Configuring Pipelines as Code with YAML

Lab requirements

* This lab requires **Microsoft Edge** or an [Azure DevOps supported browser.](https://docs.microsoft.com/azure/devops/server/compatibility)
* **Set up an Azure DevOps organization:** If you don’t already have an Azure DevOps organization that you can use for this lab, create one by following the instructions available at [Create an organization or project collection](https://docs.microsoft.com/azure/devops/organizations/accounts/create-organization).
* Identify an existing Azure subscription or create a new one.
* Verify that you have a Microsoft account or a Microsoft Entra account with the Owner role in the Azure subscription and the Global Administrator role in the Microsoft Entra tenant associated with the Azure subscription. For details, refer to [List Azure role assignments using the Azure portal](https://docs.microsoft.com/azure/role-based-access-control/role-assignments-list-portal) and [View and assign administrator roles in Azure Active Directory](https://docs.microsoft.com/azure/active-directory/roles/manage-roles-portal).

Lab overview

Many teams prefer to define their build and release pipelines using YAML. This allows them to access the same pipeline features as those using the visual designer but with a markup file that can be managed like any other source file. YAML build definitions can be added to a project by simply adding the corresponding files to the repository’s root. Azure DevOps also provides default templates for popular project types and a YAML designer to simplify the process of defining build and release tasks.

Objectives

After you complete this lab, you will be able to:

* Configure CI/CD pipelines as code with YAML in Azure DevOps.

Estimated timing: 60 minutes

Instructions

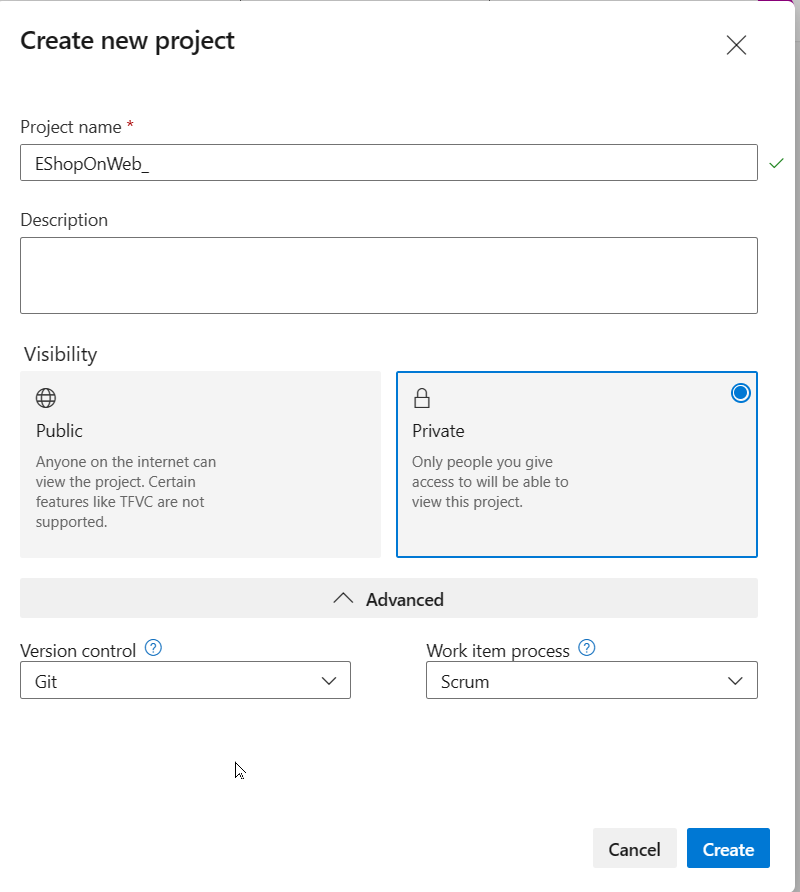
Exercise 0: Configure the lab prerequisites

In this exercise, you will set up the prerequisites for the lab, which consist of a new Azure DevOps project with a repository based on the [eShopOnWeb](https://github.com/MicrosoftLearning/eShopOnWeb).

Task 1: (skip if done) Create and configure the team project

In this task, you will create an **eShopOnWeb\_MultiStageYAML** Azure DevOps project to be used by several labs.

