

# Azure Pipelines

Once you have pushed your code to Azure Repos, you can create a build pipeline and a release pipeline using Azure Pipelines. This is also known as Continuous Integration (CI) and Continuous Delivery (CD). Build

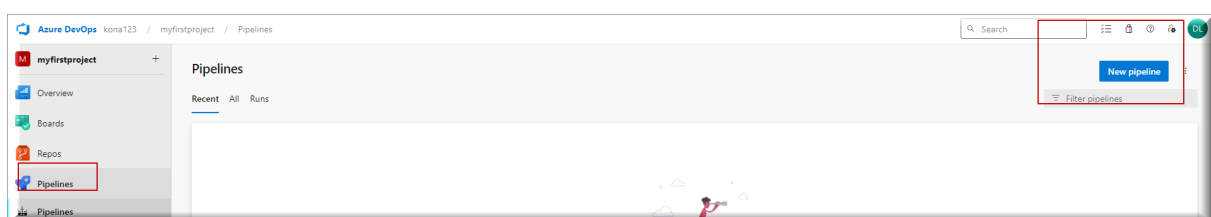
pipeline (CI pipeline) allows you to automate the build and test process of your application. You can setup a build pipeline so that it builds and tests the application code each time a developer commits a change to the source code. The release pipeline (CD pipeline), with the help of the output of the build pipeline, allows you to automate the release process and continuously deliver a high-quality product to your customers.

## Continuous Integration (CI)

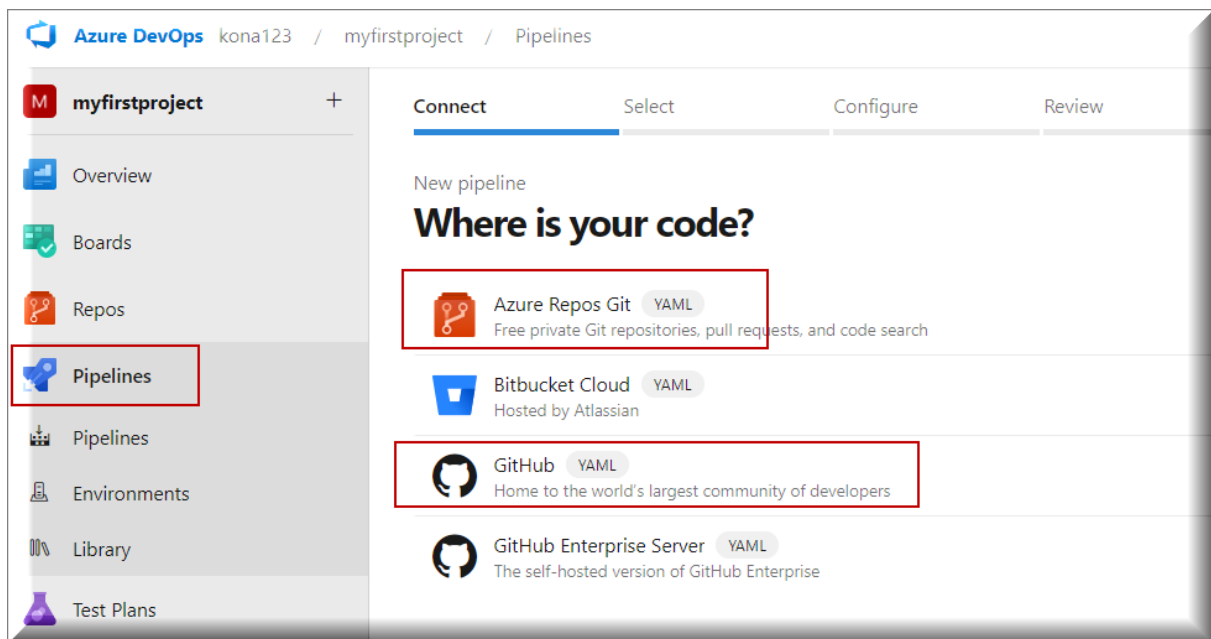
Creating a build pipeline is the first step of Azure Pipelines. There are basically two ways you can start creating a build pipeline.

**Method 1:** Navigate to your repository by clicking on **Repos** and click on the **Set up build** button. In this option, you can skip the selection of the source code location as you are already inside the myquiz repository.

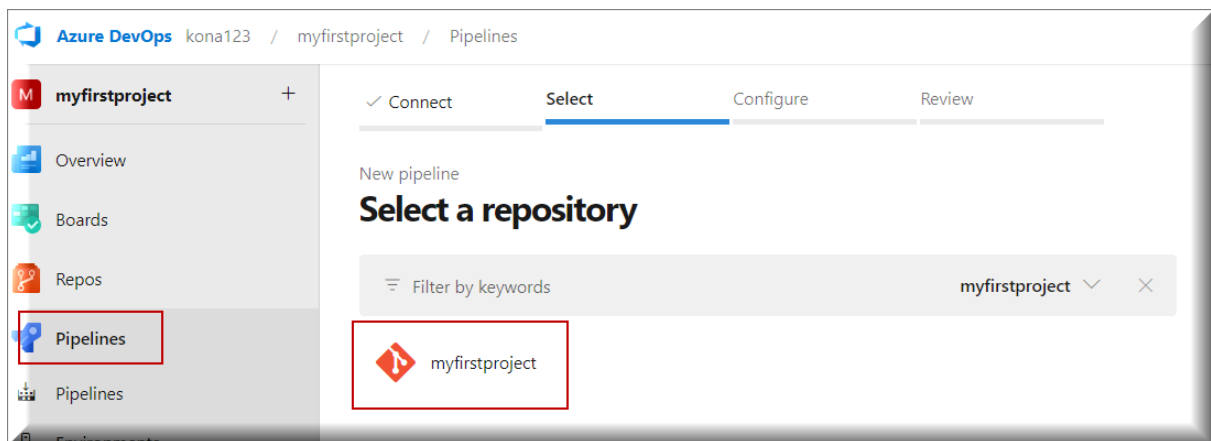
**Method 2:** Click on the menu item **Pipelines** on the left-hand side and then click on the button **Create Pipeline**.



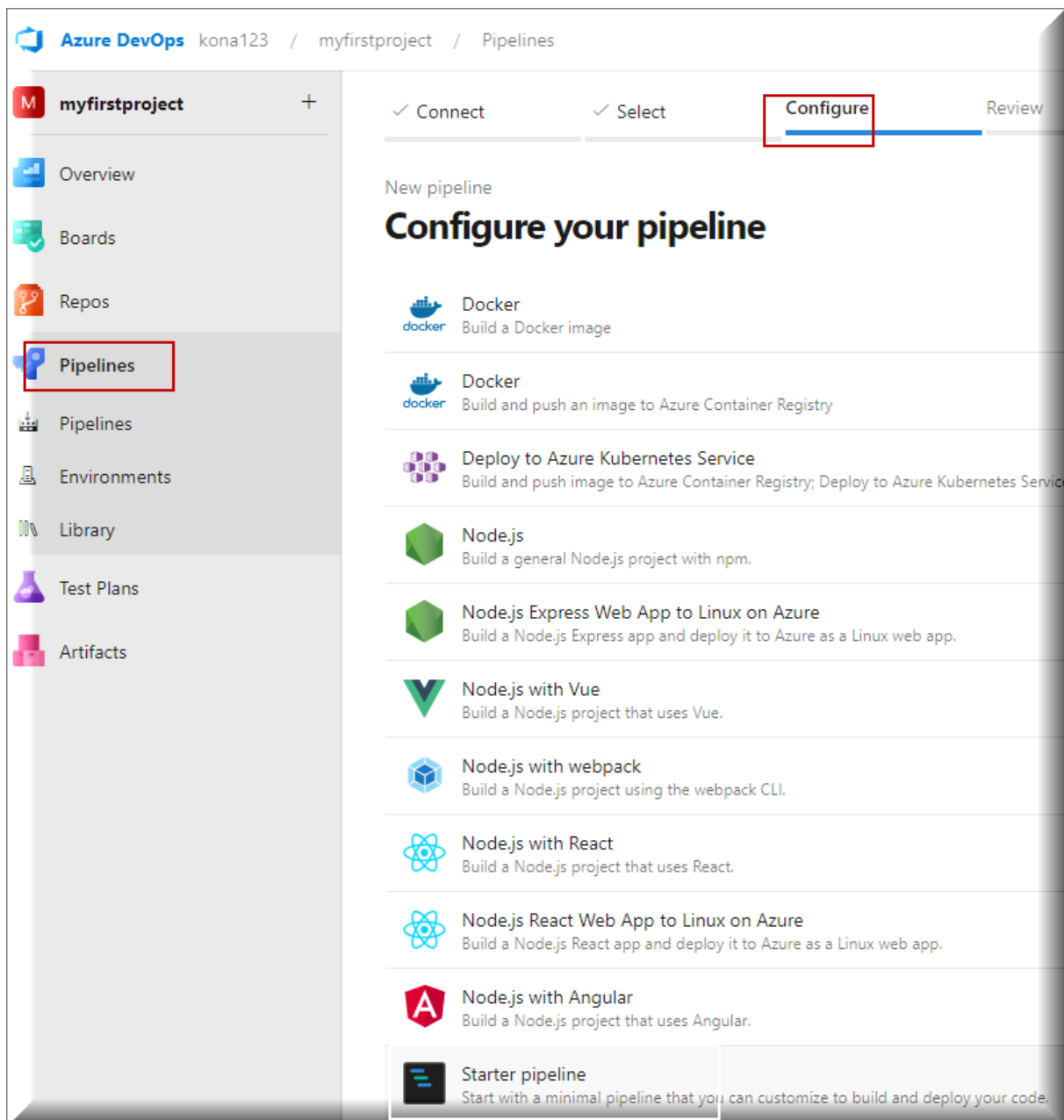
If you choose **Method 2**, you have to specify where your source code resides. In this example, our source code resides in **Azure Repos Git**. Therefore, select **Azure Repos Git (YAML)** option as shown below



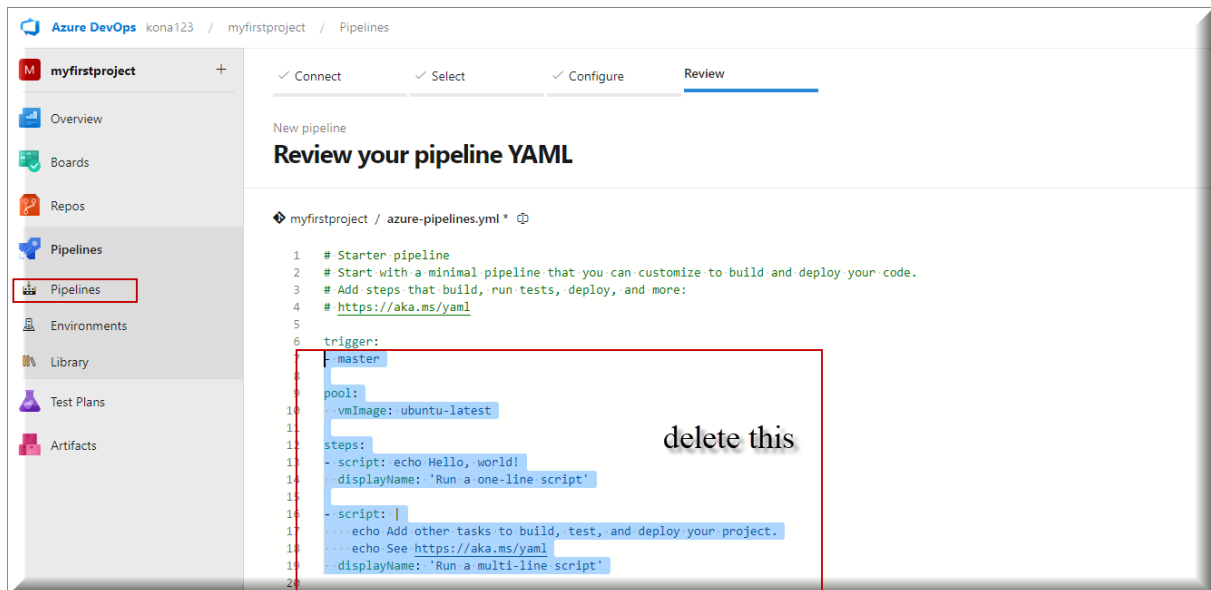
Next, you select your code repository. Select repo that we created in the previous chapter.



Then you can configure your pipeline to match the technology you have selected to build your application. As we have built our application in Vue and Node.js, we select **Node.js with Vue** option.



Based on this selection, Azure pipelines creates a basic starting pipeline definition in YAML that matches Vue.js and Node. You can see the created **azure-pipelines.yml** file in Figure 114.



In order to understand this file, we need some knowledge about YAML data serialization language. Let us try to understand the YAML syntax.

## Introduction to YAML,

**YAML** (YAML Ain't Markup Language) is a data serialization language that is used by Azure pipelines to describe different commands in the pipeline. In other words, you define your build pipeline in code. The language is quite similar to JSON (JavaScript Object Notation) and represented in key value pairs. However, you need to pay attention to the correct indentation when writing YAML and use spaces for indentation. Two space indentation is recommended [4]. YAML files have the extension “.yaml” or “.yml”. Let us look into a simple example to understand the YAML syntax.

Suppose we want to represent a person's data in YAML, and the person object has following attributes.

- *name* (string),
- *age* (integer),
- *marital status* (Boolean),
- *favourite sports* (array) and
- *contact details* (a structure that has a certain format).

We could write this information in YAML as in the following.

