

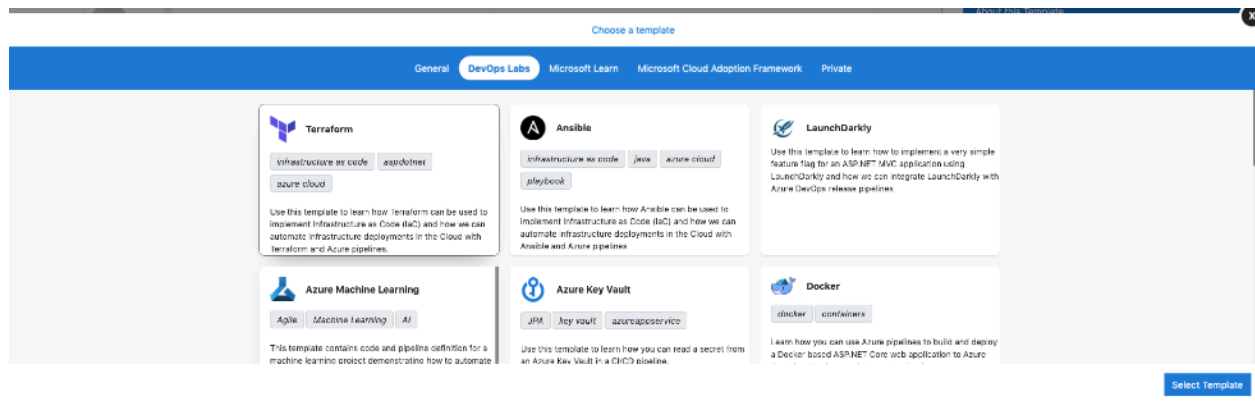
1. Log in to the Azure DevOps Demo Generator:  
<https://azuredevopsdemogenerator.azurewebsites.net/>