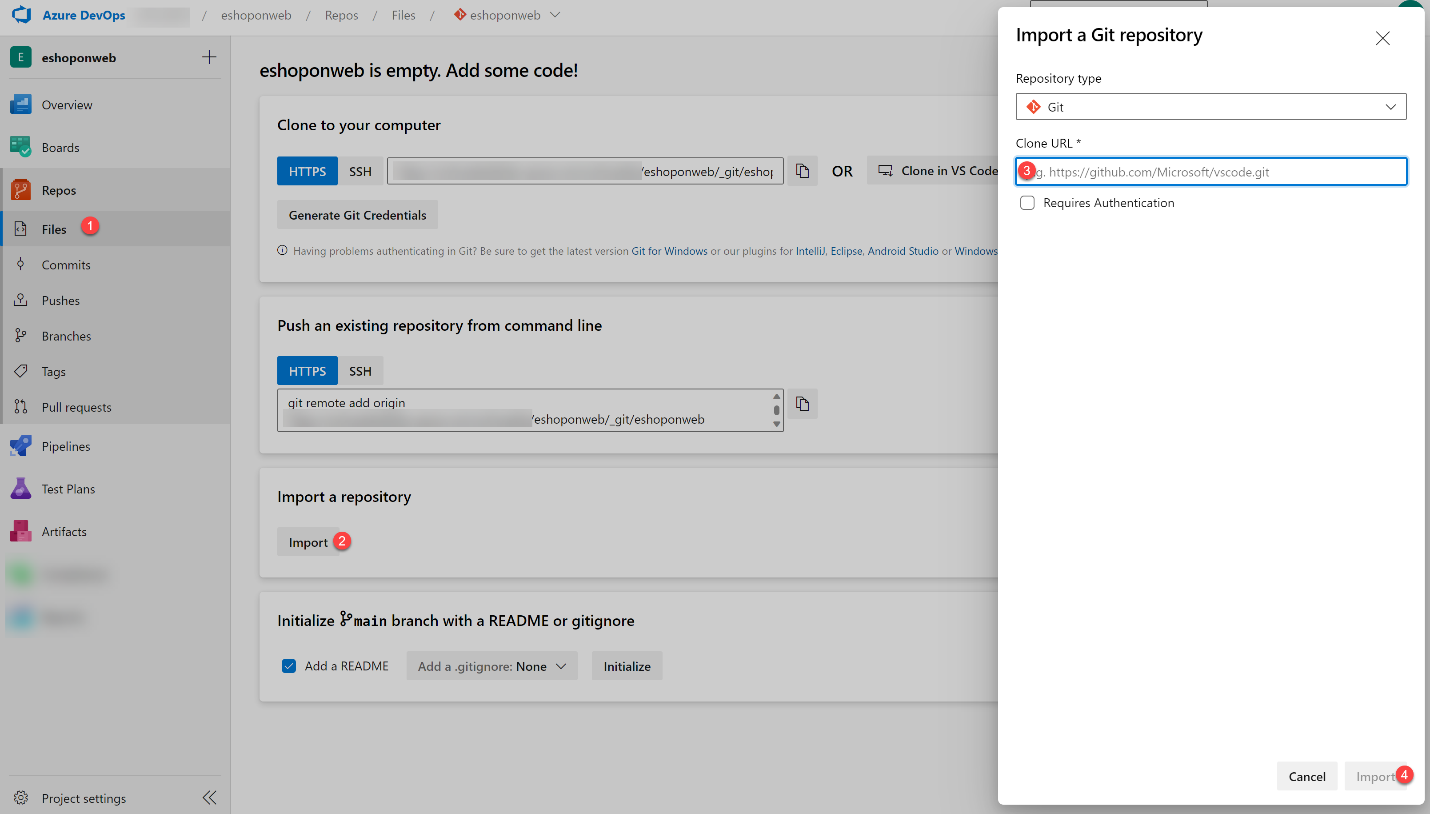
1. On your lab computer, in a browser window open your Azure DevOps organization. Click on **New Project**. Give your project the name **eShopOnWeb\_MultiStageYAML** and leave the other fields with defaults. Click on **Create**.



Task 2: (skip if done) Import eShopOnWeb Git Repository

In this task you will import the eShopOnWeb Git repository that will be used by several labs.