## Structure of the basic build definition

Using this syntax, let us try to understand the .yaml build pipeline. Consider the first key-value pair.

trigger:

- master

According to what we have learned; this represents an **array**. In other words, an array of triggers. Basically, we want to trigger our build pipeline each time a developer checks in new code changes to the **master** branch. Assume, you want to run the build pipeline on all the release branches located under the **releases** folder, then you can modify this as follows.

trigger:

- master
- releases/\*

If you want the build pipeline to kick off on every commit in every branch, then you can set it as follows.

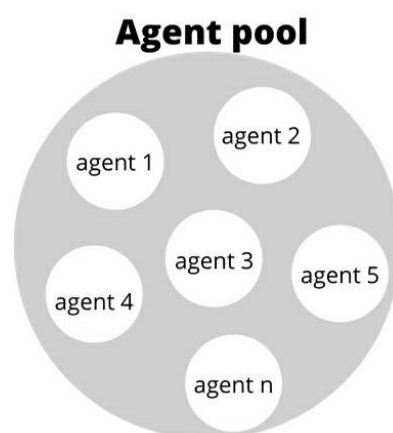
trigger:

- '\*'

Let's move on to the next command which defines the build agent pool. pool:

vmImage: 'ubuntu-latest'

In this command, it specifies a **Microsoft-hosted agent pool**. In Azure Pipelines, the name of this pool is *Azure Pipelines*. An agent pool is used to organize build agents.



A **build agent** can be considered as the heart of the build pipeline, which performs all the jobs defined in the build pipeline. In an Azure DevOps services context, it is an installable software, which is hosted in a virtual machine.

As you can see, the **pool** object contains the **vmImage** property which contains the value '**ubuntu-latest**'. This means that we want to run our build pipeline in a build agent hosted in an Ubuntu virtual machine. **Azure pipelines hosted pool** gives you the option to select from several virtual machine images.

*Table 1: Azure Pipelines hosted VM images*

Virtual machine image	YAML label
Windows Server 2019 with Visual Studio 2019	<b>windows-latest</b> OR <b>windows-2019</b>
Windows Server 2016 with Visual Studio 2017	<b>vs2017-win2016</b>
Ubuntu 18.04	<b>ubuntu-latest</b> OR <b>ubuntu-18.04</b>
Ubuntu 16.04	<b>ubuntu-16.04</b>
macOS X Mojave 10.14	<b>macOS-latest</b> OR <b>macOS-10.14</b>



You can also create a self-hosted agent if you prefer or if you are using on-premise builds using Azure DevOps server.

For example, if you want to run your build agent on a Windows Server 2019 machine having Visual Studio 2019, then you can change the yaml file as following.

pool:

vmImage: 'windows-latest'



A fresh virtual machine instance will be created in the Azure cloud each time you run your build pipeline and it will be discarded after the build process is completed.

The next set of commands define a **job** containing a series of steps performed by the agent. These steps are all about building the application.

steps:

- task: NodeTool@0 inputs:

versionSpec: '10.x' displayName: 'Install Node.js'

- script: | npm install

npm run build

displayName: 'npm install and build'

There are two steps defined here. The first one is a task to install node.js 10.x on the VM image. If you click on the **Settings** link on top of steps, you can see a graphical view of the task which gives you the possibility to add options

to various inputs.

### ← Node.js tool installer

Version Spec \* ⓘ

10.x

☐ Check for Latest Version ⓘ

Another way to achieve the same task is by adding a **demands** attribute to the agent pool. Here, you say that you want node package manager, installed on the agent machine.

pool:

vmImage: 'ubuntu-latest' demands:

- npm

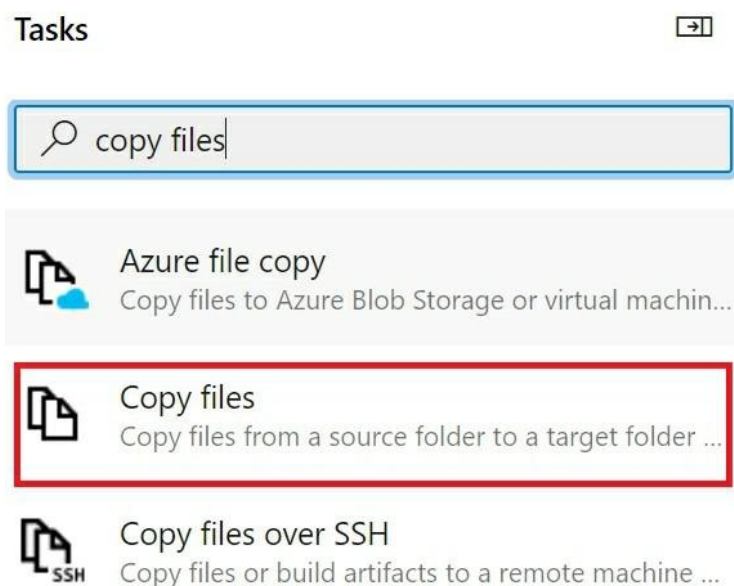
The second task use ***npm install*** command to install the node packages and create a production build of our application using ***npm run build*** command. The pipe (|) symbol is used to preserve the formatting of two commands.



***npm run build*** command will build our application and creates a folder called ***dist*** under the root folder. Contents of the ***dist*** folder will be used when deploy our application to the production.

## Copy files

There are basically three tasks associated for this step. First, you have to copy your **dist** folder to the **artifact staging directory** (represented by the Azure variable `$(Build.ArtifactStagingDirectory)`) in Azure pipelines. Search for the **Copy files task** in the assistant section as shown in Figure below.



The source folder is optional. By default, it will use the root folder of your code repository. This can be accessed by the variable

**`$(Build.SourcesDirectory)`**. Under Contents, specify the location to our dist folder, relative to the Source Folder. Type in **`dist/**`** to select all the content under the dist folder. Finally, specify the target folder by adding the variable

**`$(Build.ArtifactStagingDirectory)`**

to specify the artifact staging directory in Azure pipelines.





Be aware that for Linux build agents you have to use “/” path separator when specifying paths.

## ← Copy files

Source Folder ⓘ

\$(Build.SourcesDirectory)

Contents \* ⓘ

dist/\*\*

Target Folder \* ⓘ

\$(Build.ArtifactStagingDirectory)

Advanced



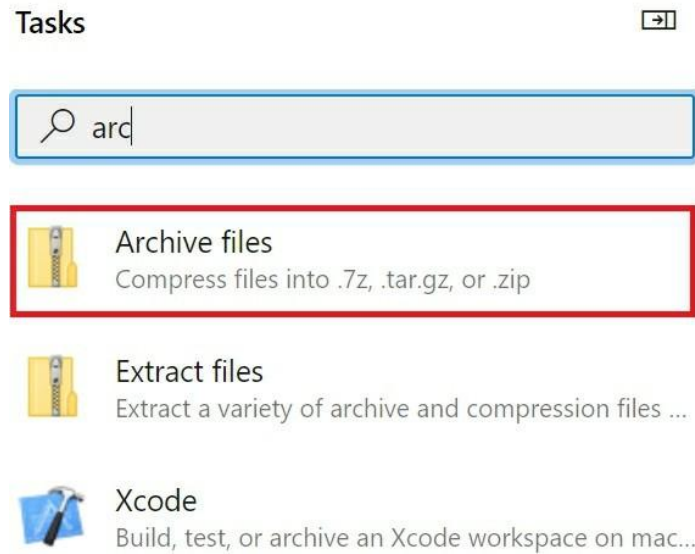
☒ Clean Target Folder ⓘ

[About this task](#)

Add

## Archive the copied files

Then archive the copied **dist** folder using a preferred compression format. So, click on the show assistant button on the right and search for **archive files** task and select it.



The options for this task are shown in Figure 119. The root folder is the dist folder copied to the artifact staging directory, and the drop.zip file will be created on the same artifact staging directory.

←

Archive files

Root folder or file to archive \* ⓘ

\$(Build.ArtifactStagingDirectory)/dist

☐ Prepend root folder name to archive paths \* ⓘ

Archive ^

Archive type \* ⓘ

zip ▾

Archive file to create \* ⓘ

\$(Build.ArtifactStagingDirectory)/drop.zip

☒ Replace existing archive \* ⓘ

☐ Force verbose output ⓘ

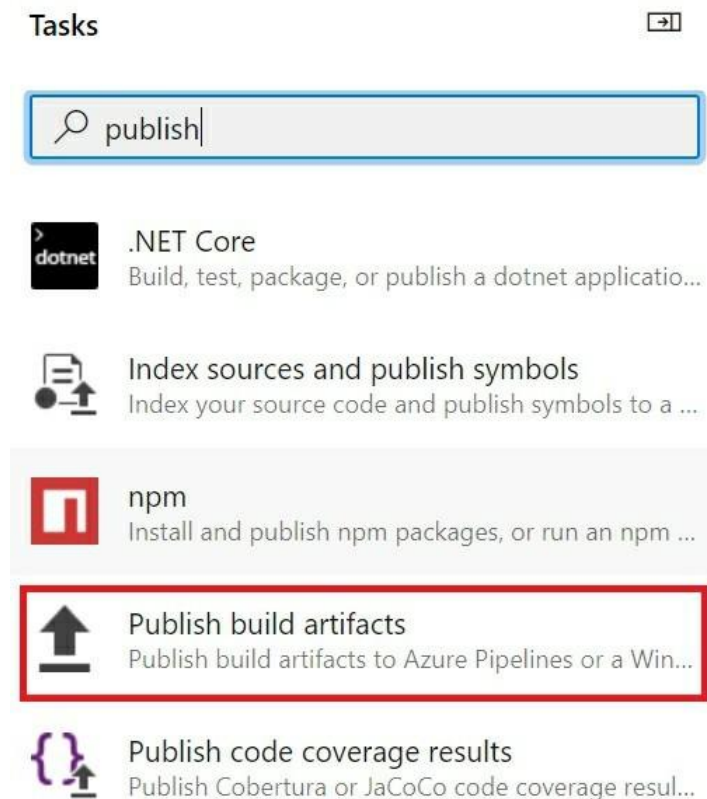
About this task

Add

Figure 119: Options for Archive files task

## Publish artifact to Azure pipelines

In the third step, we need a **publish build artifacts** task. In the *Tasks window*, type “publish” to find the task **Publish build artifacts** and click to select it.



*Figure 120: Publish build artifacts task*

Now, we have 3 options to consider here. First, specify where your build output resides at the moment. As a result of the archive files task, our deployment ready files are located in

**`$(Build.ArtifactStagingDirectory)/drop.zip`** folder. Next, you can provide a name to your artifact created in the first step. Finally, you specify where your artifact is going to be placed. This can be under your build agent - **Azure Pipelines** or in a file share which build agent can find. Here, we select the default Azure Pipelines and click **Add**.

## ← Publish build artifacts

Path to publish \* ⓘ

`$(Build.ArtifactStagingDirectory)/drop.zip`

Artifact name \* ⓘ

drop

Artifact publish location \* ⓘ

Azure Pipelines



About this task

Add

All the steps in the yaml file are listed as follow.

new pipeline

## Review your pipeline YAML

myfirstproject / azure-pipelines.yml \*

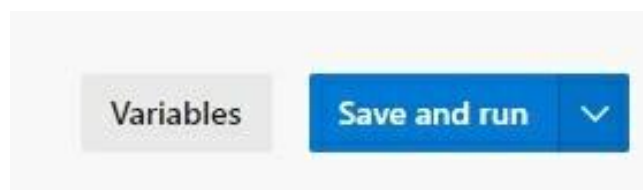
```
1  # Starter pipeline
2  # Start with a minimal pipeline that you can customize to build and deploy your code.
3  # Add steps that build, run tests, deploy, and more:
4  # https://aka.ms/yaml
5
6  trigger:
7  - master
8
9  pool:
10 | - vmImage: ubuntu-latest
11
12  steps:
13  - script: echo Hello, world!
14    displayName: 'Run a one-line script'
15
16  - script: |
17    | echo Add other tasks to build, test, and deploy your project.
18    | echo See https://aka.ms/yaml
19    displayName: 'Run a multi-line script'
20
```



This is not the only way to publish your build artifacts to the staging area. So feel free to play around with different tasks to find out different ways to publish to the staging folder.

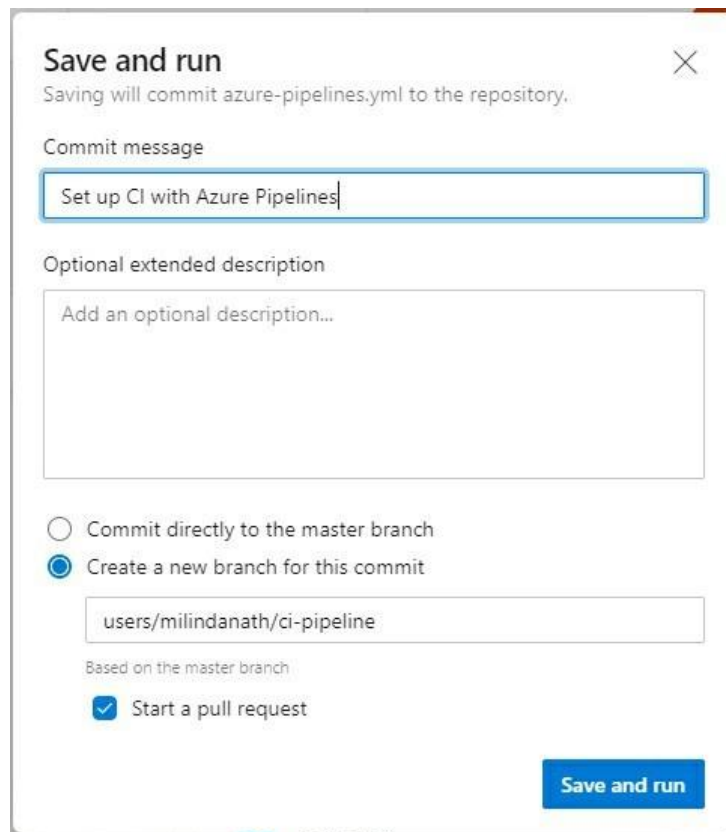
## Save and run the build pipeline

Once you have finalized your build pipeline, it is now time to save all the changes. Click on the button **Save and run** to save the pipeline in your source code and run it immediately.