## Choose Template

100 250 1000 10000 Hz      WT   0   0.1   0.4 0.50 0.41   1   1.   1   1

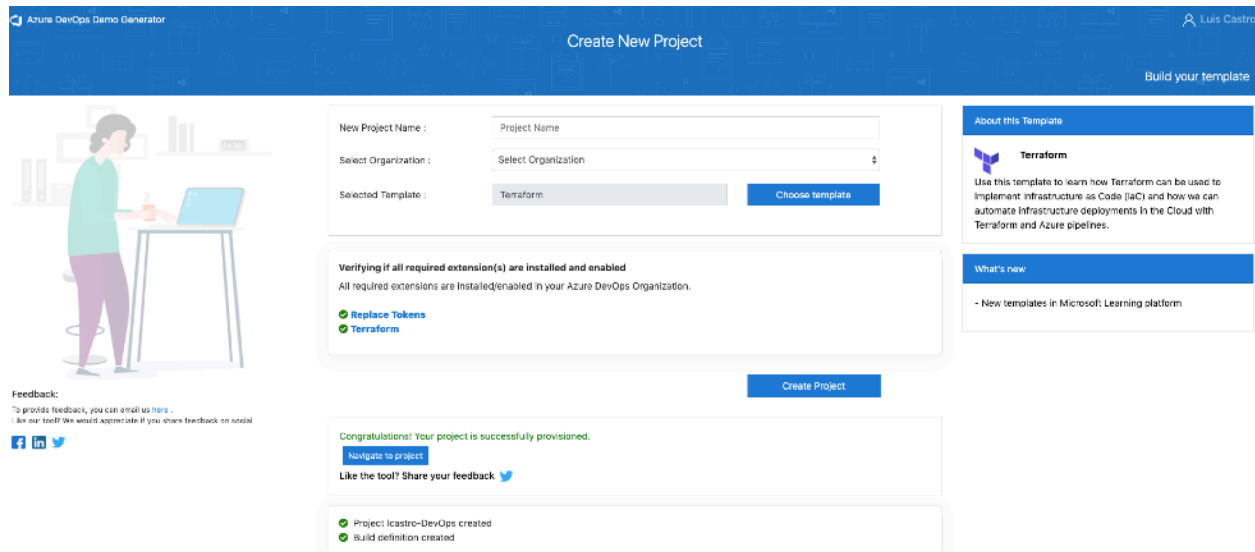
## Step 2 - Verify Terraform Version

## Select DevOps Labs and click