

# Activate Azure with DevOps

---

## Module: Automating infrastructure deployment in the Cloud with Terraform and Azure Pipelines

### Student Lab Manual

#### Conditions and Terms of Use

##### Microsoft Confidential - For Internal Use Only

This training package is proprietary and confidential and is intended only for uses described in the training materials. Content and software is provided to you under a Non-Disclosure Agreement and cannot be distributed. Copying or disclosing all or any portion of the content and/or software included in such packages is strictly prohibited.

The contents of this package are for informational and training purposes only and are provided "as is" without warranty of any kind, whether express or implied, including but not limited to the implied warranties of merchantability, fitness for a particular purpose, and non-infringement.

Training package content, including URLs and other Internet Web site references, is subject to change without notice. Because Microsoft must respond to changing market conditions, the content should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information presented after the date of publication. Unless otherwise noted, the companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place, or event is intended or should be inferred.

#### Copyright and Trademarks

© 2020 Microsoft Corporation. All rights reserved.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

For more information, see **Use of Microsoft Copyrighted Content** at <http://www.microsoft.com/about/legal/permissions/>

Microsoft®, Internet Explorer®, and Windows® are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. Other Microsoft products mentioned herein may be either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries. All other trademarks are property of their respective owners.

# Contents

## Overview

[Exercise 1: Examine the Terraform file in your Source code](#)

[Exercise 2: Build your application using Azure CI Pipeline](#)

[Exercise 3: Deploy resources using Terraform in Azure CD pipeline](#)

## Overview

**Terraform** is a tool for building, changing and versioning infrastructure safely and efficiently. Terraform can manage existing and popular cloud service providers as well as custom in-house solutions.

Configuration files describe to **Terraform** the components needed to run a single application or your entire datacenter. Terraform generates an execution plan describing what it will do to reach the desired state, and then executes it to build the described infrastructure. As the configuration changes, Terraform is able to determine what changed and create incremental execution plans which can be applied.



Want additional learning? Check out the [Provision infrastructure in Azure Pipelines](#) module on Microsoft Learn.

## What's covered in this lab

In this lab, you will see

1. How open source tools, such as Terraform can be leveraged to implement Infrastructure as Code (**IaC**)
2. How to automate your infrastructure deployments in the Cloud with Terraform and Azure Pipelines

## Before you begin

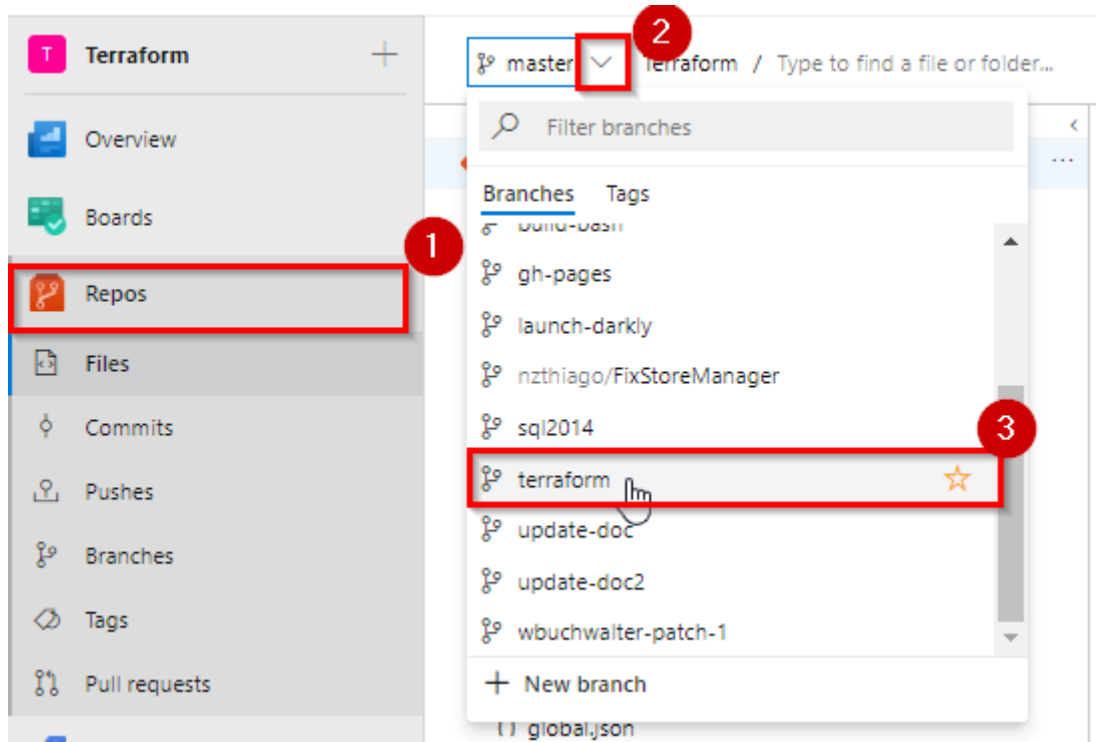
1. Use the [Azure DevOps Demo Generator](#) to provision the project on your Azure DevOps organization. This URL will automatically select **Terraform** template in the demo generator. If you want to try other projects, use this URL instead -[azuredevops generator](#)

Follow the [simple walkthrough](#) to know how to use the Azure DevOps Demo Generator.