Now, you will be asked to save the azure-pipelines.yml file to your repository. Here, you can provide a message and commit either to the master branch or to a new branch. As we have setup a policy against committing directly to our master branch,

we have to create a new branch for this and merge the changes to the master branch through a pull request.



**Save and run** ✕

Saving will commit azure-pipelines.yml to the repository.

Commit message

Set up CI with Azure Pipelines

Optional extended description

Add an optional description...

☐ Commit directly to the master branch

☒ Create a new branch for this commit

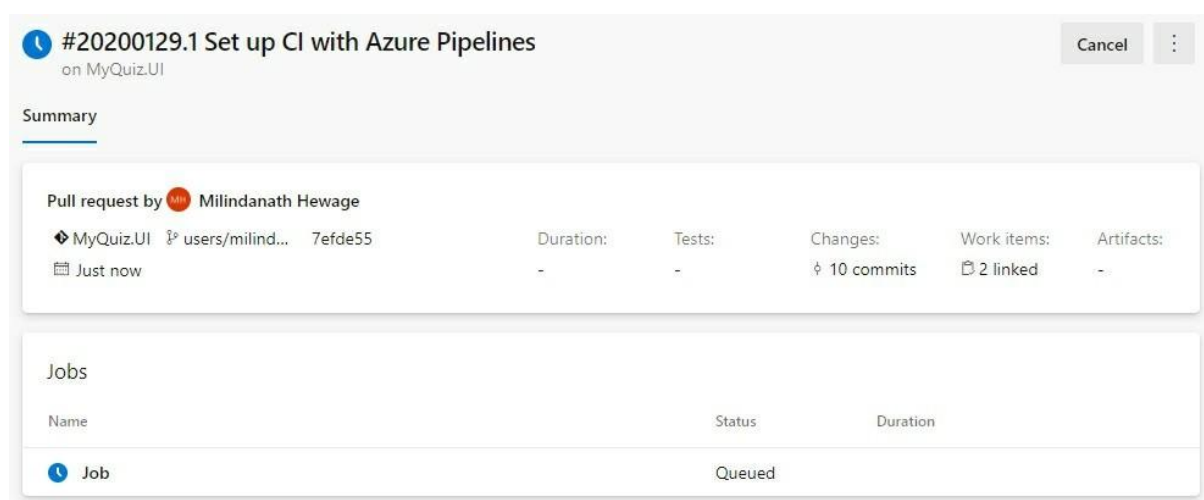
users/milindanath/ci-pipeline

Based on the master branch

☒ Start a pull request

**Save and run**

When your build pipeline runs, the build agent begins one or more jobs. In this case, we have only one job, and it starts under the section **Jobs** as shown in Figure 124. Click on **Job** to see the ongoing build process.



#20200129.1 Set up CI with Azure Pipelines  
on MyQuiz.UI

Cancel ⋮

**Summary**

Pull request by Milindanath Hewage

MyQuiz.UI users/milind... 7efde55

Just now

Duration:	Tests:	Changes:	Work items:	Artifacts:
-	-	10 commits	2 linked	-

**Jobs**

Name	Status	Duration
Job	Queued	

← Jobs in run #20200201.1

MyQuiz.UI

Jobs

✓	Job	38s
✓	Initialize job	1s
✓	Checkout MyQuiz.UI@users/milin...	1s
✓	Install Node.js	2s
✓	npm install and build	29s
✓	CopyFiles	1s
✓	PublishBuildArtifacts	1s
✓	Post-job: Checkout MyQuiz.UI@...	<1s
✓	Finalize Job	<1s
✓	Report build status	<1s

✓ Job

1 Pool: [Azure Pipelines](#)

2 Image: ubuntu-latest

3 Agent: Hosted Agent

4 Started: Today at 1:21 PM

5 Duration: 38s

6

7 ▶ Job preparation parameters

8 📁 1 artifact produced

After successful completion of the build pipeline, we end up with a deployable build artifact as shown in Figure 125. If you click on that, you can see which files will be deployed when you deploy your application.

← Artifacts

Published

Name	Size
✓ 📁 drop	251 KB
📄 drop.zip	251 KB



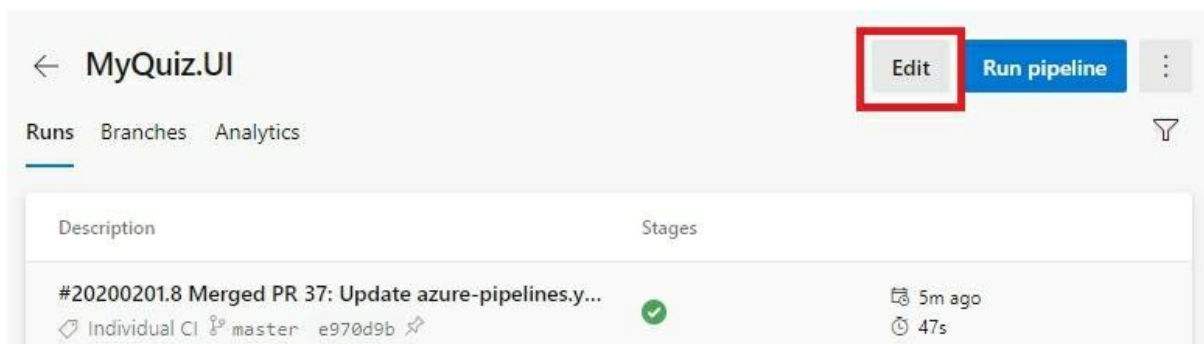
## Edit the build pipeline

If you click on the Pipelines menu on the left pane, you will see our new build pipeline as shown in Figure 127.



Figure 127: Newly created build pipeline

You can edit this pipeline by clicking on MyQuiz.UI row and then by clicking on the **Edit** button.



## Build summary

You can click on the first item in the list to see the build summary. It shows the following information related to the build

1. *who triggered the build*
2. *code repository, branch and commit in which the build was run*
3. *date and time of the build*
4. *duration it took to run*
5. *how many commits involved in the build*
6. *how many work items linked*
7. *the produced artifact*

These are highlighted in the Figure 129.

The screenshot shows the summary of a successful build. At the top, it says "#20200203.1 Merged PR 39: Update azure-pipelines.yml for Azure Pipelines" with a green checkmark icon and a "Run new" button. Below this, there are tabs for "Summary" and "Releases". The main section displays build details: "Triggered by" (Milindanath Hewage), "MyQuiz.UI" (master 019ecb7), and "Feb 3 at 9:56 PM". To the right, there are statistics: "Duration: 1m 1s", "Tests: Get started", "Changes: 2 commits", "Work items: 1 linked", and "Artifacts: 1 published". Below this, there is a table titled "Jobs" with columns "Name", "Status", and "Duration". The table shows one job named "Job" with a status of "Success" and a duration of "56s".

Name	Status	Duration
Job	Success	56s

## Approve pull request

If you examine your master branch, you will notice that the yaml configuration file for our build is not yet added to the master branch. It is waiting for the approval through the pull request created while saving our build pipeline. Once you approve the pull request, you can see that the yaml file will be part of your source code.

Having the build configuration file together with the source code is a very nice feature. This gives you the possibility to go back to previous versions of your source code at any given time and build the project without any problems using the configuration you used in that exact same version. In other words, you can version control your build pipeline.

## Disable the pipeline

If you want to disable / pause the build pipeline, you can do it through the edit page. Click on the options button on the right and select **Settings**.



Inside the settings page, you can select either paused or disabled option to disable the build pipeline. Click on the **Save** button to save your changes.

**Pipeline settings**

Processing of new run requests

☐ Enabled

☐ Paused

☒ Disabled

YAML file path

azure-pipelines.yml

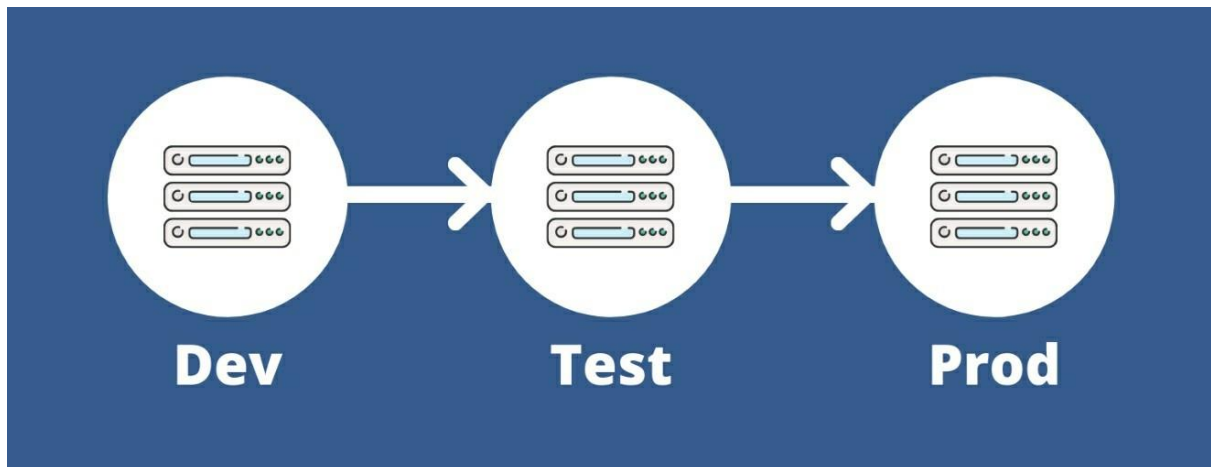
Cancel Save

## Continuous Delivery (CD)

We have automated our build process using the build pipeline. So, the next step is to automate the deployment process using a release pipeline (CD pipeline). Before creating the release pipeline, you will have to design your release pipeline.

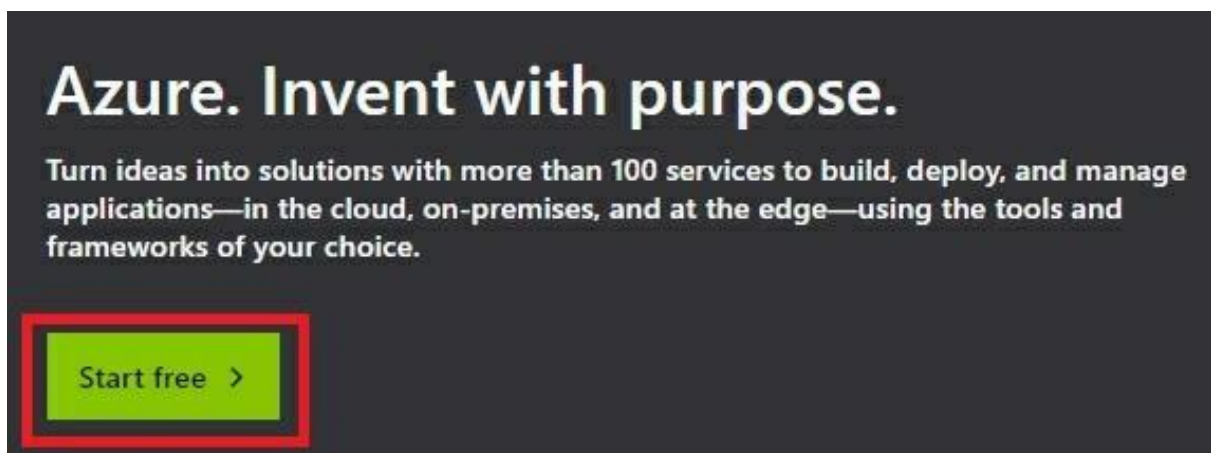
### Release environment

In the example shown in Figure 130, we have 3 *stages/deployment phases* in the release pipeline. First a **Dev** environment where you deploy the build artifacts and perform initial testing. Then, you deploy it to the **Test** environment where your test team quality assures the application thorough testing before deploying to production. You can also have a staging environment between test and production (Although I have skipped in this example).



The release environments which are connected to these 3 stages can come in different forms based on your preference. It could be an IIS web app on an on-premise server/Virtual Machine, a containerized environment like Kubernetes, a managed service like Azure App service, or a serverless environment like Azure functions. Let us use Azure App service to deploy our application.

You can navigate to <https://azure.microsoft.com/en-us/> to access the Microsoft Azure web site. If you have not created an Azure account yet, you can create a new account for free.



Once you are finished with the account creation, navigate to <https://portal.azure.com/#home> to access the Azure portal. Click on the **Create a resource** link on the home page.

## Azure services



Figure 132: Create a new resource in Azure

Now find out the **Web app** option from the next window and click on it.



Ubuntu Server 18.04 LTS

[Learn more](#)



Web App

[Quickstart tutorial](#)

Follow these steps to complete the rest of the process.

1. Select your Azure subscription
2. Create a new resource group by clicking on the Create new link

Select existing...

