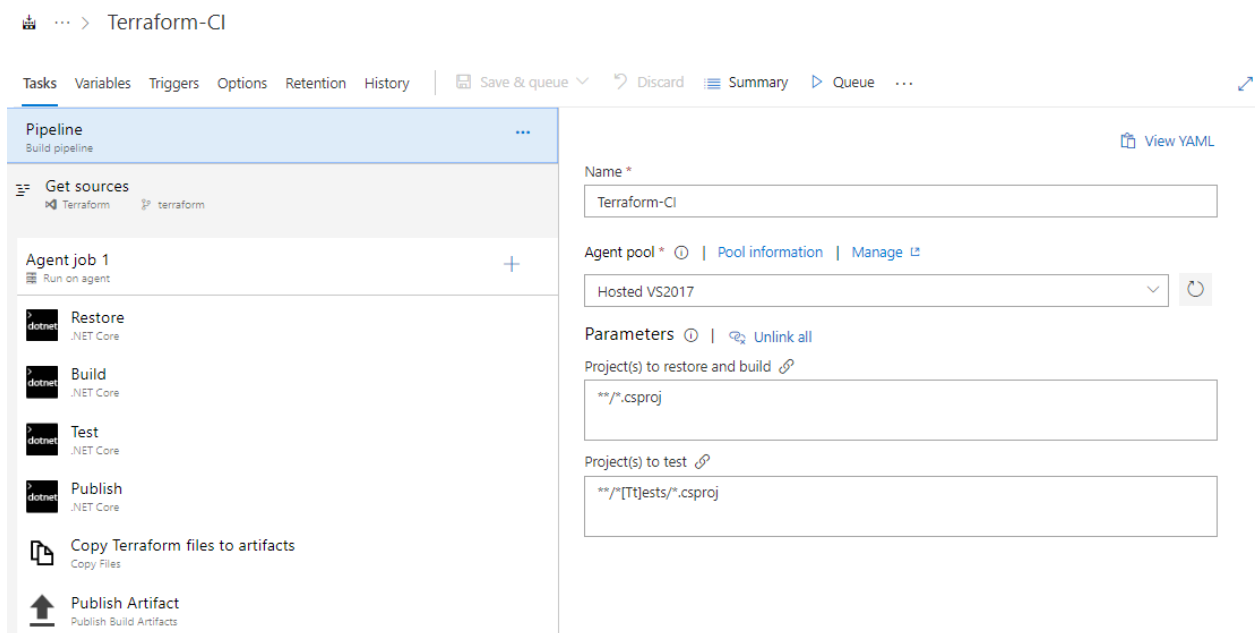
- webapp.tf** is a terraform configuration file. Terraform uses its own file format, called HCL (Hashicorp Configuration Language). This is very similar to YAML. In this example, we want to deploy an Azure Resource group, App service plan and App service required to deploy the website. And we have added 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**.



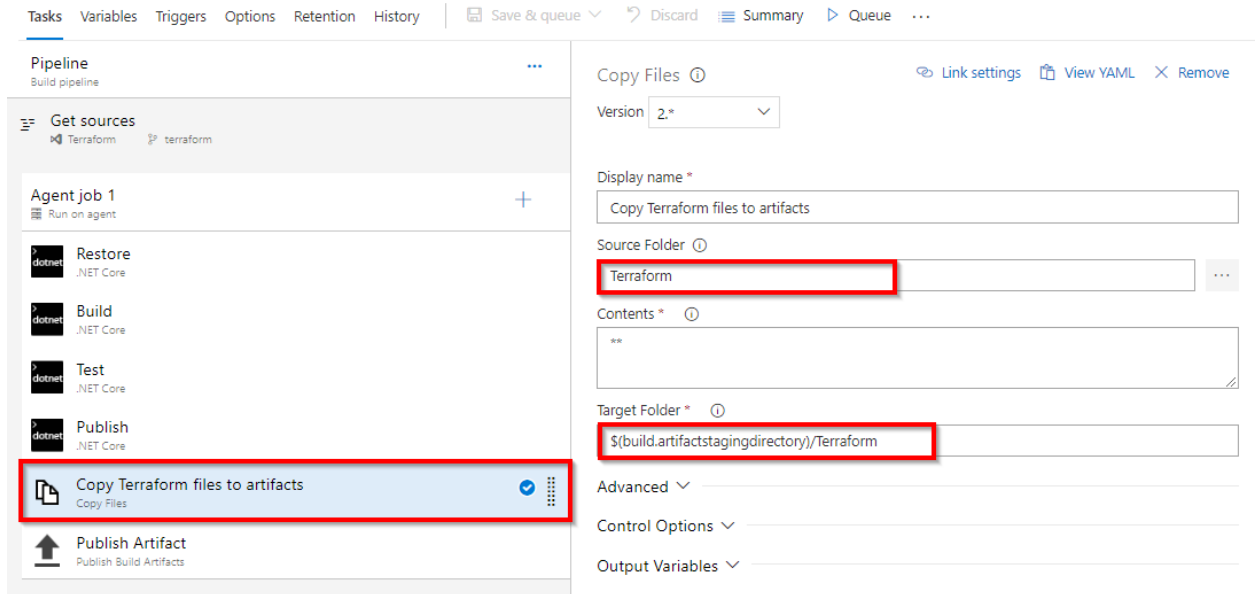
2. Your build pipeline will look like as below. This CI pipeline has tasks to compile .Net Core project. The **dotnet** tasks in the pipeline will restore dependencies, build, test and publish the build output into a zip file (package) which can be deployed to a web application.