1. On your lab computer, in a browser window open your Azure DevOps organization and the previously created **eShopOnWeb\_MultiStageYAML** project. Click on **Repos>Files** , **Import a Repository**. Select **Import**. On the **Import a Git Repository** window, paste the following URL https://github.com/MicrosoftLearning/eShopOnWeb.git and click **Import**:



1. The repository is organized the following way:
   * **.ado** folder contains Azure DevOps YAML pipelines.
   * **.devcontainer** folder container setup to develop using containers (either locally in VS Code or GitHub Codespaces).
   * **.azure** folder contains Bicep&ARM infrastructure as code templates used in some lab scenarios.
   * **.github** folder container YAML GitHub workflow definitions.
   * **src** folder contains the .NET 7 website used on the lab scenarios.

Task 2: Create Azure resources

In this task, you will create an Azure web app by using the Azure portal.

1. From the lab computer, start a web browser, navigate to the [**Azure Portal**](https://portal.azure.com/), and sign in with the user account that has the Owner role in the Azure subscription you will be using in this lab and has the role of the Global Administrator in the Microsoft Entra tenant associated with this subscription.
2. In the Azure portal, in the toolbar, click the **Cloud Shell** icon located directly to the right of the search text box.
3. If prompted to select either **Bash** or **PowerShell**, select **Bash**.

**Note**: If this is the first time you are starting **Cloud Shell** and you are presented with the **You have no storage mounted** message, select the subscription you are using in this lab, and select **Create storage**.

**Note:** for a list of regions and their alias, run the following command from the Azure Cloud Shell - Bash:

CodeCopy

az account list-locations -o table

1. From the **Bash** prompt, in the **Cloud Shell** pane, run the following command to create a resource group (replace the <region> placeholder with the name of the Azure region closest to you such as ‘centralus’, ‘westeurope’ or other region of choice).

CodeCopy

LOCATION='<region>'

CodeCopy

RESOURCEGROUPNAME='az400m05l11-RG'

az group create --name $RESOURCEGROUPNAME --location $LOCATION

1. To create a Windows App service plan by running the following command:

CodeCopy

SERVICEPLANNAME='az400m05l11-sp1'

az appservice plan create --resource-group $RESOURCEGROUPNAME --name $SERVICEPLANNAME --sku B3

1. Create a web app with a unique name.

CodeCopy

WEBAPPNAME=eshoponWebYAML$RANDOM$RANDOM

az webapp create --resource-group $RESOURCEGROUPNAME --plan $SERVICEPLANNAME --name $WEBAPPNAME

**Note**: Record the name of the web app. You will need it later in this lab.

1. Close the Azure Cloud Shell, but leave the Azure Portal open in the browser.

Exercise 1: Configure CI/CD Pipelines as Code with YAML in Azure DevOps

In this exercise, you will configure CI/CD Pipelines as code with YAML in Azure DevOps.

Task 1: Add a YAML build definition

In this task, you will add a YAML build definition to the existing project.

1. Navigate back to the **Pipelines** pane in of the **Pipelines** hub.
2. In the **Create your first Pipeline** window, click **Create pipeline**.

**Note**: We will use the wizard to create a new YAML Pipeline definition based on our project.

1. On the **Where is your code?** pane, click **Azure Repos Git (YAML)** option.
2. On the **Select a repository** pane, click **eShopOnWeb\_MultiStageYAML**.
3. On the **Configure your pipeline** pane, scroll down and select **Existing Azure Pipelines YAML File**.
4. In the **Selecting an existing YAML File** blade, specify the following parameters:
   * Branch: **main**
   * Path: **.ado/eshoponweb-ci.yml**
5. Click **Continue** to save these settings.
6. From the **Review your Pipeline YAML** screen, click **Run** to start the Build Pipeline process.

The CI definition consists of the following tasks:

* + **DotNet Restore**: With NuGet Package Restore you can install all your project’s dependency without having to store them in source control.
  + **DotNet Build**: Builds a project and all of its dependencies.
  + **DotNet Test**: .Net test driver used to execute unit tests.
  + **DotNet Publish**: Publishes the application and its dependencies to a folder for deployment to a hosting system. In this case, it’s **Build.ArtifactStagingDirectory**.
  + **Publish Artifact - Website**: Publish the app artifact (created in the previous step) and make it available as a pipeline artifact.
  + **Publish Artifact - Bicep**: Publish the infrastructure artifact (Bicep file) and make it available as a pipeline artifact.

1. Wait for the Build Pipeline to complete successfully. Ignore any warnings regarding the source code itself, as they are not relevant for this lab exercise.

**Note**: Each task from the YAML file is available for review, including any warnings and errors.

Task 2: Add continuous delivery to the YAML definition

In this task, you will add continuous delivery to the YAML-based definition of the pipeline you created in the previous task.

