

#How to deploy Terraform in Azure Devops

What does the term Terraform mean in Azure?

Terraform is an open-source tool that lets you manage infrastructure as code (IaC) in Microsoft Azure. Terraform allows you to define your infrastructure in code, making it versionable, repeatable, and auditable.

How to run Terraform in an Azure DevOps pipeline

Follow these steps to run Terraform in an Azure DevOps pipeline:

1. Set up an Azure DevOps project
2. Create Terraform configuration files
3. Define CI/CD steps in YAML
4. Configure the Azure DevOps pipeline
5. Run the pipeline

1. Set up an Azure DevOps project

With your account set up, you next need to sign into [Azure DevOps](#) and create a project where you'll be setting up your CI/CD pipelines.

1. Go to the [Azure DevOps website](#) and sign in with your Microsoft account or organizational account.
2. Navigate to Organization: If you're part of multiple organizations, select the organization where you want to create the project from the top right corner of the page.
3. Click on the "New Project" button.
4. In the "Create a new project" form, fill in the Project Name and visibility (whether the project should be public or private) and optionally, provide a description for your project, then click on the "Create" button.

2. Create Terraform configuration files

Create your Terraform configuration files and store them in a repository. This repo could be in your Azure DevOps project *network.tf*

```
provider "azurerm" {  
  features {}  
}  
resource "azurerm_resource_group" "example" {  
  name     = "example-resources"  
  location = "UK South"  
}  
resource "azurerm_virtual_network" "example" {  
  name                = "example-vnet"  
  resource_group_name = azurerm_resource_group.example.name  
  location             = azurerm_resource_group.example.location
```

```
address_space    = ["172.16.0.0/16"]
}resource "azurerm_subnet" "example" {
  name            = "example-subnet"
  resource_group_name = azurerm_resource_group.example.name
  virtual_network_name = azurerm_virtual_network.example.name
  address_prefixes  = ["172.16.1.0/24"]
}
```


Create new project ×


Project name *

Fabrikam Fibers

Description

Visibility


Public
Anyone on the internet can view the project. Certain features like TFVC are not supported.


Private
Only people you give access to will be able to view this project.

^ Advanced

Version control ?
Git

Work item process ?
Agile

Cancel

Create

```

provider "azurerm" {
  features {}
}

resource "azurerm_resource_group" "example" {
  name     = "example-resources"
  location = "UK South"
}

resource "azurerm_virtual_network" "example" {
  name                = "example-vnet"
  resource_group_name = azurerm_resource_group.example.name
  location            = azurerm_resource_group.example.location
  address_space       = ["172.16.0.0/16"]
}

resource "azurerm_subnet" "example" {
  name                 = "example-subnet"
  resource_group_name = azurerm_resource_group.example.name
  virtual_network_name = azurerm_virtual_network.example.name
  address_prefixes     = ["172.16.1.0/24"]
}

```

3. Define CI/CD steps in YAML

Create the pipeline in code in [YAML format](#) and store it in the same repository:

azure-pipelines.yml

trigger:

- main

pool:

vmImage: 'ubuntu-latest'

stages:

- stage: Plan

```
  displayName: 'Terraform Plan'
  jobs:
    - job: TerraformPlan
      displayName: 'Terraform Plan'
      steps:
        - task: TerraformInstaller@0
          inputs:
            terraformVersion: 'l- script: |
            terraform --version

            terraform init -backend-
config="storage_account_name=$(storageAccountName)" -
backend-config="container_name=$(containerName)" -
backend-config="key=$(key)" -backend-
config="access_key=$(accessKey)"

            terraform plan -out=tfplan

            displayName: 'Terraform Init and Plan'
    - stage: Deploy
      displayName: 'Terraform Apply'
      jobs:
        - job: TerraformApply
          displayName: 'Terraform Apply'
          steps:
            - task: TerraformInstaller@0
```

inputs:

terraformVersion: 'late- script: |

terraform --version

terraform init -backend-

config="storage_account_name=\$(storageAccountName)" -

backend-config="container_name=\$(containerName)" -

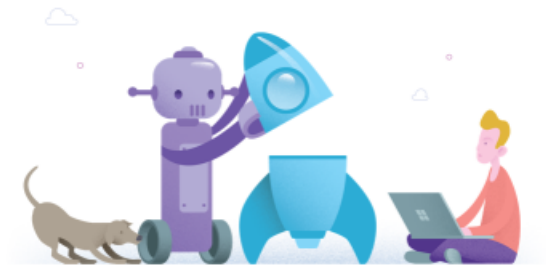
backend-config="key=\$(key)" -backend-

config="access_key=\$(accessKey)"

terraform apply -auto-approve tfplan

displayName: 'Terraform Init and Apply'

- The pipeline is triggered whenever changes are pushed to the main branch.
- The pipeline runs on an Ubuntu agent (ubuntu-latest).
- **TerraformInstaller:** Installs the latest version of Terraform on the build agent.
- **Script:** Executes [Terraform commands](#) to initialize the Terraform working directory, generate a plan, and apply changes to Azure. It initializes Terraform with a backend configuration to store the state in an Azure Storage Account (replace <your-azure-service-connection> with your actual Azure service connection details).



Create your first Pipeline

Automate your build and release processes using our wizard, and go from code to cloud-hosted within minutes.

Create Pipeline

4. Configure the Azure DevOps pipeline

To use the pipeline:

1. In Azure DevOps, go to “Pipelines” > “Pipelines” and click on “New Pipeline”.
2. Point your pipeline to the Git repository containing the azure-pipelines.yml file.
3. Azure DevOps will automatically detect the YAML configuration file. Review and confirm the configuration, then save.

5. Run the pipeline

To run your new pipeline, first, navigate to Pipelines in the left sidebar. Choose the pipeline that you want to run from the list of available pipelines.

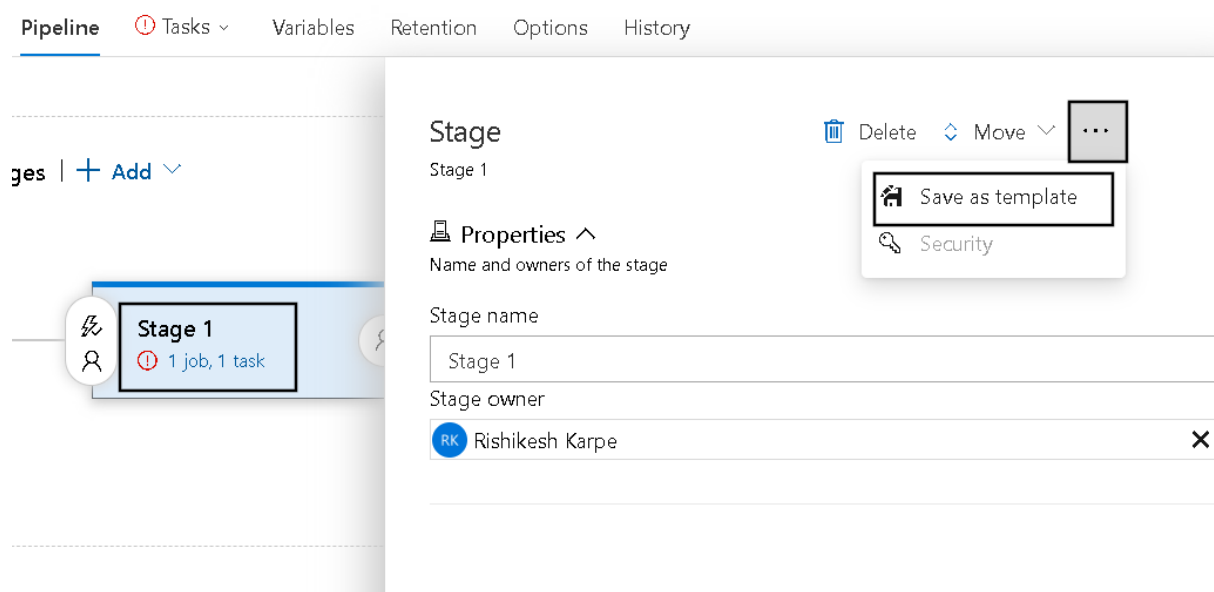
Once you’ve selected the pipeline, you’ll see a “Run pipeline” button at the top right corner. Click on this button to initiate a