Create new

A resource group is a container that holds related resources for an Azure solution.

Name \*

myquiz-resource-group ✓

OK Cancel

1. Next, provide a name for your web app for the Dev environment. Let us say we want to call our dev url *milindanath-myquiz- dev.azurewebsites.net*.
2. Then, select a runtime stack that matches the application. As we developed our application in node.js version 12, select Node 12 LTS.

Instance Details

Name \* milindanath-myquiz-dev ✓  
 .azurewebsites.net

Publish \* Code Docker Container

Runtime stack \* Select a runtime stack. ^

Operating System \* Node

Region \* Node 12 LTS  
 Node 10 LTS

1. Now, select an operating system.
2. Then, select a region matching and close to your area.
3. Create an app service plan and **remember to select a Free F1 tier** if you want to start with a basic app for free. You can change the **Sku and size** by clicking

on the **change size** link.

Windows Plan (North Europe) \* ⓘ

(New) myquiz-app-service-plan

[Create new](#)

Sku and size \*

**Standard S1**

100 total ACU, 1.75 GB memory

[Change size](#)

## Spec Picker



**Dev / Test**

For less demanding workloads

## Recommended pricing tiers

**F1**

Shared infrastructure

1 GB memory

60 minutes/day compute

Free

1. After everything is filled in, click on the **Review + create** button to create the app service for your dev environment.

Sku and size \*

**Free F1**

Shared infrastructure, 1 GB memory

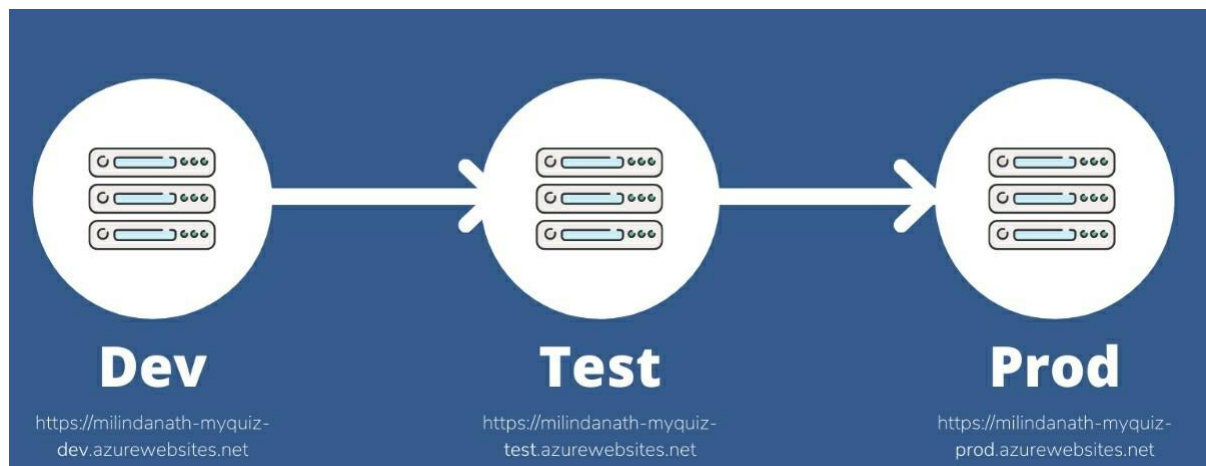
[Change size](#)

**Review + create**

< Previous

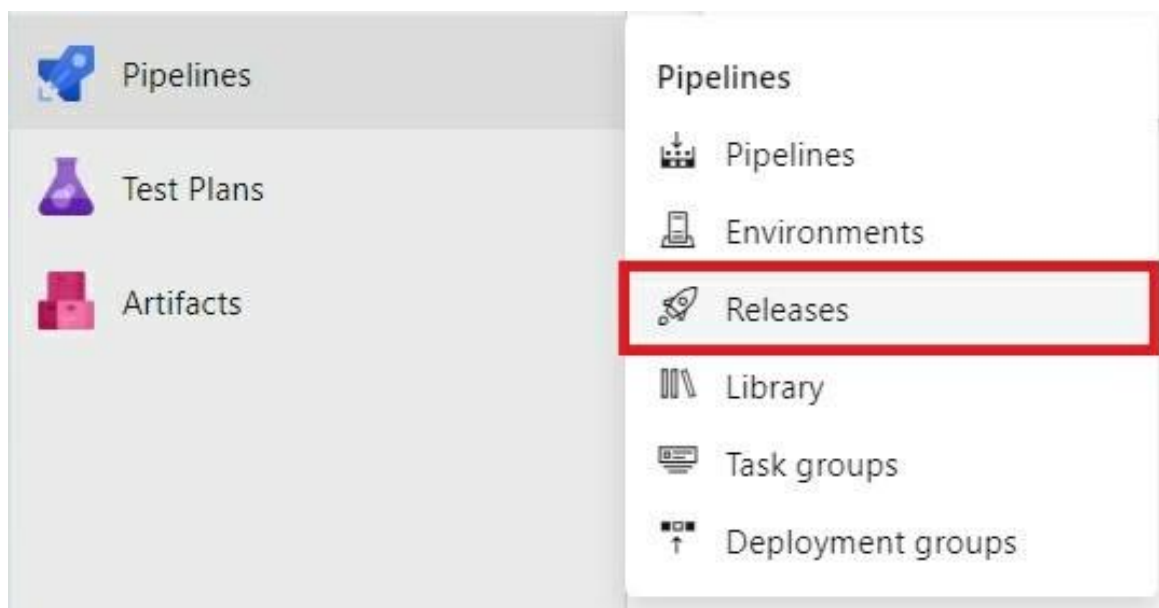
Next : Monitoring >

Repeat the same process to create the other two environments - **test** and **production**. The final setup of the environment is as shown in Figure 134.



## Create the release pipeline

Click on Pipelines → Releases to navigate to release pipeline page.



Now click on the

### New pipeline

button to create your first release pipeline.



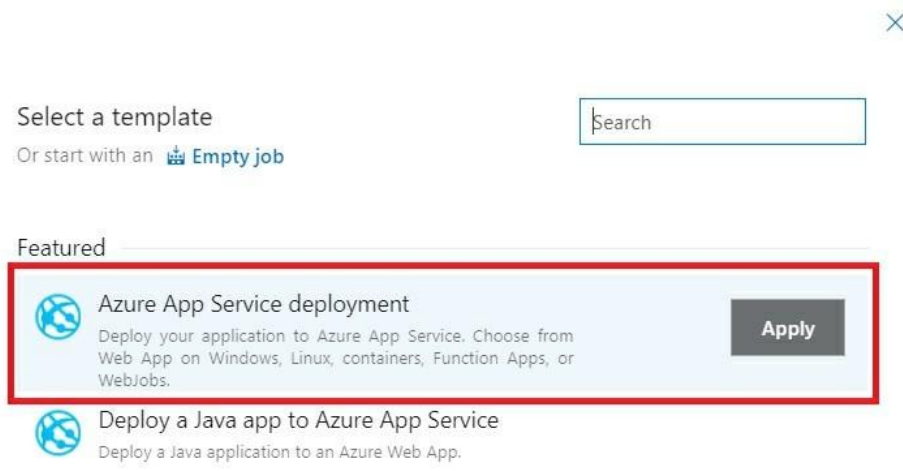


## No release pipelines found

Automate your release process in a few easy steps with a new pipeline

New pipeline

As we want to deploy our application to Azure, select the option **Azure App Service deployment** and click on the **Apply** button as shown in Figure 137.



In the next window, you have to specify to which stage you are going to deploy to. According to our plan, the first stage we want to deploy our code is **Dev**. Therefore, select **Dev** environment as shown in Figure 138.

Stage

Dev

Properties

Name and owners of the stage

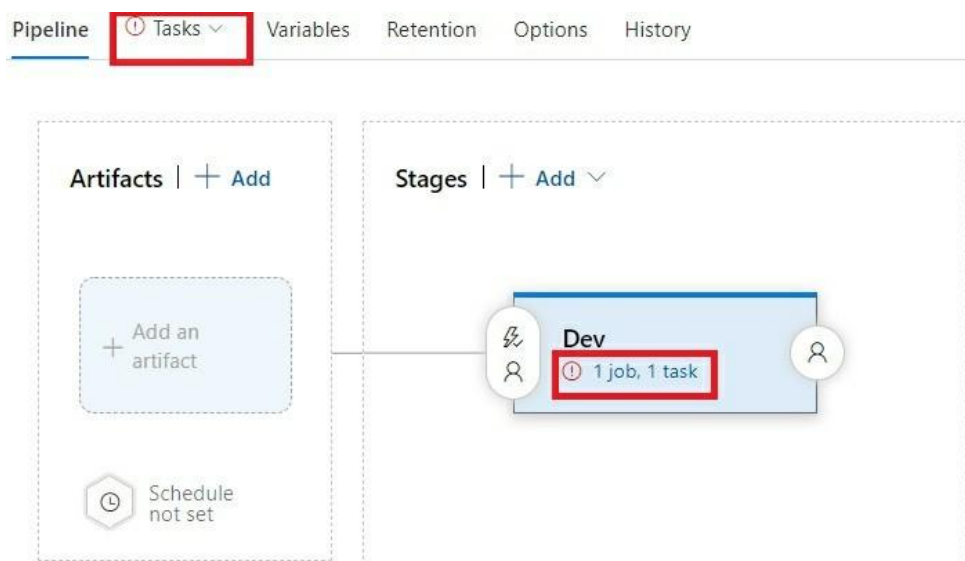
Stage name

Dev

Stage owner

Milindanath Hewage

Once you close this dialog, you can see that the Dev stage is created. Each stage has one or more jobs that runs on a release agent. You can navigate to the stage configuration page by clicking on one of the highlighted links in Figure 139.



*Figure 139: Navigate to stage configuration*

First, click on the **Dev** to setup the basic stage settings.

Pipeline Tasks Variables Retention Options History

Dev  
Some settings need attention

Run on agent  
Run on agent

Deploy Azure App Service  
Some settings need attention

Stage name  
Dev

Parameters | Unlink all

Azure subscription \* | Manage

This setting is required.

App type  
Web App on Windows

App service name \* |

This setting is required.

Here, you have to specify 2 mandatory fields. The first one is your **azure subscription**. If nothing is shown in the dropdown, click on the **Manage** link to connect your azure subscription to Azure DevOps. In the second option, select the **App service name** which was created when setting up the release environments.

App type

Web App on Windows

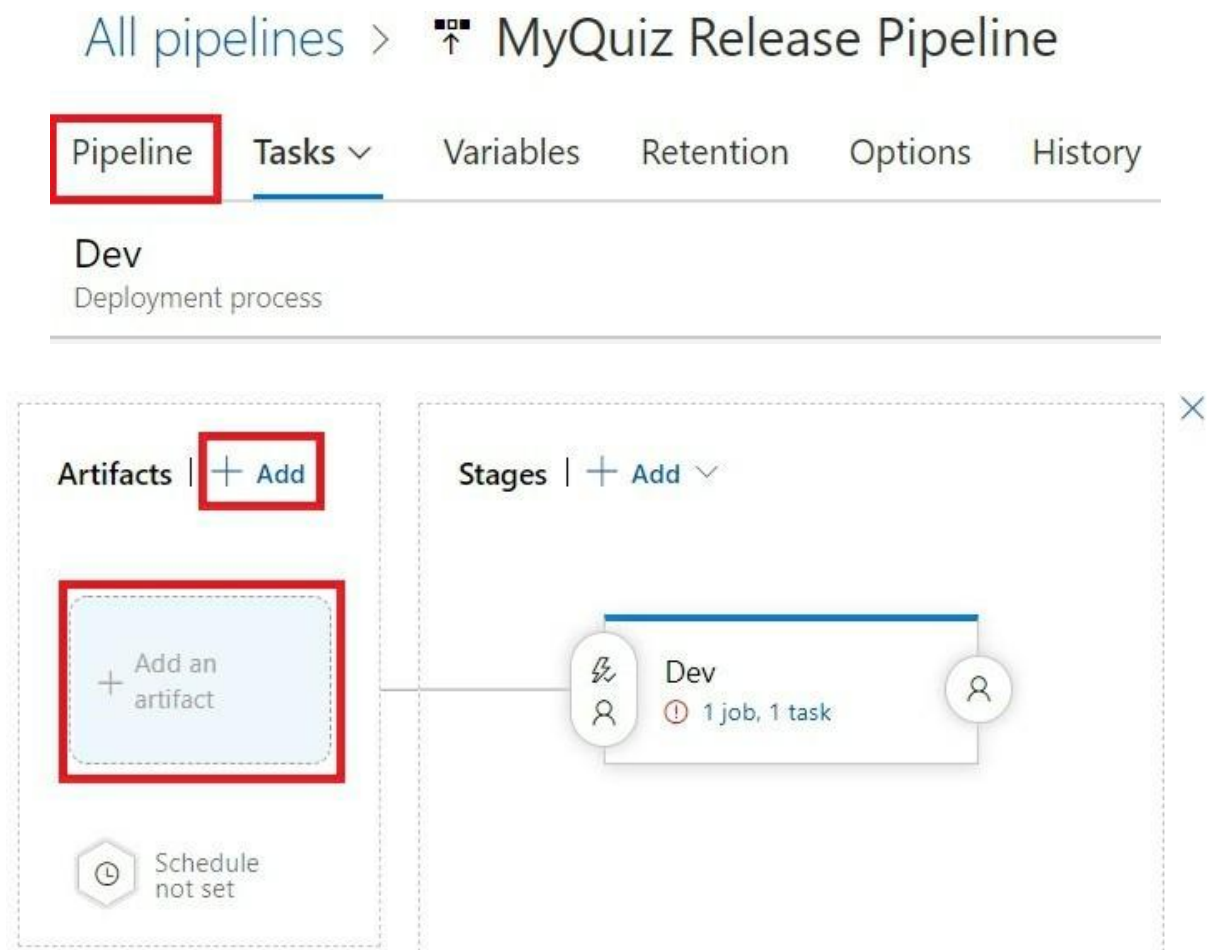
App service name \*

milindanath-myquiz-dev

Now we have successfully connected our Azure App Service Dev environment to the Dev stage in the pipeline. Now, click on the **Run on agent** section to select an agent from the **Azure Pipelines** agent pool. Let us keep the default configuration for this and move on to the task **Deploy Azure App Service**.