Terraform>Select Template



### Step 3 - Verify Extensions

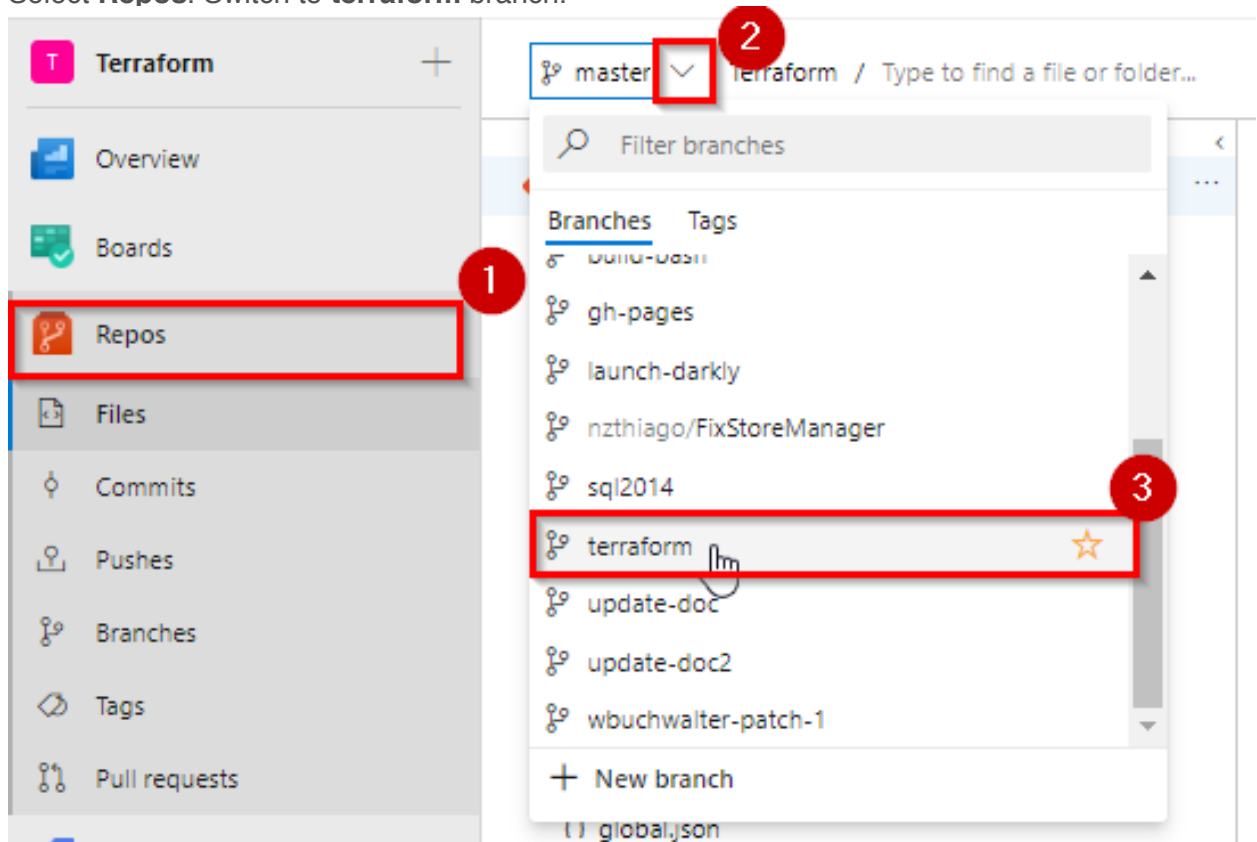
- Replace Tokens
- Terraform



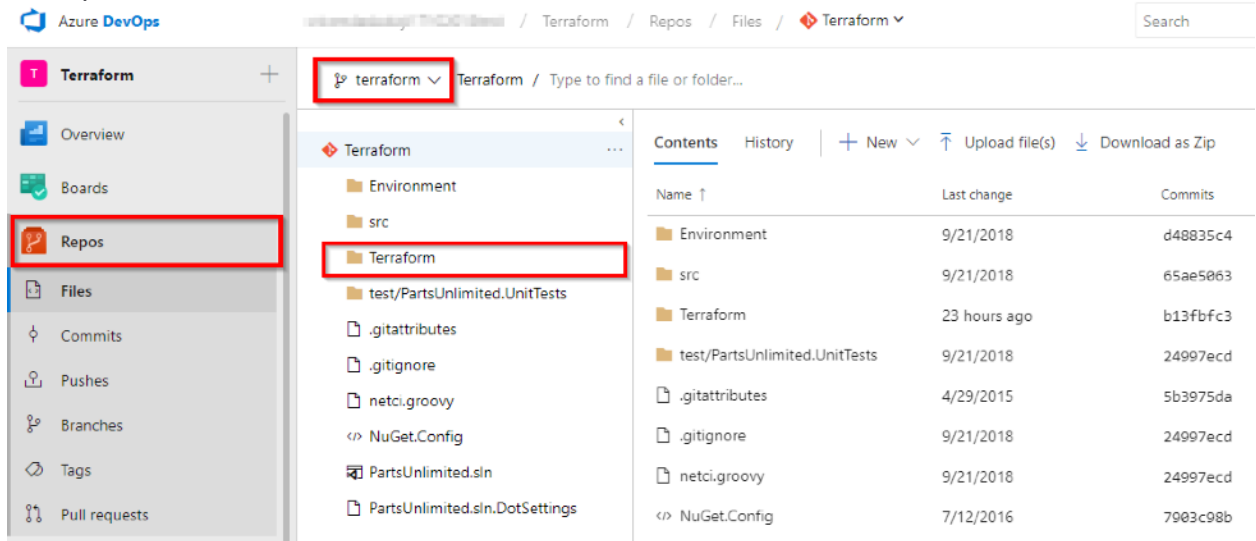
## Create Project

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