For more guidance on how to build .Net Core projects with Azure Pipelines see [here](#).

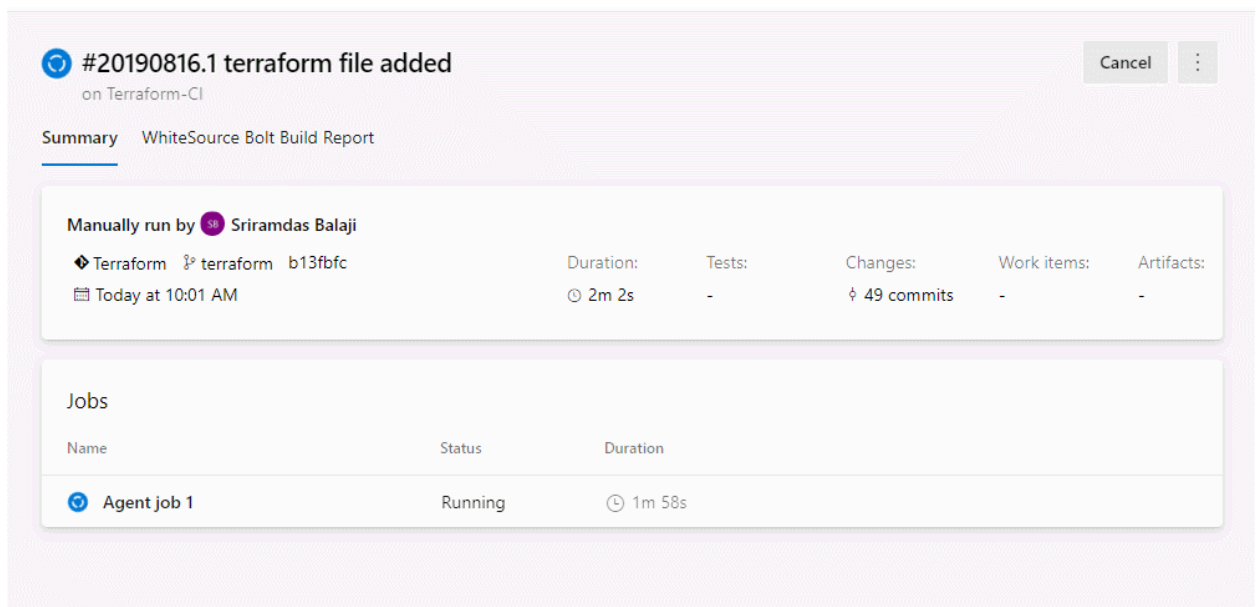
- In addition to the application build, we need to publish terraform files to build artifacts so that it will be available in CD pipeline. So we have added **Copy files** task to copy Terraform file to Artifacts directory.

🏠 ... > Terraform-CI



The screenshot shows the 'Copy Files' task configuration in the Azure DevOps pipeline editor. The task is named 'Copy Terraform files to artifacts' and is of type 'Copy Files'. The 'Source Folder' is set to 'Terraform' and the 'Target Folder' is set to '\$(build.artifactstagingdirectory)/Terraform'. The 'Contents' field is set to '\*\*'. The task is highlighted with a red box in the task list on the left.