However, before moving even further with the Dev stage setup, we have to provide the **artifact** we created in the build pipeline as an input to the release pipeline. Click

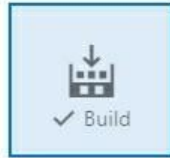
on the **Pipeline** tab and then click on add link or on the **Add an artifact** links to add this as shown in below images.



In **add an artifact** window, Select **Build** as the Source type. Then, select your project and the build pipeline name as shown in Figure 142. You can also specify which version of the artifact should be used when the release pipeline runs. Here, we take the **latest** version of the artifact.

## Add an artifact

Source type



5 more artifact types ▾

Project \* ⓘ

PracticalGuide ▾

Source (build pipeline) \* ⓘ

MyQuiz.UI ▾

Default version \* ⓘ

Latest ▾

Source alias \* ⓘ

MyQuiz\_Dist

ⓘ The artifacts published by each version will be available for deployment in release pipelines. The latest successful build of **MyQuiz.UI** published the following artifacts: **drop**.

Add

Click on the **Add** button to add the artifact as the input to the release pipeline.

## Trigger release pipeline

One of the key features in DevOps automation is continuously deliver your product to the customers. In order to do that, we have to enable continuous integration for your release pipeline. We can do it in two ways as discussed below.

### Method 1: By scheduling a new release at a specific time

In this page, you have the possibility to run your release pipeline on a regular basis. For example, suppose you want to run your release every **Tuesday at 03:00 a.m** whenever there is a new build available. In that case, you can click on the button **Schedule** as shown in Figure 143.



In the next window, enable **Create a new release at the specified times** option, and set the times as shown in Figure 144.

### Scheduled release trigger

Define schedules to trigger releases

☒ Enabled  
Create a new release at the specified times

🕒 Tue at 3:00 ^

☐ Mon ☒ Tue ☐ Wed ☐ Thu ☐ Fri ☐ Sat ☐ Sun

03h ▾ 00m ▾

(UTC) Coordinated Universal Time ▾

☒ Only schedule releases if the source or pipeline has changed

+ Add a new time

## Method 2: Each time a new build is available

The other method is to trigger the release, each time the build pipeline produces a new artifact. If you revisit the build pipeline, it will produce an artifact each time you commit changes to the master branch. Click on the button **Continuous deployment trigger** button.



Enable the **Continuous deployment trigger** option to enable this feature. By default, this will select the master branch to trigger this event.

## Continuous deployment trigger

Build: MyQuiz\_Dist

☒ Enabled

Creates a release every time a new build is available.

Build branch filters ⓘ

No filters added.

|

Suppose you do not want to trigger a release for the master branch build, but for another branch, then you can use the **Build branch filters** option. For example, if you want to trigger a release each time you create a branch under the **releases** folder, then you can do it as shown in Figure 147.

## Continuous deployment trigger

Build: MyQuiz\_Dist

☒ Enabled

Creates a release every time a new build is available.

### Build branch filters ⓘ


Type	Build branch	Build tags
Include ▾	 releases/* ▾	<div></div> 
<div> Add ▾</div>		

## Finalize the Azure app service task


Now, we can go back to our tasks list, and finalize the Deploy to azure app service task. So, click on the Tasks menu, and then click on the Deploy to Azure App Service task. What is important here is to provide the path to your drop.zip file.




① Scoped to subscription 'Pay-As-You-Go'


App Service type \* 


Web App on Windows


App Service name \* 

milindanath-myquiz-dev

☐ Deploy to Slot or App Service Environment 

Virtual application 

Package or folder \* 



## Rename the build pipeline

Now, we have setup our release pipeline. Before saving, let us rename our pipeline to a suitable name. I am going to call it **MyQuiz Release Pipeline**. To do that, click on the current name which is shown on the breadcrumb on the top.



## Release options and variables

In Azure pipelines, you can make use of variables to contain variable data that can be used in different places. There are pre-defined variables defined

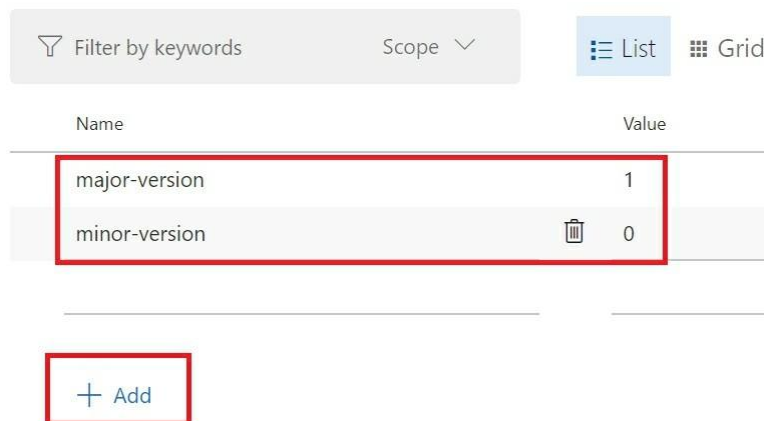
by Azure DevOps. For example, the variables we used earlier such as

**\$(Build.SourcesDirectory)** are pre-defined variables. Moreover, you can create your own custom variables.

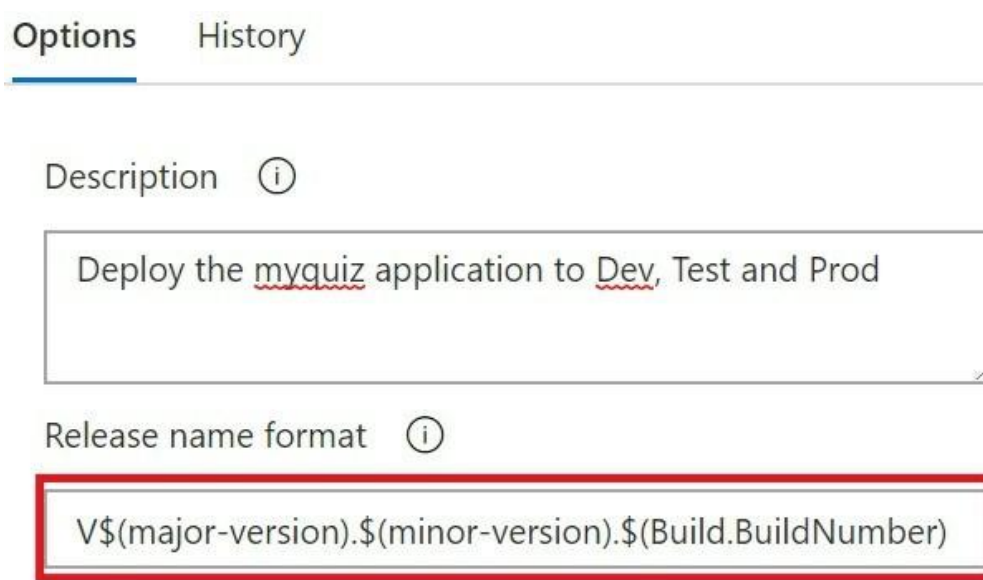
Click on the **Variables** tab in your release pipeline to create some custom variables.



Click on the **Add** button to create variables to represent the major and minor versions of the release.



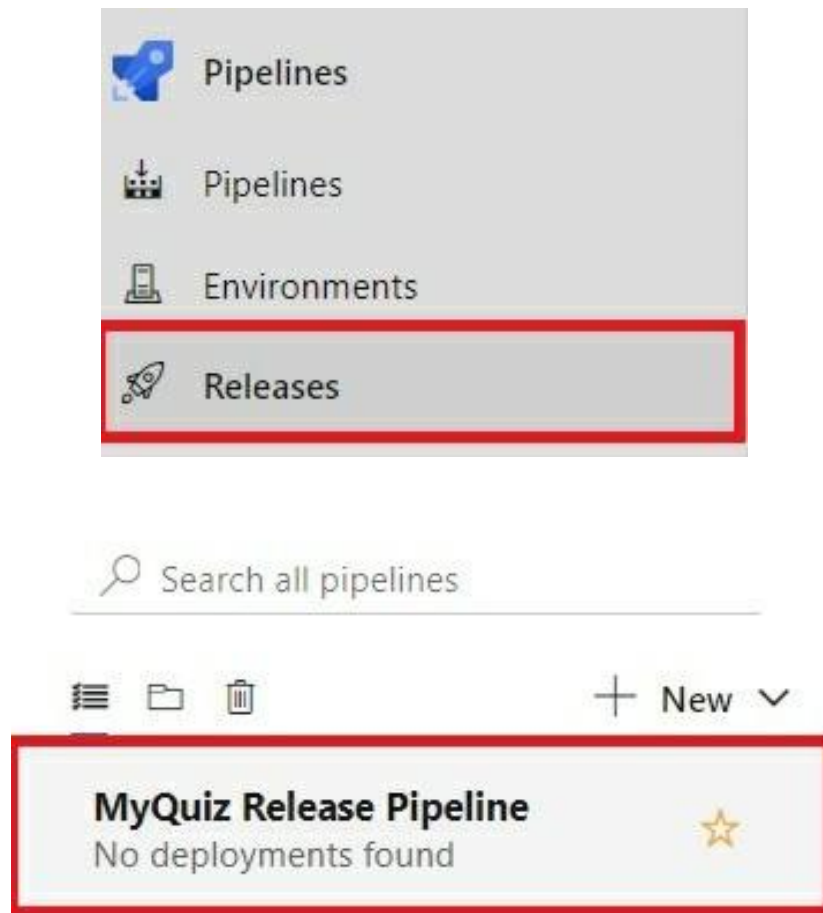
Now we can change the format of the release name using these custom variables. You can set additional information such as the format of the release name under the tab **Options**. Here, we combine the two custom variables with the pre-defined variable **\$(Build.BuildNumber)** to create a unique name for the release.



Now click on the **Save** button to save all the changes done to the pipeline.

## Edit release pipeline

You can edit your pipeline by navigating to the **Releases** section from the **Pipelines** navigation pane. Now select the newly created pipeline and click on the Edit button.

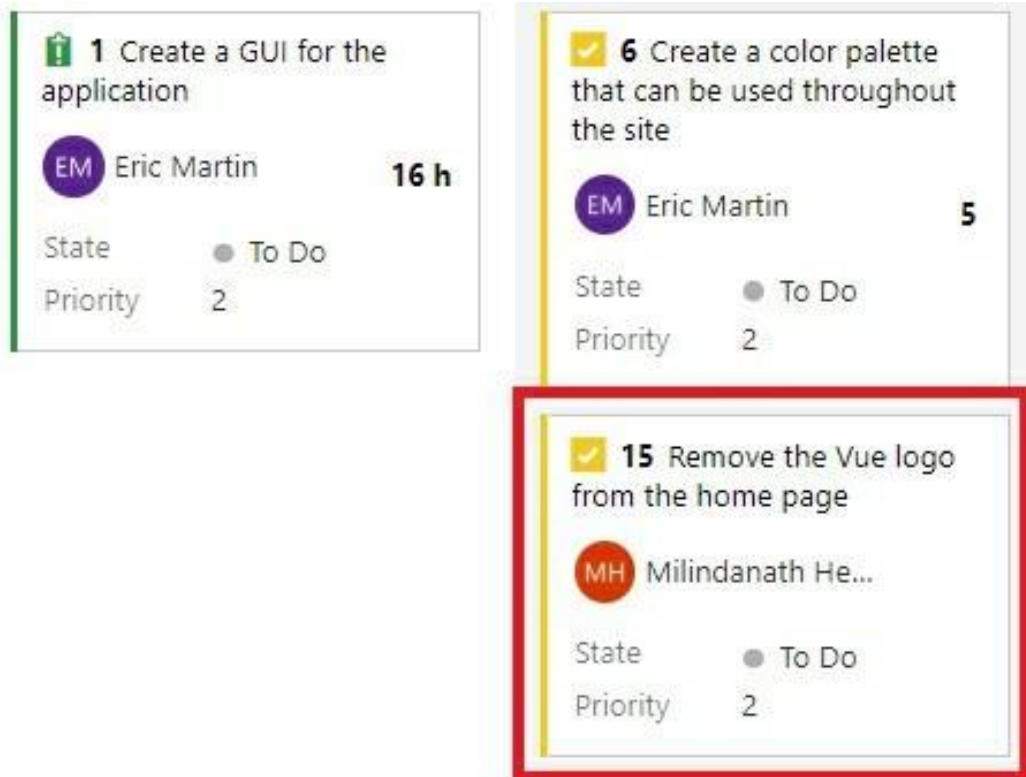


## End to end testing of the pipelines

As we have finalized setting up both the build and release pipelines, let us try to do a full end to end test to check that the whole process we defined so far works as expected.