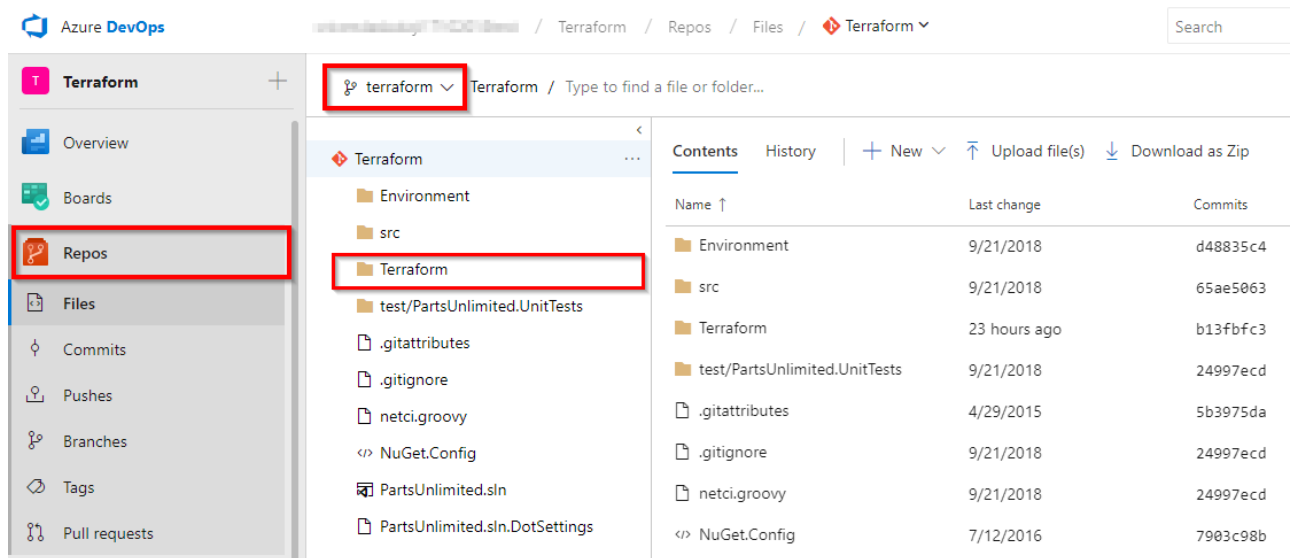
## Exercise 1: Examine the Terraform file in your Source code

In this lab, you will use PartsUnlimited which is an example eCommerce website developed using .Net Core. You will examine the terraform file which helps you to provision the Azure Resources required to deploy PartsUnlimited website.

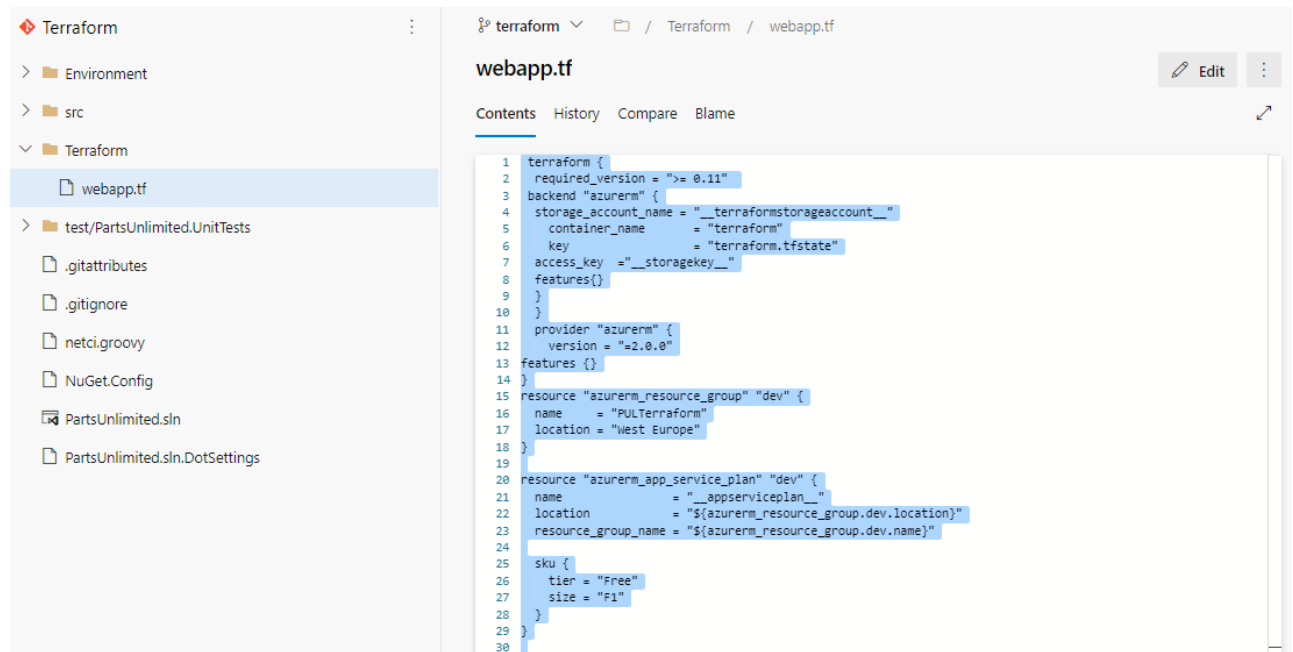
1. Navigate to the project you created above using [Azure DevOps Demo Generator](#)
2. Select **Repos**. Switch to **terraform** branch.