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

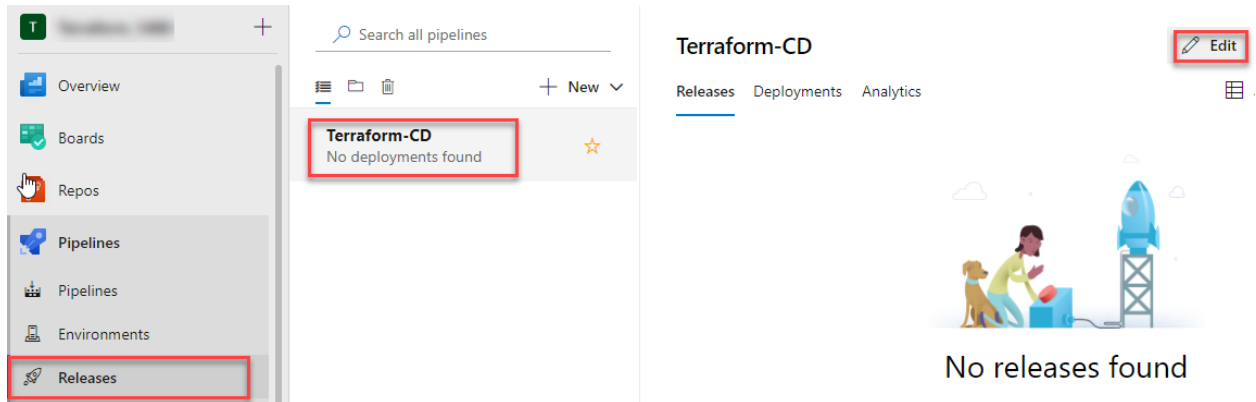


The screenshot shows the build summary for the 'Terraform-CI' build. The build was manually run by Sriramdas Balaji. The summary shows the build duration as 2m 2s, with 49 commits. The build is currently running.

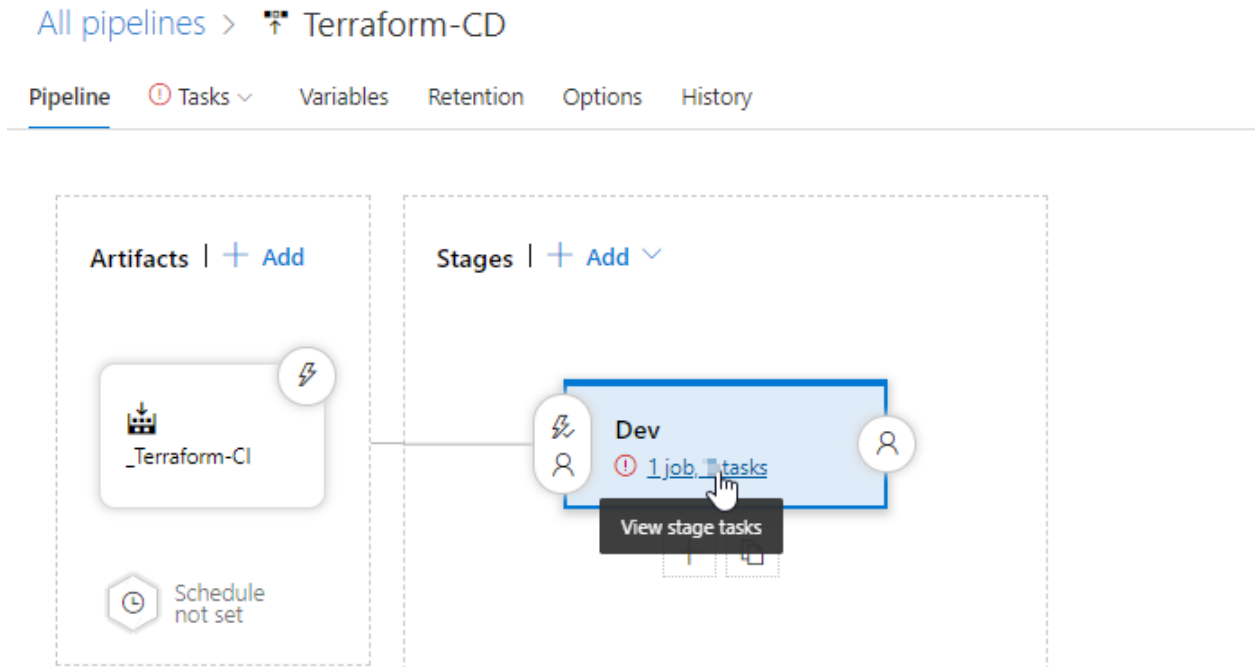
Name	Status	Duration
Agent job 1	Running	1m 58s