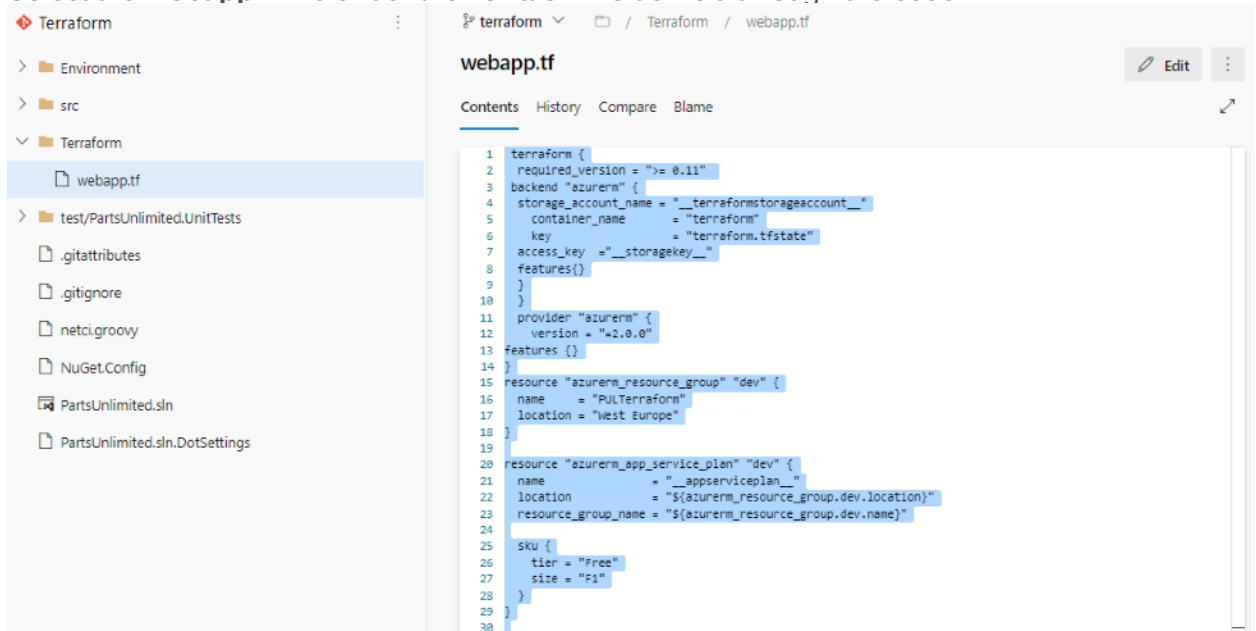
1. Navigate to the project you created above using [Azure DevOps Demo Generator](#)
2. Select **Repos**. Switch to **terraform** branch.