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



3. 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.

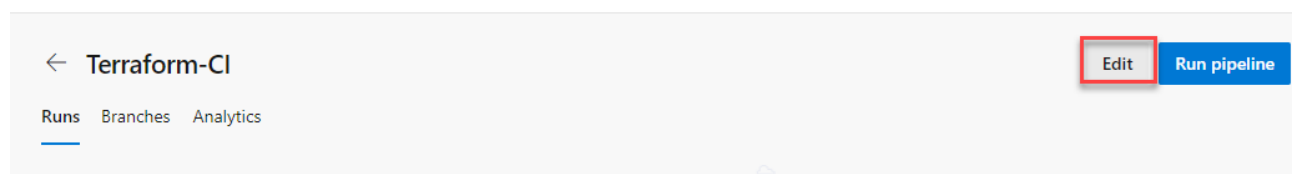
In this example, we want to deploy an Azure Resource group, App service plan and App service required to deploy the website. And we have added Terraform file (Infrastructure as Code) to source control repository in your Azure DevOps project which can deploy the required Azure resources.

Learn more about the Terraform workflow [here](#).

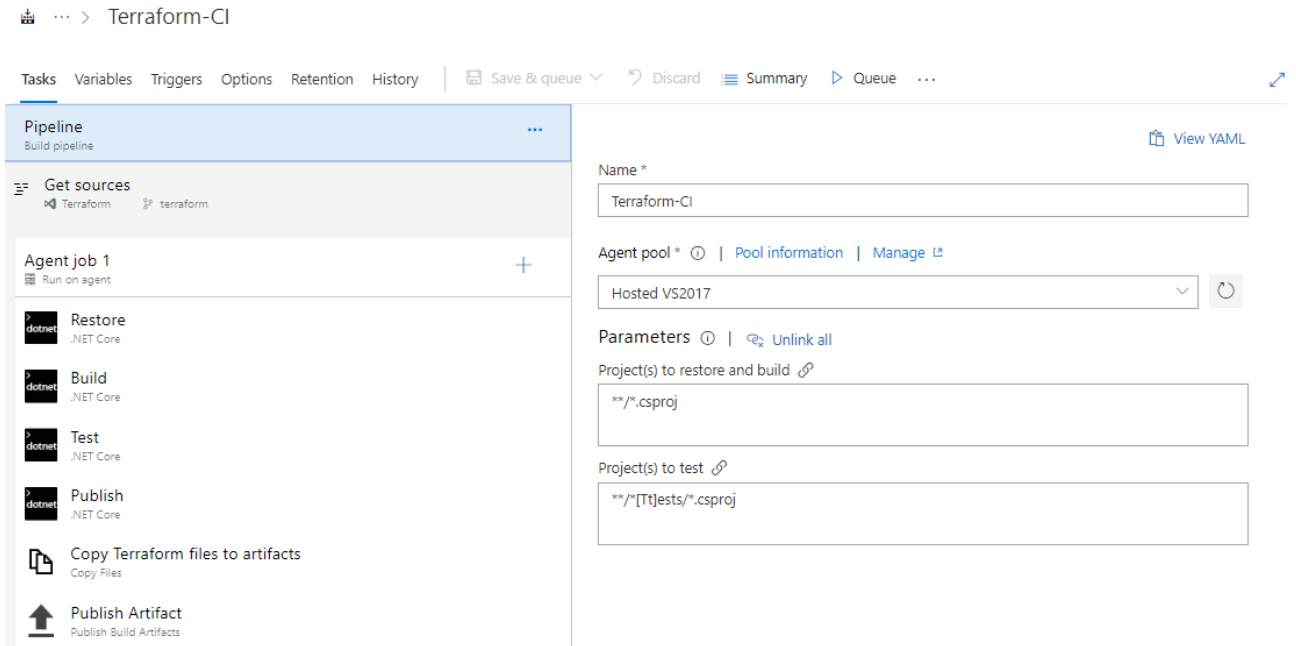
## Exercise 2: Build your application using Azure CI Pipeline

In this exercise, you will build your application and publish the required files to an artifact called drop.