## 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

Agent job

Display name \*

Agent selection ^

Agent pool | Pool information | Manage

Hosted VS2017

Demands

Name	Condition	Value
azureps	exists	

+ Add

Execution plan ^

Parallelism

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

Azure CLI

Task version 1.\*

Display name \*

Azure subscription \* | Manage

Visual Studio Enterprise

Authorize

Click Authorize to configure an Azure service connection. A new Azure service principal will be created and added to the Contributor role, having access to all resources in the selected subscription. To restrict the scope of the service principal to a specific resource group, see [connect to Microsoft Azure](#)

Script Location \*

Inline script

Inline Script \*

```
# this will create Azure resource group
call az group create --location westus --name $(terraformstorage)

call az storage account create --name $(terraformstorageaccount) --resource-group $(terraformstorage) --location westus --sku Standard_LRS

call az storage container create --name terraform --account-name $(terraformstorageaccount)

call az storage account keys list -g $(terraformstorage) -n $(terraformstorageaccount)
```

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.

The screenshot shows the configuration for the 'Azure PowerShell' task in an Azure DevOps pipeline. The task is selected in the left-hand list. The configuration panel on the right shows the following settings:

- Task version:** 3.\*
- Display name:** Azure PowerShell script to get the storage key
- Azure Connection Type:** Azure Resource Manager
- Azure Subscription:** Visual Studio Enterprise
- Script Type:** ☒ Inline Script
- Inline Script:**

```
# Using this script we will fetch storage key which is required in terraform file to authenticate backend storage account.
$key=(Get-AzureRmStorageAccountKey -ResourceGroupName $(terraformstoragegerg) -AccountName $(terraformstorageaccount)).Value[0]
Write-Host "##vso[task.setvariable variable=storagekey]$key"
```

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.

The screenshot shows the configuration for the 'Replace tokens' task in an Azure DevOps pipeline. The task is selected in the left-hand list. The configuration panel on the right shows the following settings:

- Target files:** \*\*/\*.tf
- Files encoding:** auto
- Write unicode BOM:** ☒
- Escape values type:** no escaping
- Verbosity:** normal
- Missing variables:** (empty)
- Advanced:**
  - Token prefix:** \_
  - Token suffix:** \_

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.

All pipelines > Terraform-CD Save + Release

Pipeline Tasks Variables Retention Options History

Pipeline variables

Variable groups

Predefined variables

Filter by keywords Scope

Name	Value
appservice-name	pulterraformweb81e164bf
appserviceplan	PULTerraformplan
storagekey	PipelineWillGetThisValueRuntime
terraformstorageaccount	terraformstorage81e164bf
terraformstoragereg	terraformrg

- 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).

Dev  
Deployment process

Agent job  
Run on agent

- Azure CLI to deploy required Azure resources  
Azure CLI
- Azure PowerShell script to get the storage key  
Azure PowerShell
- Replace tokens in terraform file  
Replace Tokens
- Install Terraform 0.12.3**  
Terraform tool installer

Terraform tool installer ⓘ

Task version 0.\*

Display name \*  
Install Terraform 0.12.3

Version \* ⓘ  
**0.12.3**

Control Options

Output Variables

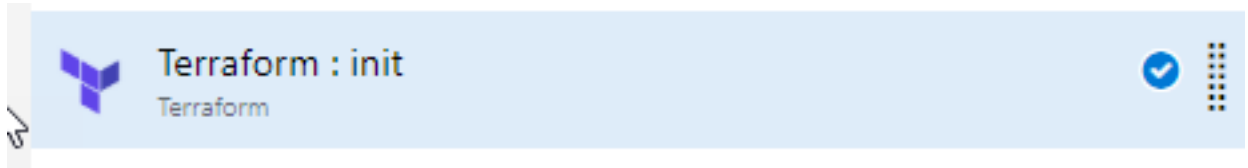
8. When running Terraform in automation, the focus is usually on the core plan/apply cycle. The main Terraform workflow is shown below:





- 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 ▼



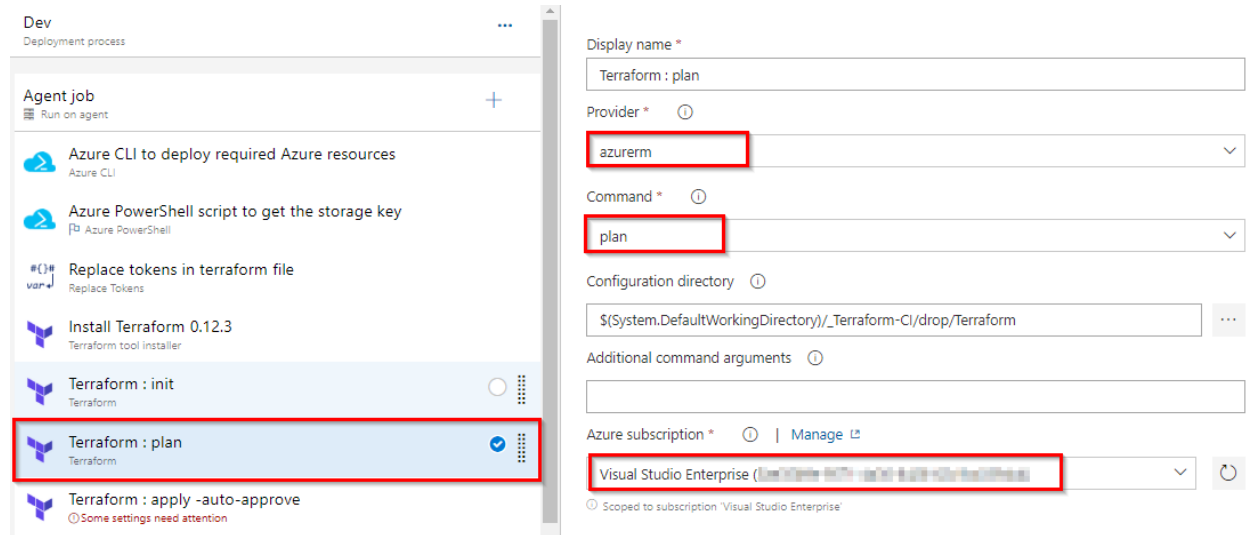
Key \* ⓘ

terraform.tfstate

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

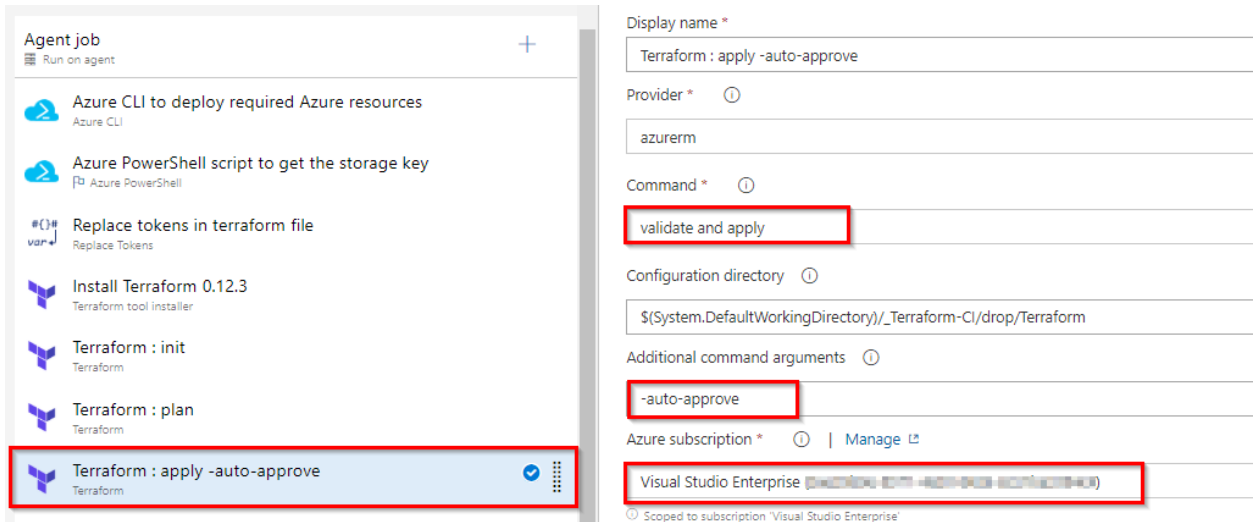
the \*.tf files in the current working directory and automatically downloads any of the providers required for them. In this example, it will download [Azure provider](#) as we are going to deploy Azure resources. For more information about [terraform init](#) command click [here](#)

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.