## Create a new task

Suppose, one of the testers in our team finds out that our home page has the Vue.js logo. So, he creates a task to fix this issue.



## Create a new branch for the task

The next step is to create a new branch for the task. Let us do it in VS Code terminal using the following git command.

```
$ git checkout -b bug/15
```

## Fix the bug

Navigate to the views folder and open Home.vue file and remove the img tag to fix the issue.

```
<template>
  <div class="home">
    
    <HelloWorld msg="Welcome to Your Vue.js App"/>
  </div>
</template>
```

## Commit and push changes

Commit and push your changes to the remote repository.

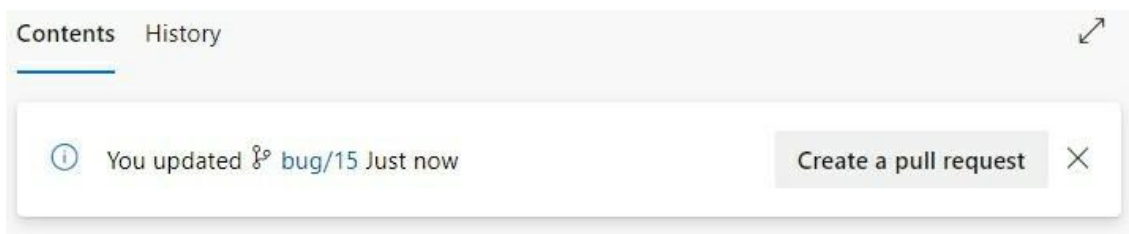
```
$ git add src/views/Home.vue
```

```
$ git commit -m "#15 Removed the Vue logo"
```

```
$ git push origin bug/15
```

## Create the pull request

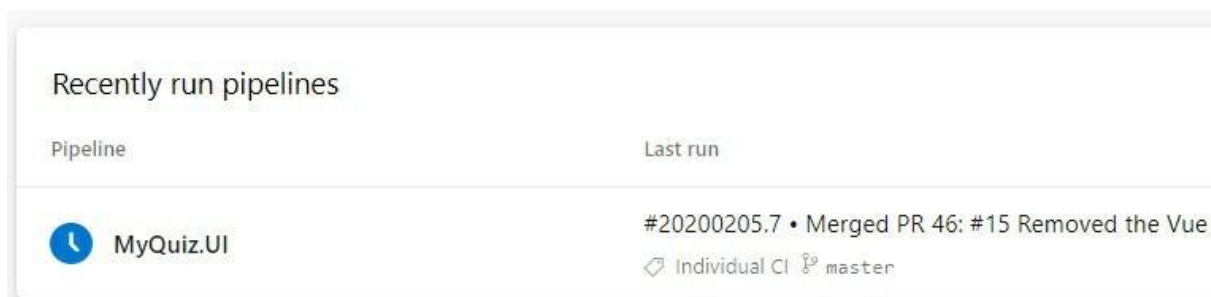
Now, go back to your Azure DevOps services and click on Azure Repos. You will see the option to Create a pull request. Click on the button.



Next, approve your changes to accept the changes and merge them into the master branch.

## Build pipeline kicks off

Remember that we have setup our build pipeline so that every commit to the master branch will trigger the build pipeline. Let us find out if it kicks off the build. This can be seen in Figure 158.



## Release pipeline kicks off next

After the build pipeline succeeded, you will see that our release pipeline kicks off, and publish the application to Azure App Service. If everything has gone well, we get a green status on the Dev stage as shown in Figure 159

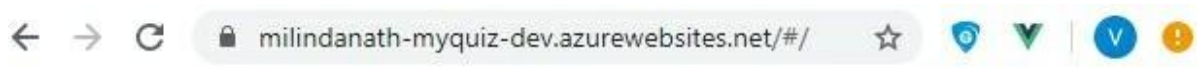


Click on the **Dev** button to inspect what has happened during the Deployment process. In this example, the release agent downloaded the artifact and published it to the Azure App service which is located at <https://milindanath-myquiz-dev.azurewebsites.net/#/>.

The screenshot shows the Azure Pipelines interface for a pipeline named 'MyQuiz Release Pipeline'. The current view is 'Release-1 > Dev', which is marked as 'Succeeded'. The 'Logs' tab is selected, showing the 'Run on agent' task. The task details include:

- Run on agent** (Succeeded) - Started: 2/5/2020, 11:26:58 PM
- Pool: Azure Pipelines · Agent: Hosted Agent · 30s
- Task steps:
  - Initialize job · succeeded (4s)
  - Download artifact - MyQuiz.Dist - drop · succeeded (2s)
  - Deploy myquiz to Azure App Service · succeeded (22s)
  - Finalize Job · succeeded (<1s)

This can be verified by navigating to the dev URL located at <https://milindanath-myquiz-dev.azurewebsites.net/#/> . You can see that the bug #15 is fixed and deployed.



[Home](#) | [About](#)

# Hello world, Welcome to MyQuiz

## Question 1: Who is the founder of Microsoft?

- ☐ A: Bill Gates
- ☐ B: Satya Nadella
- ☐ C: Steve Jobs
- ☐ D: Mark Zuckerberg

Congratulations! Now you have implemented end-to-end automation to your **Dev environment** using Azure Boards, Repos and pipelines.

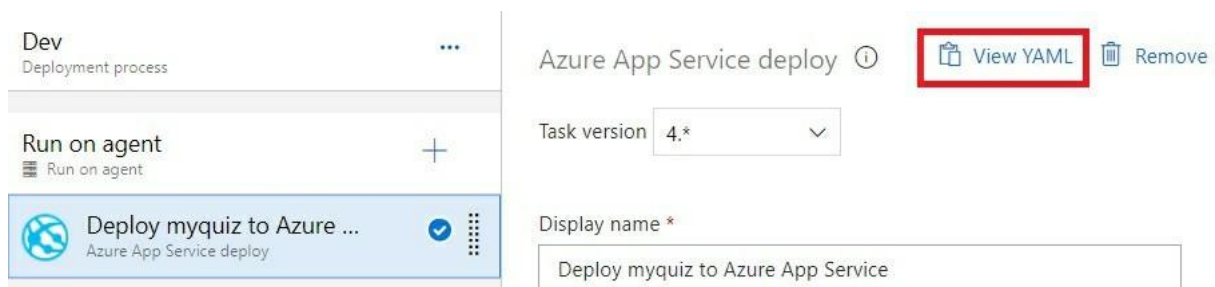
## Combine the pipelines

Probably you have noticed that there is an issue with our release pipeline. In the build pipeline, we created a `.yaml` file that defines the build process in code. That file is committed to the source control and get versioned with the build. Unfortunately, we do not have that facility in the release pipeline yet.

However, there is a method to write the release pipeline in yaml. It is by incorporating the release process into the build pipeline. Let us try to do that.

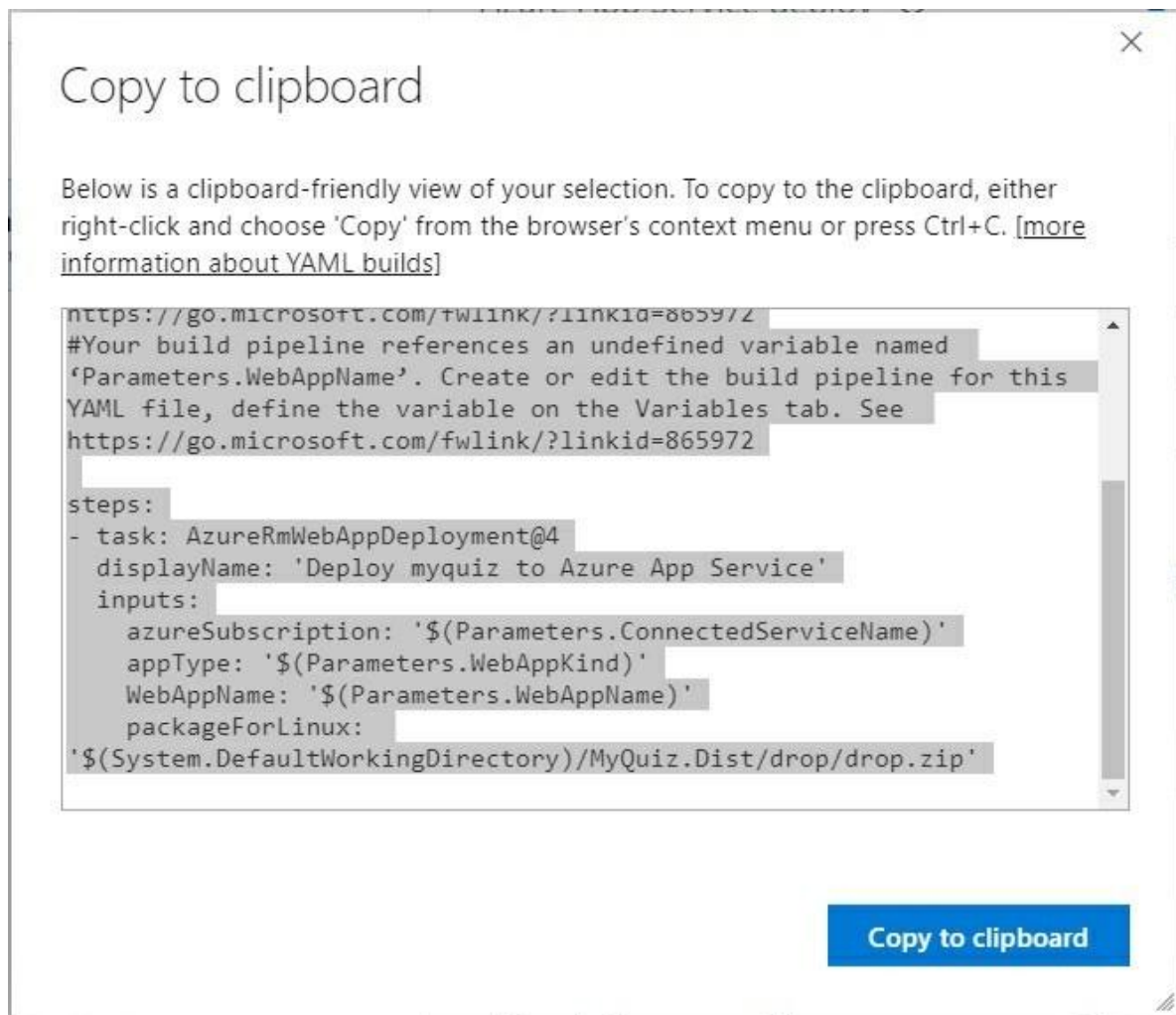
## Extract the yaml code from the release pipeline

Navigate to your release pipeline and click on the Edit button. Now go to the Dev stage to view the tasks. Click on the **Deploy myquiz to Azure App Service** task. Now click on the link **View YAML**.



Here, you can copy the web deployment step to clipboard. Click on the **Copy to clipboard** button.





There are basically 3 variables you have to note down here.

1. `$(Parameters.ConnectedServiceName)`, which is your azure subscription name
2. `$(Parameters.WebAppKind)` = webApp
3. `$(Parameters.WebAppName)` = milindanath-myquiz-dev

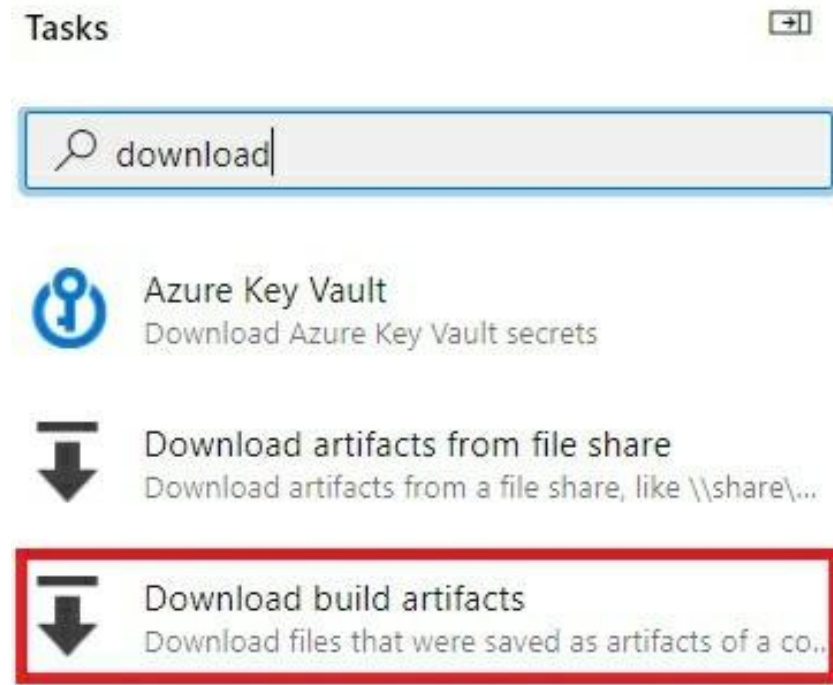
WebAppKind variable can have different values based on the deployment environment. Here are some of the options available.