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

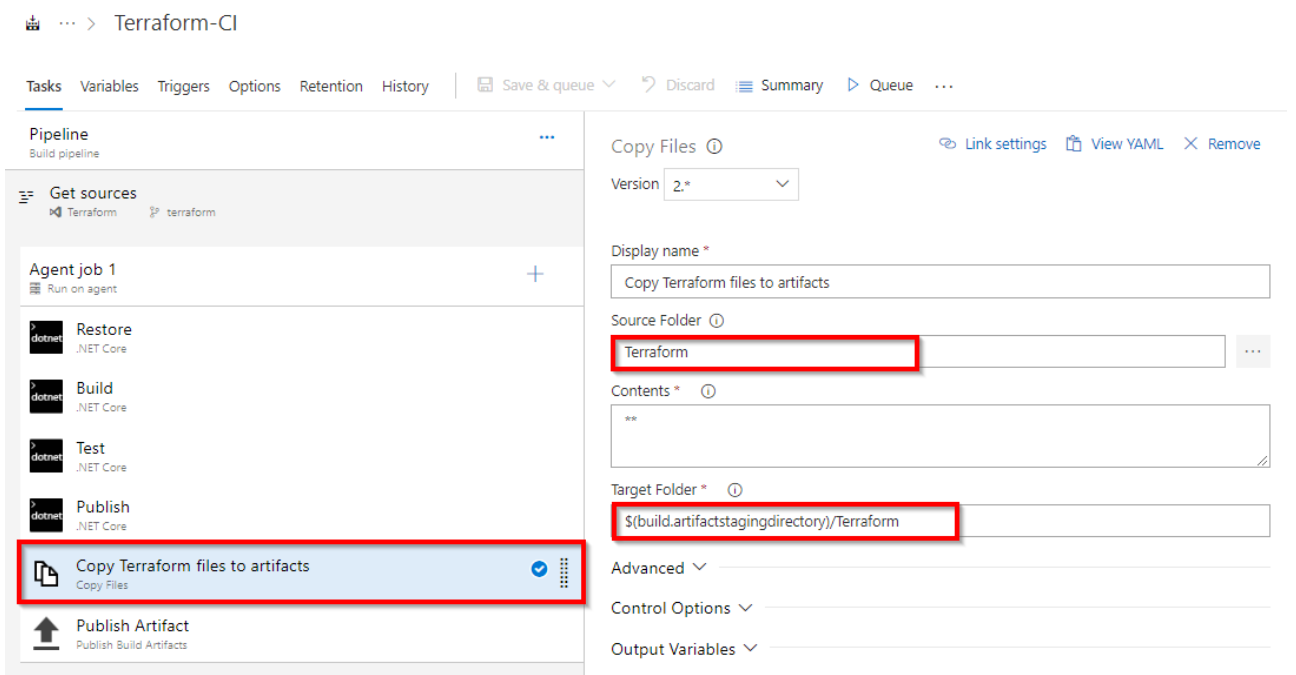


2. Your build pipeline will look like as below. This CI pipeline has tasks to compile .Net Core project. The **dotnet** tasks in the pipeline will restore dependencies, build, test and publish the build output into a zip file (package) which can be deployed to a web application.





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

 **#20190816.1 terraform file added**  
on Terraform-CD

Cancel 

**Summary** WhiteSource Bolt Build Report

**Manually run by**  **Sriramdas Balaji**

 Terraform  terraform b13fbfc

Today at 10:01 AM

Duration: 2m 2s


Tests: -

Changes: 49 commits

Work items: -

Artifacts: -


**Jobs**

Name	Status	Duration
 Agent job 1	Running	1m 58s

## Exercise 3: Deploy resources using Terraform in Azure CD pipeline

In this exercise, you will create azure resources using Terraform as part of your deployment(CD) pipeline and deploy the PartsUnlimited application to the App service provisioned by Terraform.