**Note**: Now that the build and test processes are successful, we can now add delivery to the YAML definition.

1. On the pipeline run pane, click the ellipsis symbol in the upper right corner and, in the dropdown menu, click **Edit pipeline**.
2. On the pane displaying the content of the **eShopOnWeb\_MultiStageYAML/.ado/eshoponweb-ci.yml** file, navigate to the end of the file (line 56), and hit **Enter/Return** to add a new empty line.
3. Being on line **57**, add the following content to define the **Release** stage in the YAML pipeline.

**Note**: You can define whatever stages you need to better organize and track pipeline progress.

CodeCopy

- stage: Deploy

displayName: Deploy to an Azure Web App

jobs:

- job: Deploy

pool:

vmImage: 'windows-2019'

1. Set the cursor on a new line at the end of the YAML definition.

**Note**: This will be the location where new tasks are added.

1. In the list of tasks on the right side of the code pane, search for and select the **Azure App Service Deploy** task.
2. In the **Azure App Service deploy** pane, specify the following settings and click **Add**:
   * in the **Azure subscription** drop-down list, select the Azure subscription into which you deployed the Azure resources earlier in the lab, click **Authorize**, and, when prompted, authenticate by using the same user account you used during the Azure resource deployment.
   * in the **App Service name** dropdown list, select the name of the web app you deployed earlier in the lab.
   * in the **Package or folder** text box, **update** the Default Value to $(Build.ArtifactStagingDirectory)/\*\*/Web.zip.
3. Confirm the settings from the Assistant pane by clicking the **Add** button.

**Note**: This will automatically add the deployment task to the YAML pipeline definition.

1. The snippet of code added to the editor should look similar to below, reflecting your name for the azureSubscription and WebappName parameters:

CodeCopy

- task: AzureRmWebAppDeployment@4

inputs:

ConnectionType: 'AzureRM'

azureSubscription: 'AZURE SUBSCRIPTION HERE (b999999abc-1234-987a-a1e0-27fb2ea7f9f4)'

appType: 'webApp'

WebAppName: 'eshoponWebYAML369825031'

packageForLinux: '$(Build.ArtifactStagingDirectory)/\*\*/Web.zip'

1. Validate the task is listed as a child of the **steps** task. If not, select all lines from the added task, press the **Tab** key twice to indent it four spaces, so that it listed as a child of the **steps** task.

**Note**: The **packageForLinux** parameter is misleading in the context of this lab, but it is valid for Windows or Linux.

**Note**: By default, these two stages run independently. As a result, the build output from the first stage might not be available to the second stage without additional changes. To implement these changes, we will add a new task to download the deployment artifact in the beginning of the deploy stage.

1. Place the cursor on the first line under the **steps** node of the **deploy** stage, and hit Enter/Return to add a new empty line (Line 64).
2. On the **Tasks** pane, search for and select the **Download build artifacts** task.
3. Specify the following parameters for this task:
   * Download Artifacts produced by: **Current Build**
   * Download Type: **Specific Artifact**
   * Artifact Name: **select “Website” from the list** (or **type “Website”** directly if it doesn’t appear automatically in the list)
   * Destination Directory: **$(Build.ArtifactStagingDirectory)**
4. Click **Add**.
5. The snippet of added code should look similar to below:

CodeCopy

- task: DownloadBuildArtifacts@0

inputs:

buildType: 'current'

downloadType: 'single'

artifactName: 'Website'

downloadPath: '$(Build.ArtifactStagingDirectory)'

1. If the YAML indentation is off, With the added task still selected in the editor, press the **Tab** key twice to indent it four spaces.

**Note**: Here as well you may also want to add an empty line before and after to make it easier to read.

1. Click **Save**, on the **Save** pane, click **Save** again to commit the change directly into the main branch.

**Note**: Since our original CI-YAML was not configured to automatically trigger a new build, we have to initiate this one manually.

1. From the Azure DevOps left menu, navigate to **Pipelines** and select **Pipelines** again.
2. Open the **EShopOnWeb\_MultiStageYAML** Pipeline and click **Run Pipeline**.
3. Confirm the **Run** from the appearing pane.
4. Notice the 2 different Stages, **Build .Net Core Solution** and **Deploy to Azure Web App** appearing.
5. Wait for the pipeline to kick off and wait until it completes the Build Stage successfully.
6. Once the Deploy Stage wants to start, you are prompted with **Permissions Needed**, as well as an orange bar saying:

CodeCopy

This pipeline needs permission to access a resource before this run can continue to Deploy to an Azure Web App

1. Click on **View**
2. From the **Waiting for Review** pane, click **Permit**.
3. Validate the message in the **Permit popup** window, and confirm by clicking **Permit**.
4. This sets off the Deploy Stage. Wait for this to complete successfully.