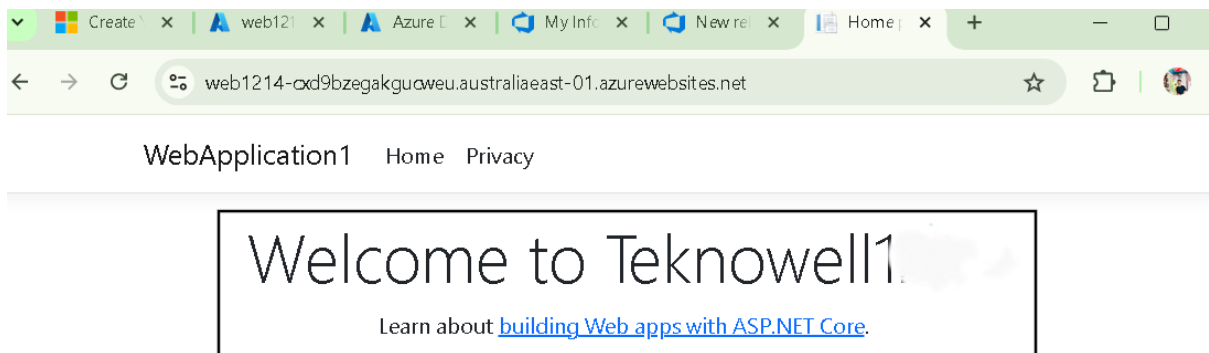
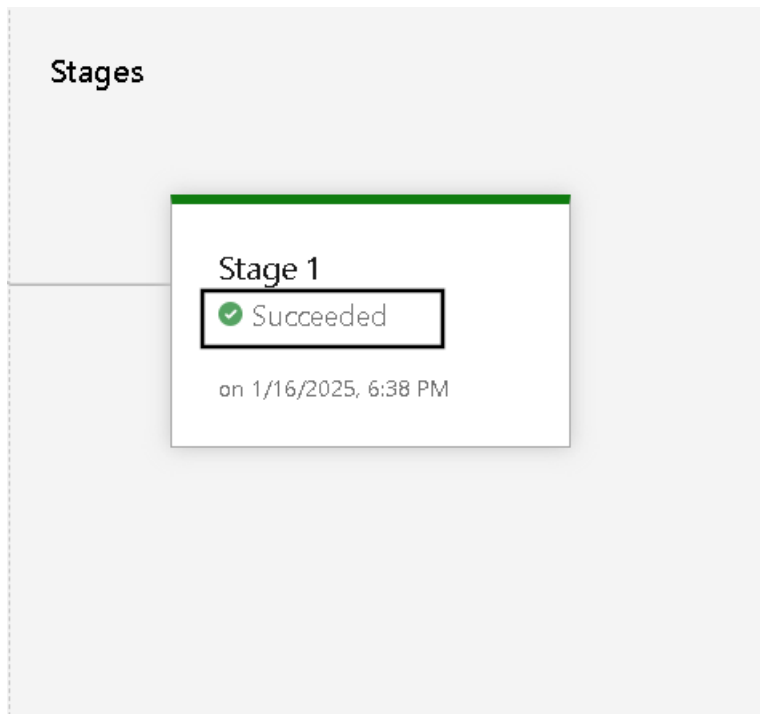
manual run of the pipeline. If your pipeline is set up to trigger on specific branches,

Review the configuration and settings for the pipeline run, and then click on the “Run” button to confirm and start the pipeline execution.

Once the pipeline run is started, you’ll be taken to the pipeline run page, where you can monitor the progress of the pipeline execution. You’ll see the various stages and tasks being executed in real-time. As the pipeline runs, you can view detailed logs for each task to see the output and any error messages. You can also access any artifacts generated by the pipeline, such as build outputs or deployment packages.

After the pipeline run completes, you can review the results to see if the pipeline executed successfully or if there were any failures.





In this way we have deploy terraform successfully.