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


 **Terraform-CD**


Overview Boards Repos Pipelines Pipelines Environments **Releases**

Search all pipelines

**Terraform-CD**  
No deployments found

**Terraform-CD**  
Releases Deployments Analytics

 **Edit**

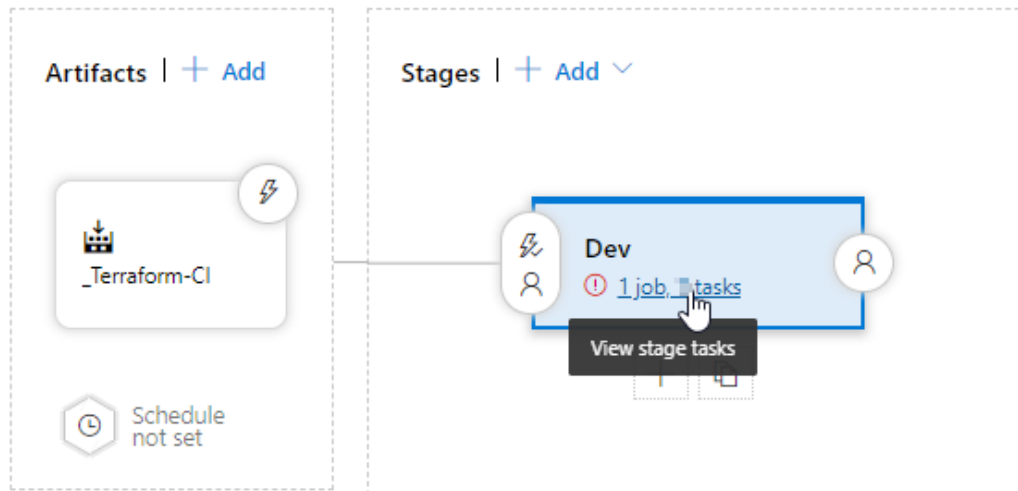


No releases found

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

## All pipelines &gt; Terraform-CD

Pipeline Tasks Variables Retention Options History



3. You will see the tasks as below.

The screenshot shows the 'Tasks' tab for the 'Terraform-CD' pipeline. The 'Agent job' is selected, and the 'Azure CLI to deploy required Azure resources' task is highlighted. The 'Agent selection' dropdown is set to 'Hosted VS2017'. The 'Demands' table shows 'azureps' with condition 'exists'.

Name	Condition	Value
azureps	exists	

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 ①

Task version 1,\*

Display name \*  
Azure CLI to deploy required Azure resources

Azure subscription \* ② | Manage ③  
Visual Studio Enterprise (Contributor Role) [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 \* ①  
Inline script