**Note**: If the deployment should fail, because of an issue with the YAML Pipeline syntax, use this as a reference:

CodeCopy

#NAME THE PIPELINE SAME AS FILE (WITHOUT ".yml")

# trigger:

# - main

resources:

repositories:

- repository: self

trigger: none

stages:

- stage: Build

displayName: Build .Net Core Solution

jobs:

- job: Build

pool:

vmImage: ubuntu-latest

steps:

- task: DotNetCoreCLI@2

displayName: Restore

inputs:

command: 'restore'

projects: '\*\*/\*.sln'

feedsToUse: 'select'

- task: DotNetCoreCLI@2

displayName: Build

inputs:

command: 'build'

projects: '\*\*/\*.sln'

- task: DotNetCoreCLI@2

displayName: Test

inputs:

command: 'test'

projects: 'tests/UnitTests/\*.csproj'

- task: DotNetCoreCLI@2

displayName: Publish

inputs:

command: 'publish'

publishWebProjects: true

arguments: '-o $(Build.ArtifactStagingDirectory)'

- task: PublishBuildArtifacts@1

displayName: Publish Artifacts ADO - Website

inputs:

pathToPublish: '$(Build.ArtifactStagingDirectory)'

artifactName: Website

- task: PublishBuildArtifacts@1

displayName: Publish Artifacts ADO - Bicep

inputs:

PathtoPublish: '$(Build.SourcesDirectory)/.azure/bicep/webapp.bicep'

ArtifactName: 'Bicep'

publishLocation: 'Container'

- stage: Deploy

displayName: Deploy to an Azure Web App

jobs:

- job: Deploy

pool:

vmImage: 'windows-2019'

steps:

- task: DownloadBuildArtifacts@0

inputs:

buildType: 'current'

downloadType: 'single'

artifactName: 'Website'

downloadPath: '$(Build.ArtifactStagingDirectory)'

- task: AzureRmWebAppDeployment@4

inputs:

ConnectionType: 'AzureRM'

azureSubscription: 'AZURE SUBSCRIPTION HERE (b999999abc-1234-987a-a1e0-27fb2ea7f9f4)'

appType: 'webApp'

WebAppName: 'eshoponWebYAML369825031'

packageForLinux: '$(Build.ArtifactStagingDirectory)/\*\*/Web.zip'

Task 4: Review the deployed site

1. Switch back to web browser window displaying the Azure portal and navigate to the blade displaying the properties of the Azure web app.
2. On the Azure web app blade, click **Overview** and, on the overview blade, click **Browse** to open your site in a new web browser tab.
3. Verify that the deployed site loads as expected in the new browser tab, showing the EShopOnWeb E-commerce website.

Exercise 2: Configure Environment settings for CI/CD Pipelines as Code with YAML in Azure DevOps

In this exercise, you will add approvals to a YAML-based Pipeline in Azure DevOps.

Task 1: Set up Pipeline Environments

YAML Pipelines as Code don’t have Release/Quality Gates as we have with Azure DevOps Classic Release Pipelines. However, some similarities can be configured for YAML Pipelines-as-Code using **Environments**. In this task, you will use this mechanism to configure approvals for the Build Stage.

1. From the Azure DevOps Project **EShopOnWeb\_MultiStageYAML**, navigate to **Pipelines**.
2. Under the Pipelines Menu to the left, select **Environments**.
3. Click **Create Environment**.
4. In the **New Environment** pane, add a Name for the Environment, called **approvals**.
5. Under **Resources**, select **None**.
6. Confirm the settings by pressing the **Create** button.
7. Once the environment got created, click on the “ellipsis” (…) next to the button “Add Resource”.
8. Select **Approvals and Checks**.
9. From the **Add your first check**, select **Approvals**.
10. Add your Azure DevOps User Account Name to the **approvers** field.