1. webApp (used in this example)
2. webAppLinux
3. webAppContainer
4. functionApp
5. functionAppLinux
6. functionAppContainer

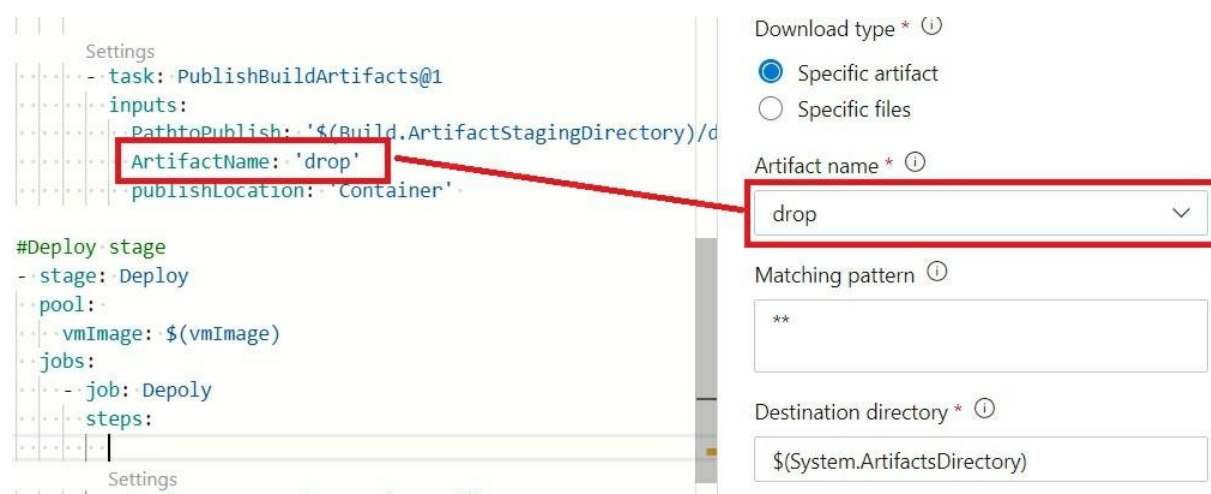


## Add the yaml code to the build pipeline

First, open your build pipeline definition. Before pasting the yaml code copied to the clipboard, you have to add another task that downloads the zip files from the *artifact staging directory* to the *artifacts directory*. So, search for the **Download build artifacts** task as shown in Figure 163.



Here, we download the drop artifact from the Artifact staging directory to the Artifact directory.



The final pipeline definition are as follows. The highlighted text are the new changes added to the previous build pipeline definition.

```

# Node.js with Vue

# Build and deploy a Node.js project that uses Vue.

# Add steps that analyze code, save build artifacts, deploy, and more:
# https://docs.microsoft.com/azure/devops/pipelines/languages/javascript

trigger:

  • master

variables:

Parameters.ConnectedServiceName: <<your_azure_subscription>>
Parameters.WebAppKind: webApp

Parameters.WebAppName: milindanath-myquiz-dev vmImage: 'ubuntu-latest'

stages:

#Build stage

  • stage: Build pool:

vmImage: $(vmImage)

demands:

  • npm jobs:

  • job: Build steps:
    • task: NodeTool@0 inputs:

versionSpec: '10.x' displayName: 'Install Node.js'

  • script: | npm install

npm run build

displayName: 'npm install and build'

  • task: CopyFiles@2 inputs:

SourceFolder: '$(Build.SourcesDirectory)' Contents: 'dist/**'

TargetFolder: '$(Build.ArtifactStagingDirectory)' CleanTargetFolder: true

  • task: ArchiveFiles@2 inputs:

rootFolderOrFile: '$(Build.ArtifactStagingDirectory)/dist' includeRootFolder: false

archiveType: 'zip'

```

archiveFile: '\$(Build.ArtifactStagingDirectory)/drop.zip' replaceExistingArchive: true

- task: PublishBuildArtifacts@1 inputs:

PathToPublish: '\$(Build.ArtifactStagingDirectory)/drop.zip' ArtifactName: 'drop'

publishLocation: 'Container'

#Deploy stage

- stage: Deploy pool:

vmImage: \$(vmImage)

jobs:

- job: Depoly steps:
  - task: DownloadBuildArtifacts@0 inputs:

buildType: 'current' downloadType: 'single' artifactName: 'drop'

downloadPath: '\$(System.ArtifactsDirectory)'

- task: AzureRmWebAppDeployment@4 displayName: 'Deploy myquiz to Azure App Service' inputs:

azureSubscription: '\$(Parameters.ConnectedServiceName)' appType:

'\$(Parameters.WebAppKind)'

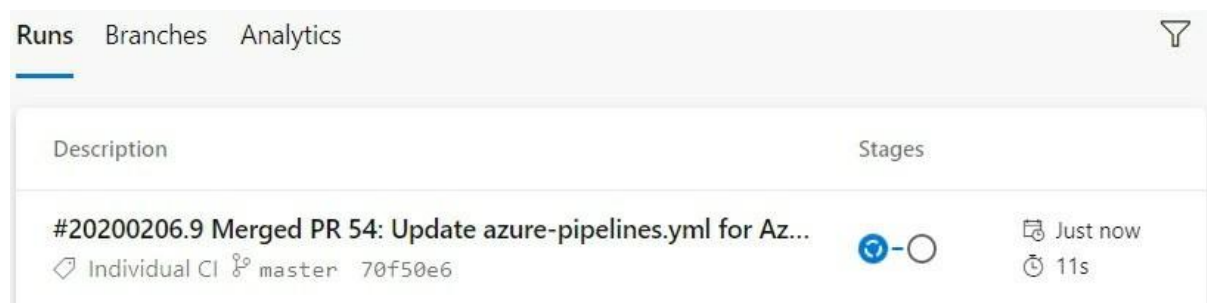
WebAppName: '\$(Parameters.WebAppName)' packageForLinux:

'\$(System.ArtifactsDirectory)/\*\*/\*.zip'

Click on **save** and create a pull request. Before you approve the pull request, go to the release pipeline and disable the **continuous deployment trigger** and

the **scheduled time** we add earlier. If you accept the pull request, you can see your code is built and deployed from one single pipeline file as shown in Figure 164. Most importantly, it will be committed to your source code.

When the build is starting it shows a progress icon as shown in Figure 164.



#20200206.3 Merged PR 49: Update az...  
on MyQuiz.UI

Cancel

SummaryReleases

Triggered by Milindanath Hewage

MyQuiz.UI master 84370db

Just now

Duration: 1m 22s

Tests: -

Changes: 6 commits

Work items: 1 linked

Artifacts: -

StagesJobs

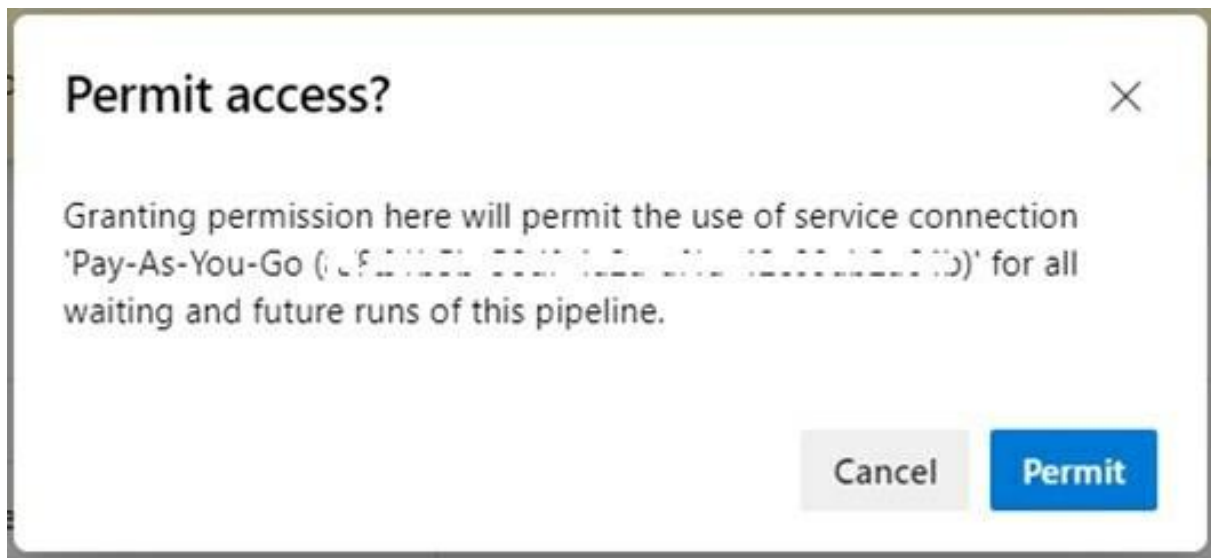
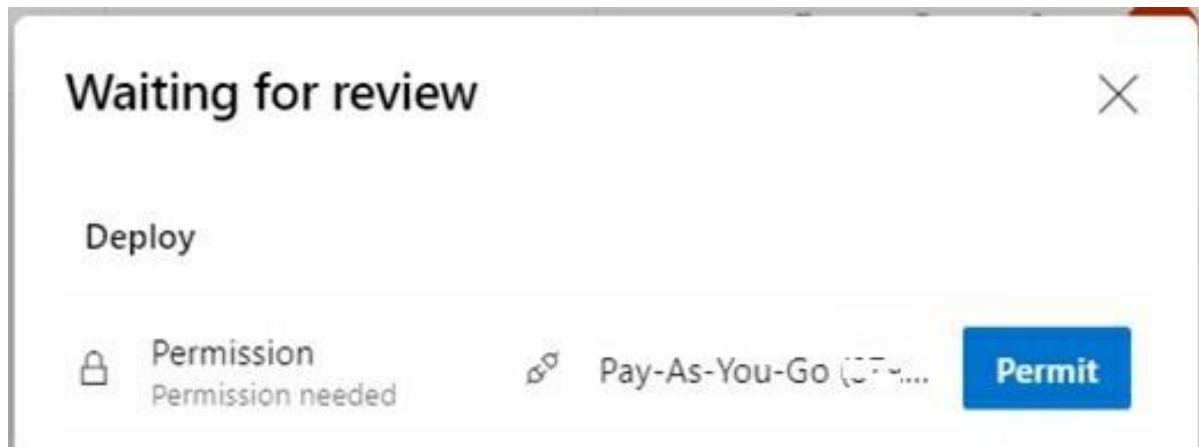
Build  
0/1 completed1m 20s

Build1m 20s

Cancel

Deploy  
Not started

If you have not authorized your azure subscription, then you have to give permission to continue to deploy to azure as shown in Figures below



If the build and deploy stages are successful, you will see green status icons as shown in Figure 168.



Figure 168: Successful runs of Build and Deploy stages

## Test and Production environments

So far, we have only setup our Dev environment which is basically used by developers to test their changes in a build environment. However, when we do a full release of the application, we need to deploy it to a **test/QA** server where the testers can do their testing routines. However, we do not want that each developer's commit to the master branch gets deployed to the test or production environment.

## Plan the release process

Suppose our team uses the trunk-based branching flow when dealing with the source code and releases. As we already have done, we create a branch when we want to try a new feature or a bug fix. Then we merge it to master branch using a pull request with code review.

We can use the same strategy when we want to do a release. Simply, create a branch off of master branch for the release and name the branch as release/1.0.



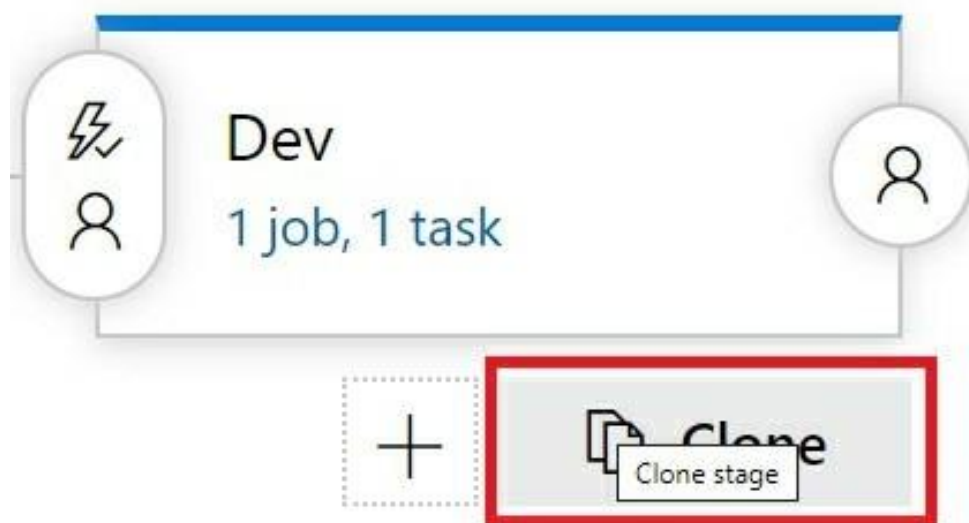
Before moving into setting up the Test and Prod environments, remember to comment out the Deploy stage of your current build pipeline.

Let us change the build pipeline so that it triggers our build on any commit to any branch. However, we want to control the release pipeline.

```
6 trigger:  
7 - . '*'
```

## Test stage

Go to the edit page of the release pipeline and clone the Dev stage.



*Figure 169: Clone the Dev stage*

Rename the stage to **Test**.

# Stage

Test


## Properties ^

Name and owners of the stage

Stage name

Test


Stage owner

 Milindanath Hewage

In the Tasks page, point the **App service name** to the correct azure test environment you created earlier.