Inline Script \* ①

```
# 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.

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 : 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 PowerShell ①

Task version 3,\*

Display name \*  
Azure PowerShell script to get the storage key

Azure Connection Type ②  
Azure Resource Manager

Azure Subscription \* ③ | Manage ③  
Visual Studio Enterprise (Contributor Role) [Authorize]

Scoped to subscription 'Visual Studio Enterprise'

Script Type ①  
☐ Script File Path ☒ Inline Script

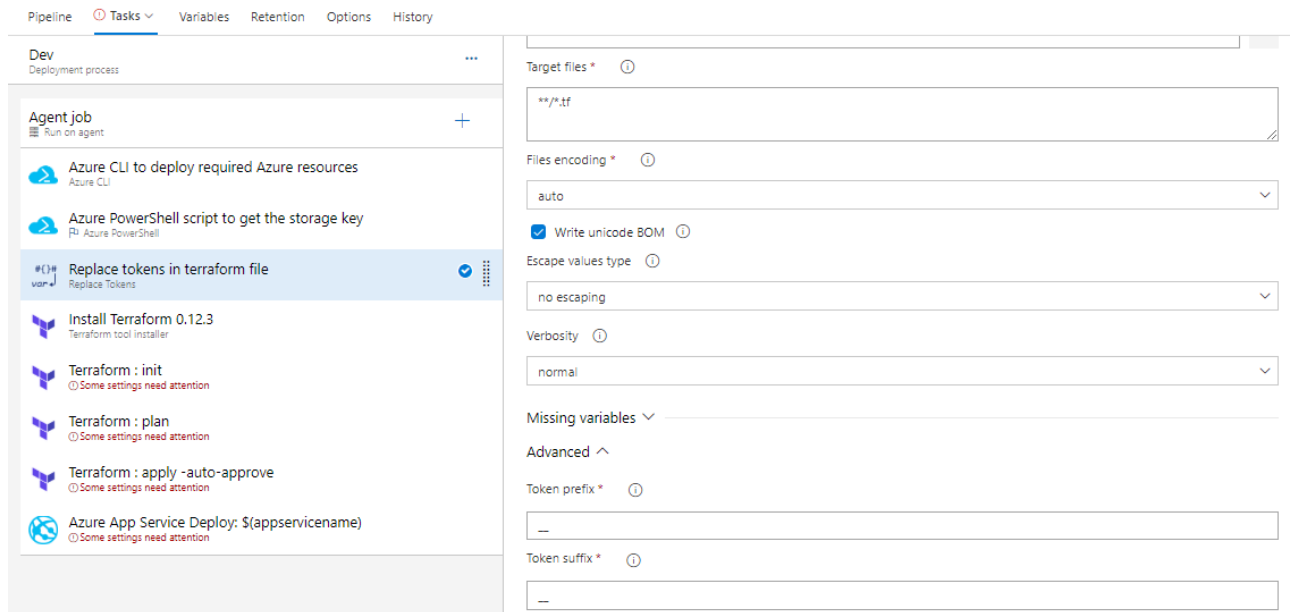
Inline Script ①

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

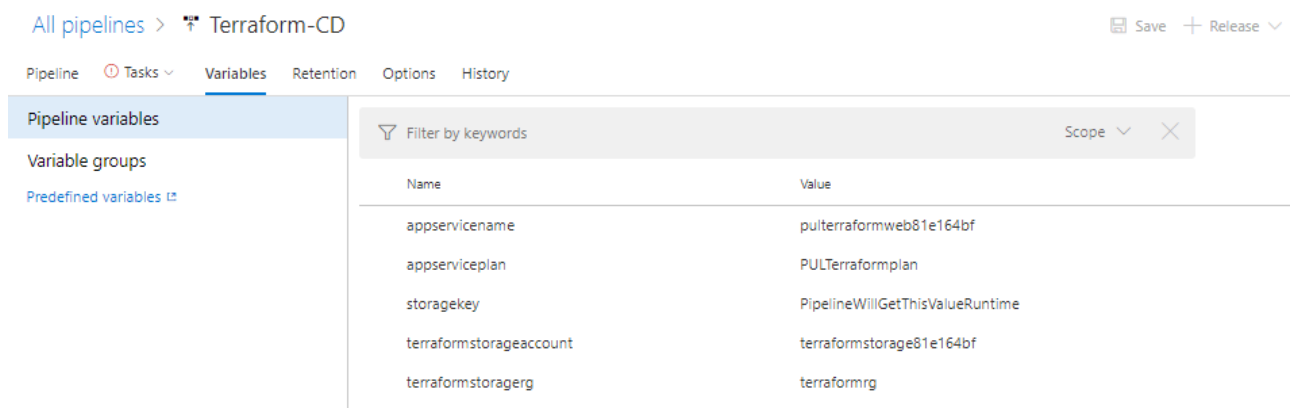
To configure the Terraform **backend** we need Storage account access key. Here we are using Azure PowerShell task to get the Access key of the storage account provisioned in the previous step.

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

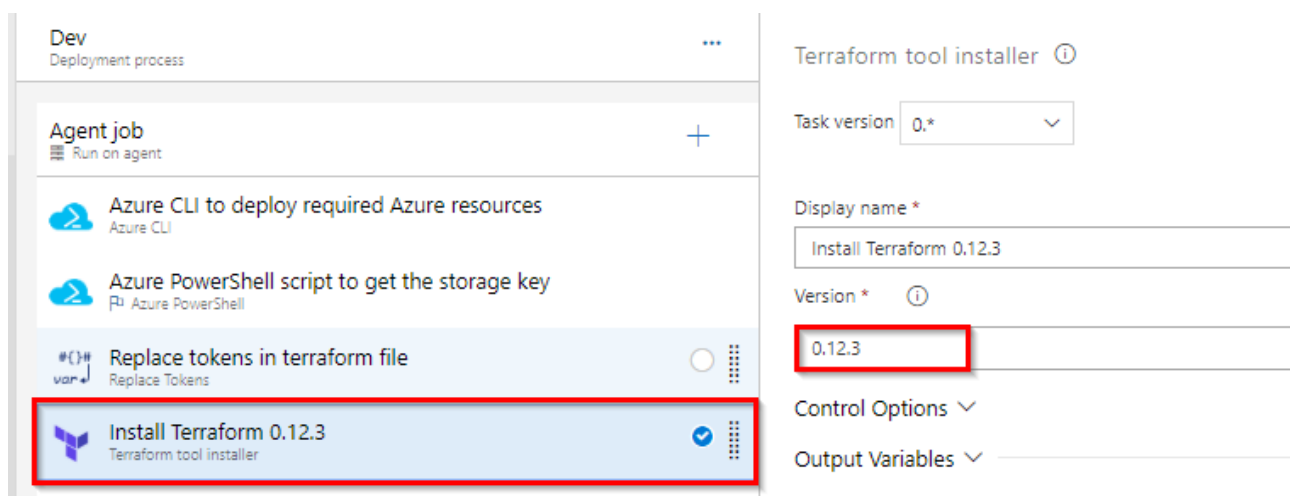




If you observe the **webapp.tf** file in **Exercise 1, Step 3** you will see there are few values are suffixed and prefixed with `_`. For example **terraformstorageaccount**. Using **Replace tokens** task we will replace those values with the variable values defined in the release pipeline.



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.

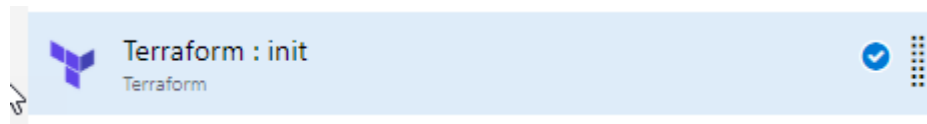
The main Terraform workflow is shown below:



- 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

Display name \*