**Note:** In a real-life scenario, this would reflect the name of your DevOps team working on this project.

1. Confirm the approval settings defined, by pressing the **Create** button.
2. Last, we need to add the necessary “environment: approvals” settings to the YAML pipeline code for the Deploy Stage. To do this, navigate to **Repos**, browse to the **.ado** folder, and select the **eshoponweb-ci.yml** Pipeline-as-Code file.
3. From the Contents view, click the **Edit** button to switch to Editing mode.
4. Navigate to the start of the **Deploy job** (-job: Deploy on Line 60)
5. Add a new empty line right below, and add the following snippet:

CodeCopy

environment: approvals

The resulting snippet of code should look like this:

CodeCopy

jobs:

- job: Deploy

environment: approvals

pool:

vmImage: 'windows-2019'

1. As the environment is a specific setting of a deployment stage, it cannot be used by “jobs”. Therefore, we have to make some additional changes to the current job definition.
2. On Line **60**, rename “- job: Deploy” to **- deployment: Deploy**
3. Next, under Line **63** (vmImage: Windows-2019), add a new empty line.
4. Paste in the following Yaml Snippet:

CodeCopy

strategy:

runOnce:

deploy:

1. Select the remaining snippet (Line **67** all the way to the end), and use the **Tab** key to fix the YAML indentation.

The resulting YAML snippet should look like this now, reflecting the **Deploy Stage**:

CodeCopy

- stage: Deploy

displayName: Deploy to an Azure Web App

jobs:

- deployment: Deploy

environment: approvals

pool:

vmImage: 'windows-2019'

strategy:

runOnce:

deploy:

steps:

- task: DownloadBuildArtifacts@0

inputs:

buildType: 'current'

downloadType: 'single'

artifactName: 'Website'

downloadPath: '$(Build.ArtifactStagingDirectory)'

- task: AzureRmWebAppDeployment@4

inputs:

ConnectionType: 'AzureRM'

azureSubscription: 'AZURE SUBSCRIPTION HERE (b999999abc-1234-987a-a1e0-27fb2ea7f9f4)'

appType: 'webApp'

WebAppName: 'eshoponWebYAML369825031'

packageForLinux: '$(Build.ArtifactStagingDirectory)/\*\*/Web.zip'

1. Confirm the changes to the code YAML file by clicking **Commit** and clicking **Commit** again in the appearing Commit pane.
2. Navigate to the Azure DevOps Project menu to the left, select **Pipelines**, select **Pipelines** and notice the **EshopOnWeb\_MultiStageYAML** Pipeline used earlier.
3. Open the Pipeline.
4. Click **Run Pipeline** to trigger a new Pipeline run; confirm by clicking **Run**.
5. Just like before, the Build Stage kicks off as expected. Wait for it to complete successfully.
6. Next, since we have the *environment:approvals* configured for the Deploy Stage, it will ask for an approval confirmation before it kicks off.
7. This is visible from the Pipeline view, where it says **Waiting (0/1 checks passed)**. A notification message is also displayed saying **approval needs review before this run can continue to Deploy to an Azure Web App**.
8. Click the **View** button next to this message.
9. From the appearing pane **Checks and manual validations for Deploy to Azure Web App**, click the **Approval Waiting** message.
10. Click **Approve**.
11. This allows the Deploy Stage to kick off and successfully deploying the Azure Web App source code.

**Note:** While this example only used the approvals, know the other checks such as Azure Monitor, REST API, etc… can be used in a similar way

Exercise 3: Remove the Azure lab resources

In this exercise, you will remove the Azure resources provisioned in this lab to eliminate unexpected charges.

**Note**: Remember to remove any newly created Azure resources that you no longer use. Removing unused resources ensures you will not see unexpected charges.

Task 1: Remove the Azure lab resources

In this task, you will use Azure Cloud Shell to remove the Azure resources provisioned in this lab to eliminate unnecessary charges.

1. In the Azure portal, open the **Bash** shell session within the **Cloud Shell** pane.
2. List all resource groups created throughout the labs of this module by running the following command:

ShellCopy

az group list --query "[?starts\_with(name,'az400m05l11-RG')].name" --output tsv

1. Delete all resource groups you created throughout the labs of this module by running the following command:

ShellCopy

az group list --query "[?starts\_with(name,'az400m05l11-RG')].[name]" --output tsv | xargs -L1 bash -c 'az group delete --name $0 --no-wait --yes'

**Note**: The command executes asynchronously (as determined by the –nowait parameter), so while you will be able to run another Azure CLI command immediately afterwards within the same Bash session, it will take a few minutes before the resource groups are actually removed.

Review

In this lab, you configured CI/CD pipelines as code with YAML in Azure DevOps.