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



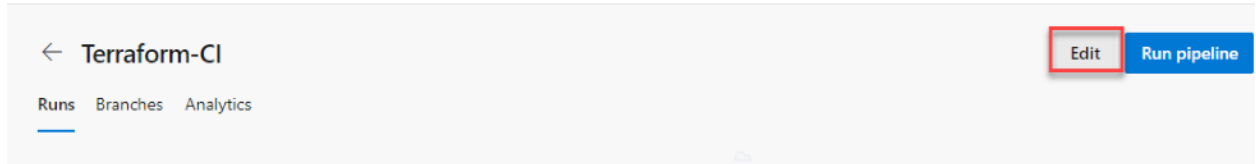
- Select the **webapp.tf** file under the Terraform folder. Go through the code.



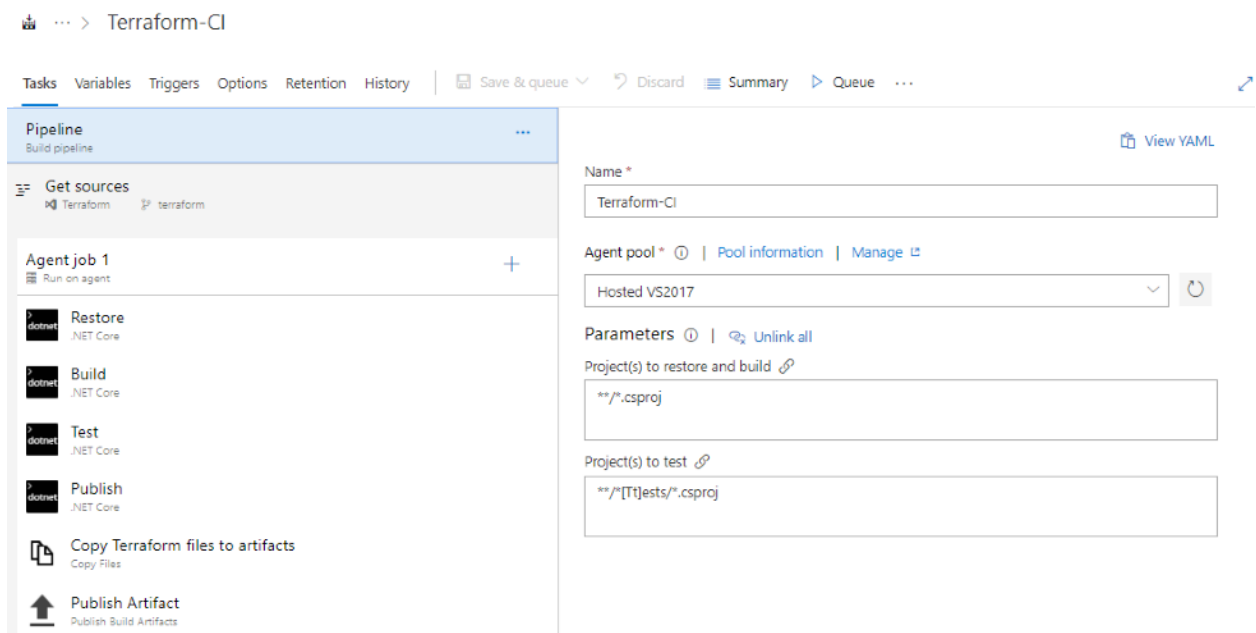
- webapp.tf** is a terraform configuration file. Terraform uses its own file format, called HCL (Hashicorp Configuration Language). This is very similar to YAML. In this example, we want to deploy an Azure Resource group, App service plan and App service required to deploy the website. And we have added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