Provider \* ⓘ

Command \* ⓘ

Configuration directory ⓘ  
 ...

AzureRM backend configuration ^

Azure subscription \* ⓘ | [Manage](#)

ⓘ Scoped to subscription 'Visual Studio Enterprise'

Resource group \* ⓘ

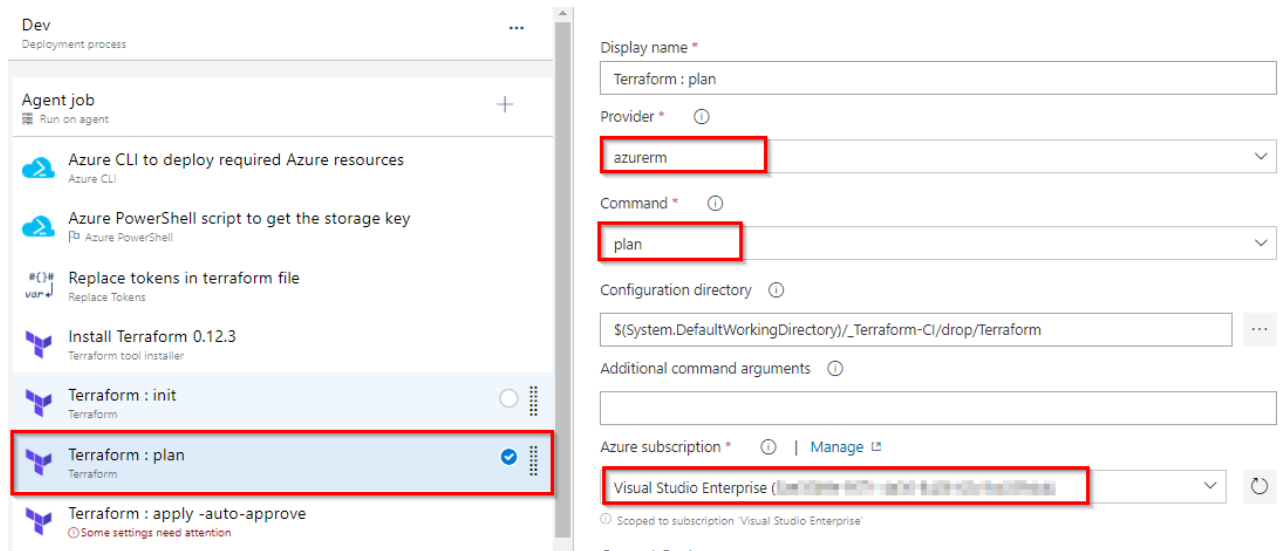
Storage account \* ⓘ

Container \* ⓘ

Key \* ⓘ

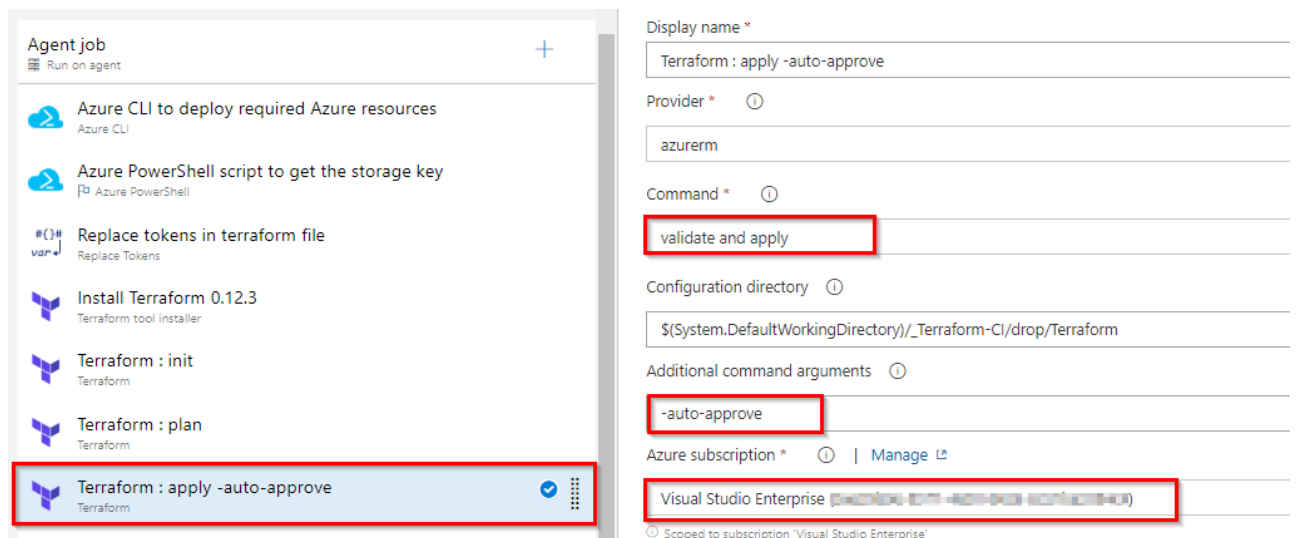
This task runs `terraform init` command. The `terraform init` command looks through all of the \*.tf files in the current working directory and automatically downloads any of the providers required for them. In this example, it will download [Azure provider](#) as we are going to deploy Azure resources. 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.



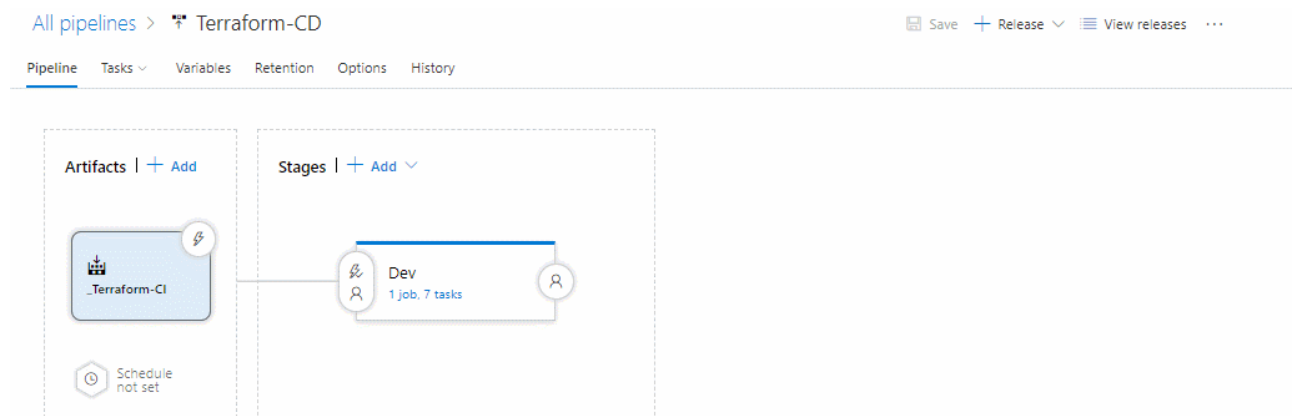
This task will run the `terraform apply` command to deploy the resources. By default, it will also prompt for confirmation that you want to apply those changes. Since we are automating the deployment we are adding `auto-approve` argument to not prompt for confirmation.

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

The screenshot shows the configuration for the 'Azure App Service deploy' task. On the left, a list of tasks in an 'Agent job' is shown, with 'Azure App Service Deploy: \$(appservicename)' highlighted at the bottom. On the right, the configuration for this task is displayed. The 'Task version' is set to '3.