

Azure DevOps

Hands-on Lab 2

Introduction to YAML Pipelines

DISCLSIMER

© 2023 Microsoft Corporation. All rights reserved. Microsoft, Windows and other product names are or may be registered trademarks and/or trademarks in the U.S. and/or other countries.

The information herein is for informational purposes only and represents the current view of Microsoft Corporation or any Microsoft Group affiliate as of the date of this presentation. Because Microsoft must respond to changing market conditions, it should not be interpreted to be a commitment on the part of Microsoft, and Microsoft cannot guarantee the accuracy of any information provided after the date of this presentation.

MICROSOFT MAKES NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, AS TO THE INFORMATION IN THIS PRESENTATION.

CONFIDENTIALITY

© 2023 Microsoft Corporation. All rights reserved. Any use or distribution of these materials without express authorization of Microsoft Corp. is strictly prohibited.

Contents

Lab 2 - Introduction to Azure DevOps Pipelines	4
Exercise 1- Create Classic Pipelines for your Python application (40min)	5
Module 1 – Create a Build pipeline for Python Application	5
Module 2 – Troubleshoot Build pipeline errors.....	7
Module 3 – Fix Build pipeline errors.....	9
Module 4 – Create Python Package and Publish (upload) artefacts.....	10
Summary.....	12
Exercise 2- Deploy Python Application to Azure using Classic Mode (15min)	13
Module 1 – Create a Release pipeline for Python Application	13
Module 2 – View your Python App in Azure.....	15
Summary.....	15
Exercise 3- Export Classic pipelines to YAML (5min).....	16
Module 1 – Export a Build classic pipeline to YML.....	16
Exercise 4- Create YAML Pipelines with Build definition (30min).....	18
Module 1 – Create a YAML pipeline	18
Module 2 – Update YAML definition in ADO YAML pipeline editor.....	19
Module 3 – Understand Build definition in YAML	21
Module 4 – Run YAML pipeline.....	22
Summary.....	22
Exercise 5- Add Release definition in YAML (15min)	23
Module 1 – Update YAML pipeline for Release definition.....	23
Module 2 – Understand YAML definition for Release process	25
Summary.....	25
Exercise 6- Manage YAML scripts (1h)	27
Module 1 – Use Variables	27
Module 2 – Create YAML templates	29
Module 3 – Create another YAML pipeline from your own YAML files	32
Module 4 – Configure permissions for your YAML pipeline	34
Module 5 – Run pipeline to deploy your application	34
Summary.....	35
Exercise 7 (Optional) – Azure Artifacts (30min).....	36
Module 1 – Set up Upstream feed in ADO Artifacts	36
Module 2 – Link your Artifacts feed for CI process.....	37
Module 3 – Check CICD pipelines	38
Module 4 – Let's make changes.....	38

Lab 2 - Introduction to Azure DevOps Pipelines

Objective(s)	<ul style="list-style-type: none"> To be able to create ADO pipelines using the Classic mode To be able to troubleshoot pipeline errors and fix errors To be able to deploy Python application to Azure App Service in Azure through your pipeline processes To be able to create YAML pipelines for your repo To be able to create your own YAML files for pipelines To be able to create YAML templates To be able to configure the deployment method to force Web App Deploy method in Azure deploy task for your pipelines
Duration of Lab	<ul style="list-style-type: none"> 3h
Prerequisite(s)	<ul style="list-style-type: none"> Azure Subscription and Resource Group Contributor Role in a selected Resource Group in Azure Git for Windows installed locally (Git for Windows) Visual Studio Code editor Azure DevOps (ADO) organisation and project Contributor role in the selected ADO project Azure App Service for Python App running on Linux in Azure
Tool(s)	<ul style="list-style-type: none"> Azure Portal Azure DevOps (ADO) Visual Studio Code
Exercises	<ol style="list-style-type: none"> Create Classic pipelines for your Python application (40min) Deploy Python Application to Azure using Classic mode (15min) Export Classic pipelines to YAML (5min) Create YAML Pipelines with Build definition (30min) Add Release definition in YAML (15min) Manage YAML scripts (60min) Optional Azure Artifacts (30min)
Subscription	[Selected Subscription]
Resource Group	[Selected RG]
Navigation	Throughout this Lab, we will open and use several Browser tabs for easy access. Until the end of the Lab, keep your Browser tabs open.