 ... > Terraform-Cl

TasksVariablesTriggersOptionsRetentionHistory

Pipeline  
Build pipeline

Get sources  
Terraform terraform

Agent job 1  
Run on agent

dotnet

Restore  
JNET Core

dotnet

Build  
JNET Core

dotnet

Test  
JNET Core

dotnet

Publish  
NET Core

Copy Terraform files to artifacts

Copy Files

Publish Artifact

Publish Build Artifacts

Copy Files

Version2.\*

Display name \*  
Copy Terraform files to artifacts

Source Folder \*  
Terraform

Contents \*  
\*\*

Target Folder \*  
\$(build.artifactstagingdirectory)/Terraform

Advanced

Control Options

Output Variables

- 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.


#20190816.1 terraform file added



on Terraform-CI


Cancel

Summary

WhiteSource Bolt Build Report

Manually run by  Sriramdas Balaji

 Terraform  terraform b13fbfc

 Today at 10:01 AM


Duration:

Tests:


Changes:

Work items:

Artifacts:

 2m 2s



-

 49 commits

-

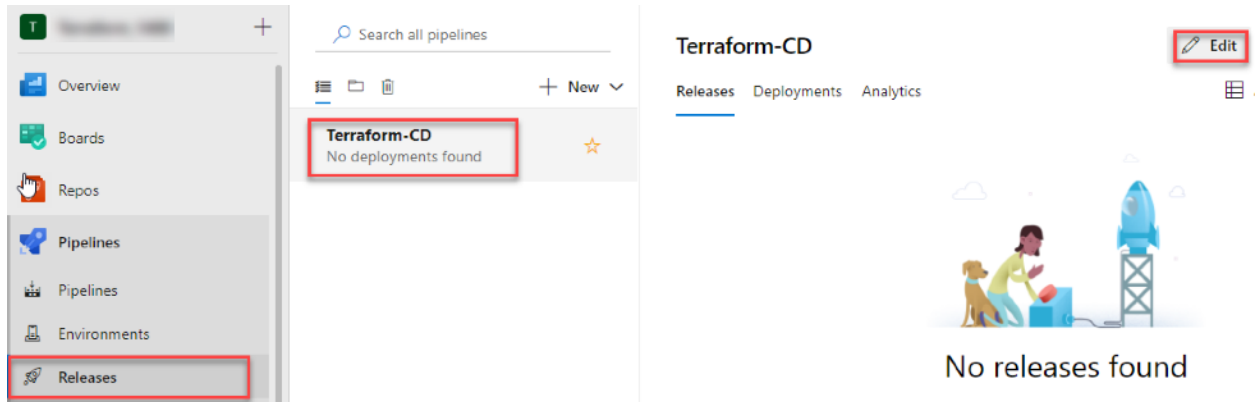
-

Jobs

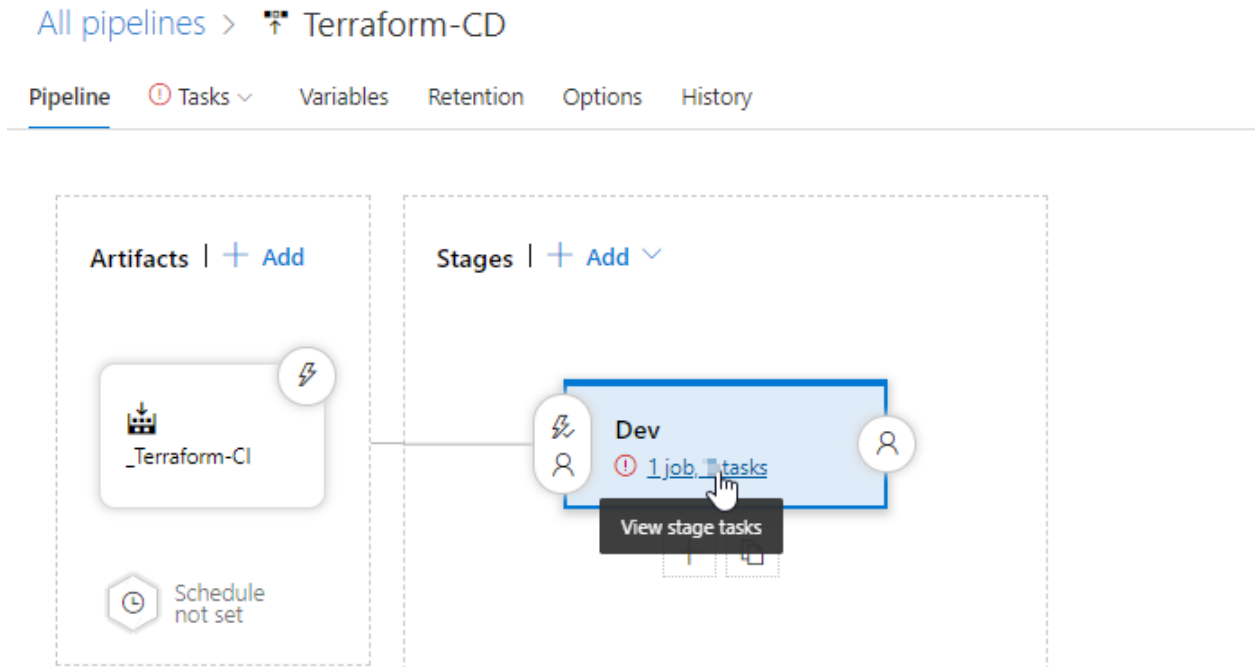
Name	Status	Duration
 Agent job 1	Running	 1m 58s

## 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. You will see the tasks as below.

All pipelines > Terraform-CD

Save

Create release

View releases

...

Pipeline

Tasks

Variables

Retention

Options

History

Deployment process

Agent job

Run on agent

Azure CLI to deploy required Azure resources

Azure PowerShell script to get the storage key

Replace tokens in terraform file

Install Terraform 0.12.3

Terraform : init

Terraform : plan

Terraform : apply -auto-approve

Azure App Service Deploy: \$(appservicename)

Agent job

Remove

Display name \*

Agent job

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.

Dev  
Deployment process

Agent job  
Run on agent

Azure CLI to deploy required Azure resources  
Some settings need attention

Azure PowerShell script to get the storage key  
Some settings need attention

Replace tokens in terraform file  
Replace Tokens

Install Terraform 0.12.3  
Terraform tool installer

Terraform : init  
Some settings need attention

Terraform : plan  
Some settings need attention

Terraform : apply -auto-approve  
Some settings need attention

Azure App Service Deploy: \$(appservicename)  
Some settings need attention

Azure CLI

Task version 1.4

Display name \*  
Azure CLI to deploy required Azure resources

Azure subscription \*  
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 $(terraformstorageg)

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

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

call az storage account keys list -g $(terraformstorageg) -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. On the left, a list of tasks is shown, with 'Azure PowerShell script to get the storage key' highlighted and marked with a red circle '1'. The main configuration area on the right shows the following settings:

- Task version:** 3.0
- Display name:** Azure PowerShell script to get the storage key
- Azure Connection Type:** Azure Resource Manager (marked with a red circle '2')
- Azure Subscription:** Visual Studio Enterprise (marked with a red circle '3')
- 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 $(terraformstorageorg) -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. On the left, the task list shows 'Replace tokens in terraform file' highlighted. The main configuration area 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
appserviceplan	pulterraformweb81e164bf
appserviceplan	PULTerraformplan
storagekey	PipelineWillGetThisValueRuntime
terraformstorageaccount	terraformstorage81e164bf
terraformstoragekey	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

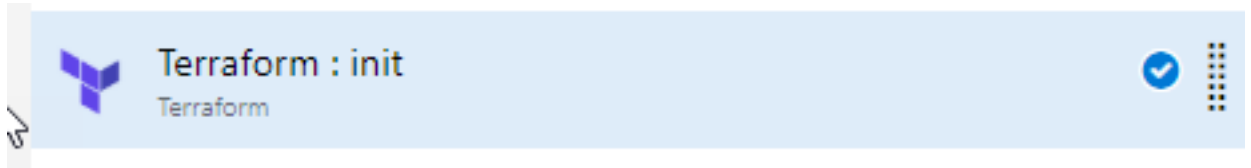
8. When running Terraform in automation, the focus is usually on the core plan/apply cycle. The main Terraform workflow is shown below:





- 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 ▼



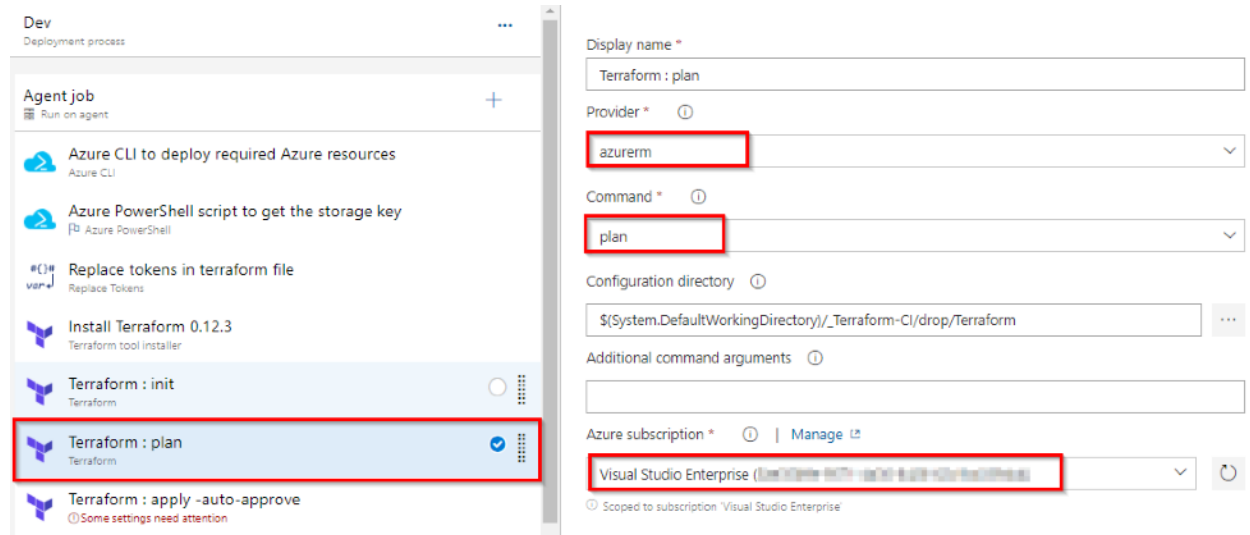
Key \* ⓘ

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. On the left, a list of tasks in an 'Agent job' includes 'Terraform : apply -auto-approve', which is highlighted with a red box. On the right, the configuration details are shown: 'Display name' is 'Terraform : apply -auto-approve', 'Provider' is 'azurerm', 'Command' is 'validate and apply' (highlighted with a red box), 'Configuration directory' is '\$(System.DefaultWorkingDirectory)/\_Terraform-CI/drop/Terraform', 'Additional command arguments' is '-auto-approve' (highlighted with a red box), and 'Azure subscription' is 'Visual Studio Enterprise' (highlighted with a red box).

