# Lab 3 – Deploy Terraform using Pipelines

Your proof of concept has shown value to your organization. The CTO has asked you to implement CI/CD, to deploy your Infrastructure. You will be using Azure DevOps Pipelines to deploy your infrastructure as code.

At the end of this lab you:

* Have created a CI pipeline that validates your Terraform IAC
* Have created a CD pipeline that deploys your Terraform changes to a dev, test, acc and production environment

Requirements:

* Your pipelines must be created using .yaml pipelines
* Your pipelines must use multistage pipeline design ([Tutorial: Create a multistage pipeline with Azure DevOps - Azure Pipelines | Microsoft Learn](https://learn.microsoft.com/en-us/azure/devops/pipelines/process/create-multistage-pipeline?view=azure-devops))

**Exercise 0: Create a Hello world pipeline**

Create a simple .yaml pipeline that prints out Hello-World.

**Exercise 1: Create A CI pipeline to validate your infrastructure as code**

To prevent any invalid infrastructure as Code from being merged into the main branch, you are going to create a CI branch that performs the validation of your infrastructure as code created in lab 2.

* Create a new pipeline without a trigger
* Create a CI stage
* Within the stage, create the required to validate your infrastructure as code

For this, it’s good to understand what pipeline variables are available to you. Find them here: <https://learn.microsoft.com/en-us/azure/devops/pipelines/build/variables?view=azure-devops>

Then make sure your new pipeline runs every time a pull request is created as part of a branch policy.

**Exercise 2: Create a CD pipeline to deploy your infrastructure to environments**

Now that you have successfully validated your infrastructure, you will now create a pipeline that deploys infrastructure.

When deploying from Azure DevOps you need to use a storage account for the storage of the state file. Perform the following steps to do so.

Use the following piece of code to init your Terraform project.

- task: AzureCLI@1

  displayName: Terraform credentials

  inputs:

    azureSubscription: <your service connection>

    scriptLocation: inlineScript

    inlineScript: |

      set -eu

      subscriptionId=$(az account show --query id -o tsv)

      echo "##vso[task.setvariable variable=ARM\_CLIENT\_ID]$servicePrincipalId"

      echo "##vso[task.setvariable variable=ARM\_SUBSCRIPTION\_ID]$subscriptionId"

      echo "##vso[task.setvariable variable=ARM\_TENANT\_ID]$tenantId"

    addSpnToEnvironment: true

- task: AzureCLI@1

  displayName: Terraform init

  inputs:

    azureSubscription: <your azureServiceConnection>

    scriptLocation: inlineScript

    inlineScript: |

      set -eux  # fail on error

      subscriptionId=$(az account show --query id -o tsv)

      terraform init \

        -backend-config=storage\_account\_name=<your TerraformBackendStorageAccount> \

        -backend-config=container\_name=<your TerraformBackendStorageContainer \

        -backend-config=key=<your environment&name>.tfstate \

        -backend-config=resource\_group\_name=<your TerraformBackendResourceGroup \

        -backend-config=subscription\_id=$subscriptionId \

        -backend-config=tenant\_id=$tenantId \

        -backend-config=client\_id=$servicePrincipalId \

        -backend-config=client\_secret=<your clientSecret>

    workingDirectory: $(Build.SourcesDirectory)/infra

    addSpnToEnvironment: true

Requirements:

* Use a multistage pipeline that deploys to dev, tst, acc, prd
* Use different tvar files for each environment in Terraform
* Your pipeline must only run when changes are merged to main
* Perform the Terraform steps required when you want to want to apply infra to an environment