App type 

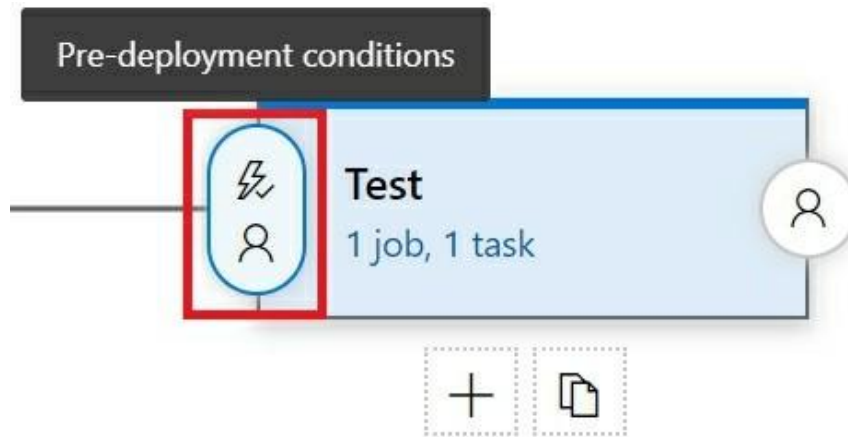
Web App on Windows

App service name \* 

milindanath-myquiz-test

Now, go back to the Pipeline tab, and click on the **pre-deployment conditions** button on the Test stage.





Select **After release** options under the **Triggers** section.

## Pre-deployment conditions

Test

### ⚡ Triggers ^

Define the trigger that will start deployment to this stage

Select trigger ⓘ



Now, enable Artifact filter and click on the Add button. Select the Artifact MyQuiz.Dist.

## Artifact filters ⓘ

+ Add ▾ ☒ Enabled

No filters added.

MyQuiz.Dist

Figure 174: Artifact filters

Type in **releases/\*** and press the enter key on the keyboard.

ⓘ MyQuiz.Dist ^

Type

Build branch

Build tags

Include ▾

Select a branch... ▾

ⓘ Specify branch or tag

Mine

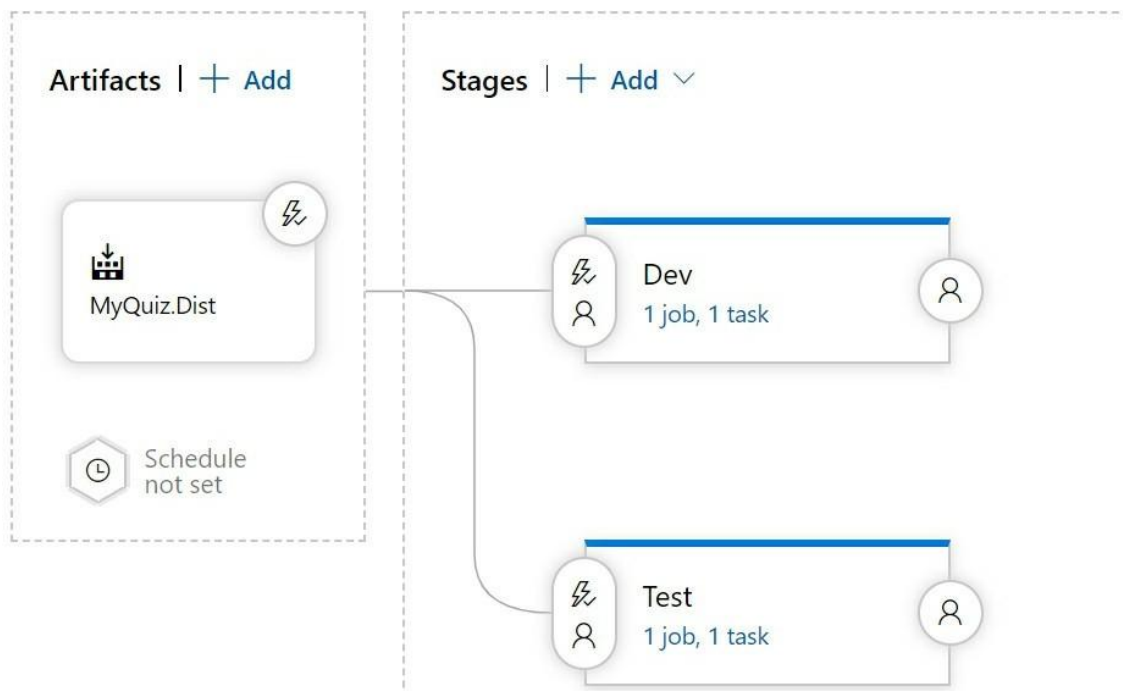
All branches

releases/\*

Press Enter to search in 'All branches'

+ Add

Close the window by clicking on the X button. Now you will see two stages are in parallel.






However, the Test stage will only run when you create a branch under the path **releases/**. Dev stage will be triggered as usual for all the changes in any branch including the release branch.

Save everything and create a new branch off master branch. Create a pull request and merge it to master branch. In both cases, you will see that only Dev release will occur as shown in Figure 177.

Created	Stages
2/7/2020, 12:22:20 AM	<div> <input checked="" type="checkbox"/> Dev         </div> <div> <input type="checkbox"/> Test         </div>

Now, let us create a new branch from the master and name it **releases/6**. After you create the branch, the build pipeline will kick off immediately.

Description	Stages	
#20200206.12 Merged PR 57: Update azure-pi... Individual CI  releases/6 38b4ef2		<div>  Just now         </div> <div>  41s         </div>

Not only that, it will deploy to both **Dev** and **Test** after the build is succeeded.

Releases	Created	Stages
<div>  <b>Release-6</b>   20200206...  releases/6         </div>	2/7/2020, 12:27:36 AM	<div> <input checked="" type="checkbox"/> Dev         </div> <div> <input checked="" type="checkbox"/> Test         </div>


 milindanath-myquiz-test.azurewebsites.net/#/

[Home](#) | [About](#)

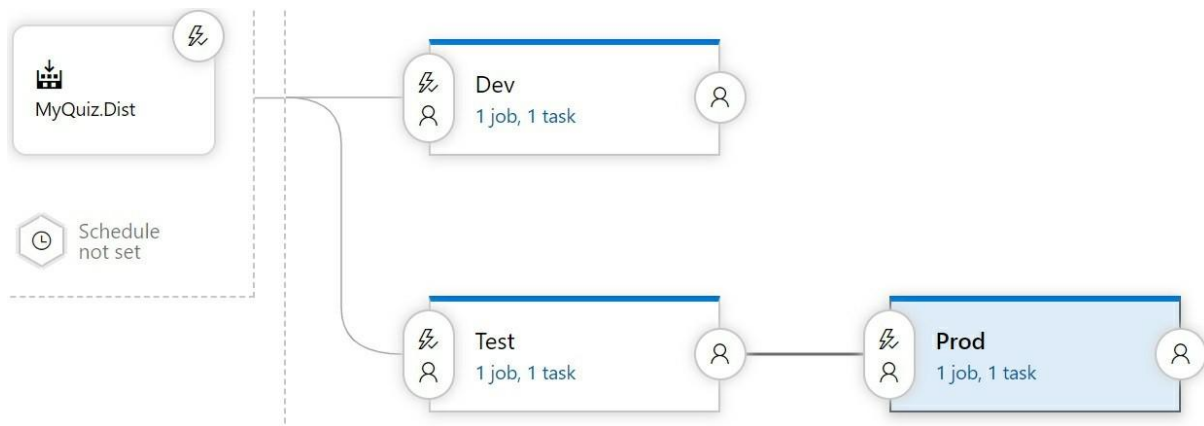
# Hello world, Welcome to MyQuiz

## Question 1: Who is the founder of Microsoft?

- ☐ A: Bill Gates
- ☐ B: Satya Nadella
- ☐ C: Steve Jobs
- ☐ D: Mark Zuckerberg

**Production stage**

The testing process is done, and it is time to deploy to production. So, we need to create a new stage for the prod environment. Clone the Test stage and rename it to Prod. Set the App service name to your production azure app service (here it is *milindanath-myquiz-prod*). Now the stages look like the following.



However, there is a problem with this setup. With this setup, the application will be deployed to both test and production each time you create a release branch. We do not want that to happen. So, we need some control here.

## Approvals

This can be achieved by having approvals at certain key stages in the pipeline. For example, suppose your test team performs testing in the Test environment. Once they are satisfied with the testing, the leader of the test team or whoever responsible for testing, can approve the release to go forward. Let us see how we can achieve this.

Click on the **post-deployment conditions** button for the Test stage as shown in Figure 182



Now enable post deployment approvals and select the test leader as the approver. Here, you can also setup approval policies as shown in Figure 183.

🔍 Post-deployment approvals ^ Enabled

Select the users who can approve or reject deployments to this stage

Approvers ⓘ

Test responsible

VE Viveka Edirisinghe ✕

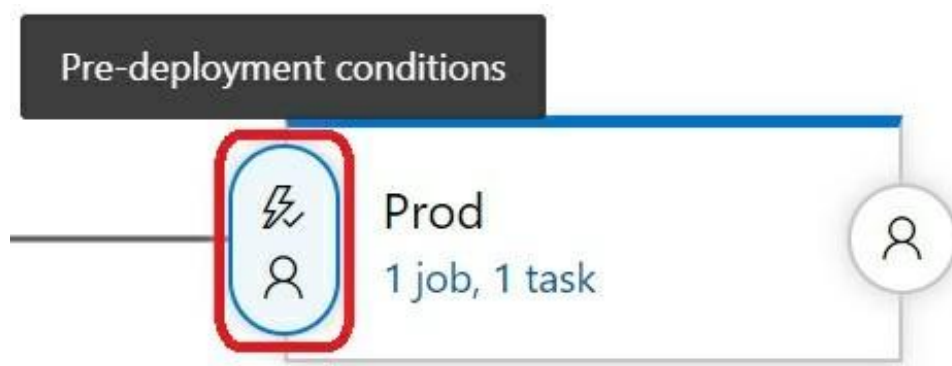
Search users and groups for approvers

Timeout ⓘ




Approval policies

- ☐ The user requesting a release or deployment should not approve it
- ☐ Revalidate identity of approver before completing the approval. ⓘ


Based on this, the release pipeline is paused at the Test stage until the test leader gives her permission to release to the production. Now, we have some control over the production release. But this control along might not be enough to release to the production. Probably, you need to perform some actions prior to every production release. For an example, your **Database Administrator (DBA)** wants to run the release scripts and other checks prior to production. We can setup the pre-deployment conditions of the Prod stage to achieve this.






Here you can assign your DBA as the approver, and without his clear signal the release will not go forward.


 Pre-deployment approvals   Enabled

Select the users who can approve or reject deployments to this stage



Approvers 

 Eric Martin  Search users and groups for approvers



Timeout 



Days 

Approval policies

- ☐ The user requesting a release or deployment should not approve it
- ☐ Revalidate identity of approver before completing the approval. 
- ☐ Skip approval if the same approver approved the previous stage 



Under the triggers section, you have the possibility to schedule the release. For example, if you want your releases to automatically be deployed on a Tuesday at 23:00 local time, then you can set it as below.


Schedule   Enabled

 Tue at 23:00 

☐ Mon ☒ Tue ☐ Wed ☐ Thu ☐ Fri ☐ Sat

☐ Sun



Even after the release, you can take some actions. For example, you might want to do things like checking if there are any alerts from the deployed environment after the deployment. As we are using Azure to deploy our application, we can add a gate to check for any Azure Monitor alerts as shown in Figure 187.



→] Gates ^ Enabled

Define gates to evaluate after the deployment. [Learn more](#)

The delay before evaluation ⓘ

Minutes ▾


Deployment gates ⓘ + Add ▾

 **Query Azure Monitor alerts** Enabled 

Query Azure Monitor alerts ⓘ

Task version  ▾

Display name \*

Azure subscription \* ⓘ | [Manage](#) 

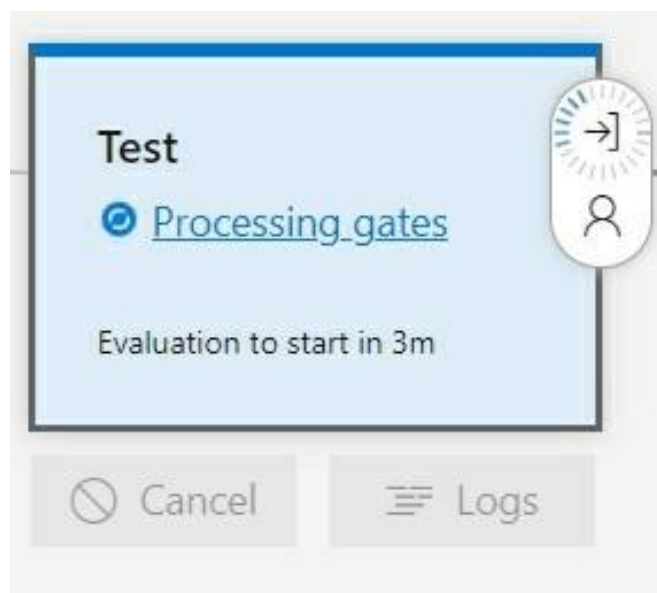
') ▾ ↻

ⓘ Scoped to subscription 'Pay-As-You-Go'

Resource group \* ⓘ

▾ ↻

Create a release and try to check all the conditions. You will see windows like the following where things need to be approved.



In this way, you have full control over your release process and automating makes your life easier as a developer, release manager or any other involved in the process. In other words, the whole organization develops a DevOps culture that will unite people, processes and products which allows continuous delivery of high-quality value to your customers.

## Summary

In this chapter about Azure Pipelines, you learned the most vital section in the DevOps process. You created a build pipeline that builds your application on a build agent. We used yaml as a data serialization language to define the build definition. Then you created a release pipeline that is used to take the output of the build pipeline as an input and deploy it to various environments such as dev, test and production. Approvals, triggers and gates help us to have control over the full release process.