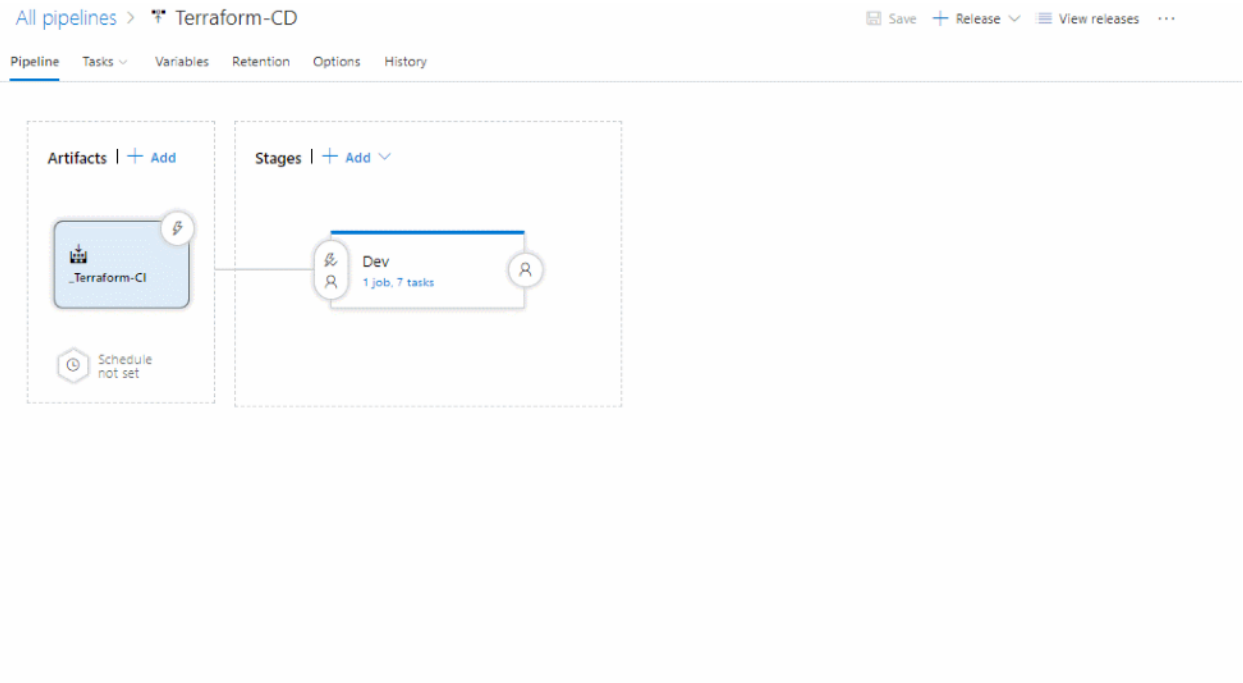
This task will run the **terraform apply** command to deploy the resources. By default, it will also prompt for confirmation that you want to apply those changes. Since we are automating the deployment we are adding **auto-approve** argument to not prompt for confirmation.

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. On the left, the task is highlighted with a red box in the 'Agent job' list. On the right, the configuration details are shown: 'Task version' is '3.\*', 'Display name' is 'Azure App Service Deploy: \$(appservicename)', 'Azure subscription' is 'Visual Studio Enterprise' (highlighted with a red box), 'App type' is 'Web App', and 'App Service name' is '\$(appservicename)' (highlighted with a red box).

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.