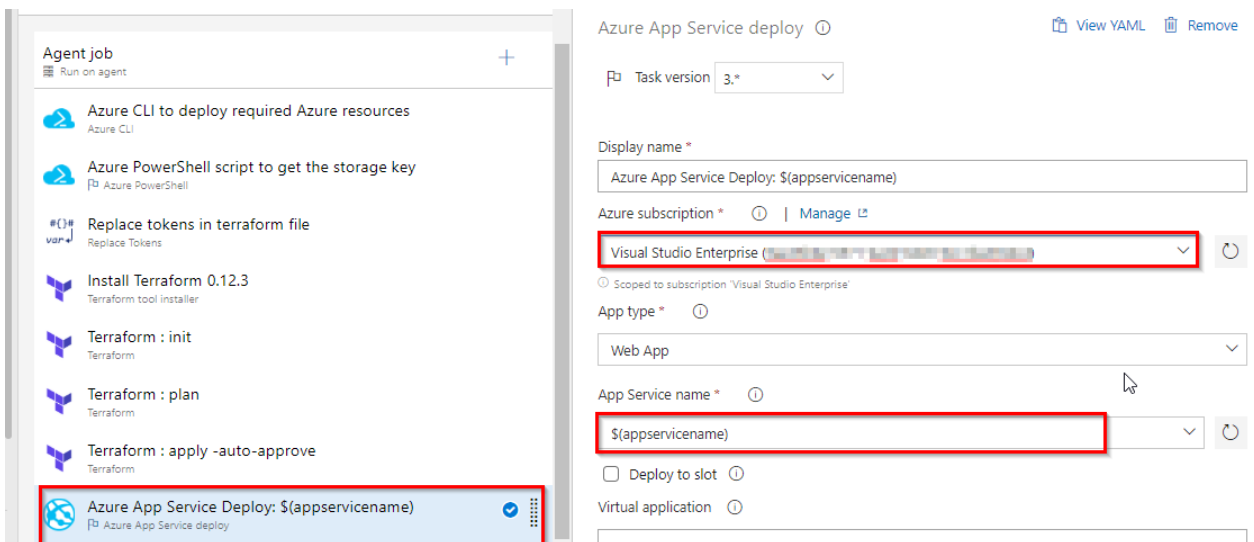


The screenshot shows the configuration for the 'Terraform : apply -auto-approve' task. The task is selected in the left-hand list. The configuration details on the right are as follows:

- Display name \***: Terraform : apply -auto-approve
- Provider \***: azurearm
- Command \***: validate and apply
- Configuration directory**: \$(System.DefaultWorkingDirectory)/\_Terraform-CI/drop/Terraform
- Additional command arguments**: -auto-approve
- Azure subscription \***: Visual Studio Enterprise (scoped to subscription 'Visual Studio Enterprise')

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.



The screenshot shows the configuration for the 'Azure App Service Deploy: \$(appservicename)' task. The task is selected in the left-hand list. The configuration details on the right are as follows:

- Task version**: 3.\*
- Display name \***: Azure App Service Deploy: \$(appservicename)
- Azure subscription \***: Visual Studio Enterprise (scoped to subscription 'Visual Studio Enterprise')
- App type \***: Web App
- App Service name \***: \$(appservicename)
- Deploy to slot**: ☐
- Virtual application**: ☐

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

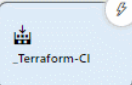
13. Once you are done **Save** the changes and **Create a release**.

[All pipelines](#) > Terraform-CD

Save + Release View releases ...

Pipeline Tasks Variables Retention Options History


Artifacts | + Add



\_Terraform-CD

Schedule not set

Stages | + Add



Dev

1 job, 7 tasks



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.