\*'. The 'Display name' is 'Azure App Service Deploy: \$(appservicename)'. The 'Azure subscription' is set to 'Visual Studio Enterprise' (highlighted with a red box). The 'App type' is 'Web App'. The 'App Service name' is set to '\$(appservicename)' (highlighted with a red box). The 'Deploy to slot' checkbox is unchecked. The 'Virtual application' field is empty.

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. Search for **pulterraformweb** in App services. Select **pulterraformweb-xxxx** and browse to view the application deployed.

The screenshot shows the Microsoft Azure portal interface. On the left sidebar, 'App Services' is highlighted with a red box and a red circle labeled '1'. The main content area shows the 'App Services' page for the subscription 'Visual Studio Enterprise - Don't...'. A search bar at the top contains 'pulterraformweb' (highlighted with a red box and a red circle labeled '2'). Below the search bar, a table lists 1 item. The table has columns: NAME, STATUS, APP TYPE, and APP SERVICE PLAN. The first item is 'pulterraformweb81e164bf' (highlighted with a red box and a red circle labeled '3'), with a status of 'Running', app type of 'Web app', and app service plan of 'PULTerraformplan'. Below the screenshot, a browser window shows the website 'Parts Unlimited a Fabrikam subsidiary' with a banner for 'ALL OIL AND FILTERS 20% DISCOUNT' and a 'Shop Now' button.

Do you want to learn more about Terraform? If yes click [here](#) for Terraform documentation.

## Summary

In this lab, you have learned how to automate repeatable deployments with Terraform on Azure using Azure Pipelines.