Naming Convention for Labs

For completing various labs during the workshop, we will use this naming convention. It is slightly different from Microsoft online guidance ([Define your naming convention - Cloud Adoption Framework | Microsoft Learn](#)).

The naming convention below is designed to group your Azure resources together for easy access.

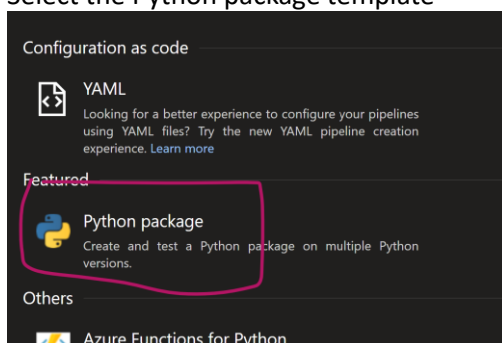
[you name/initials]-[acronym/short name for Azure resource]-[service description]

Exercise 1- Create Classic Pipelines for your Python application (40min)

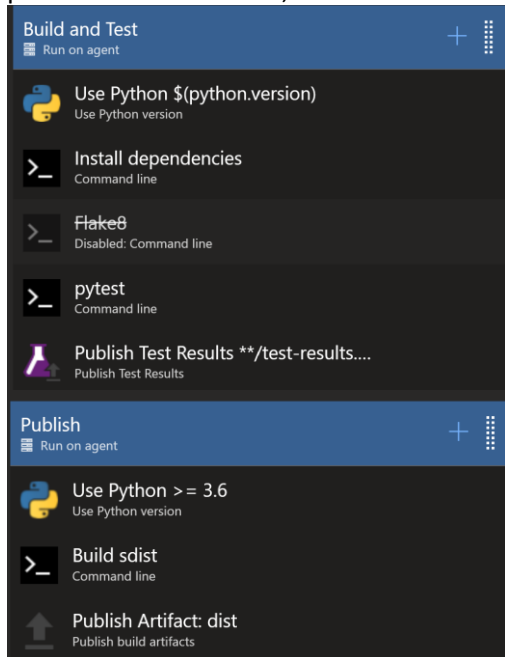
Prerequisite(s)	<ul style="list-style-type: none"> Azure DevOps (ADO) organisation ADO project with repo for Python application ADO service connection accessing to a selected <i>Azure Resource Group (RG)</i> with the Contributor role You are added to the Contributor role at the selected <i>RG</i> Visual Studio Code editor Local copy of your Python application, synced from your ADO repo Azure App Service for Python App running on Linux in Azure
Topics	<p>In this exercise, we will cover the following topics.</p> <ul style="list-style-type: none"> Create a Build pipeline for Python Application Troubleshoot Build pipeline errors Fix Build pipeline errors Create Python Package and Publish artefacts
Duration	<ul style="list-style-type: none"> 40min
Tool(s)	<ul style="list-style-type: none"> Azure Portal Azure DevOps (ADO) Visual Studio (VS) Code
Lab Scenario	<p>In the previous Lab 1, we have learnt the basics of ADO Pipeline features, and pipeline processes.</p> <p>Today, we are going to deploy the Python application to Azure App Service using ADO Pipelines for CI/CD.</p> <p>We will start with the classic pipelines to deploy. So, you will be able to visually see what's happening with your pipelines.</p>
Subscription	[selected subscription]
Resource Group	[selected RG]

Module 1 – Create a Build pipeline for Python Application

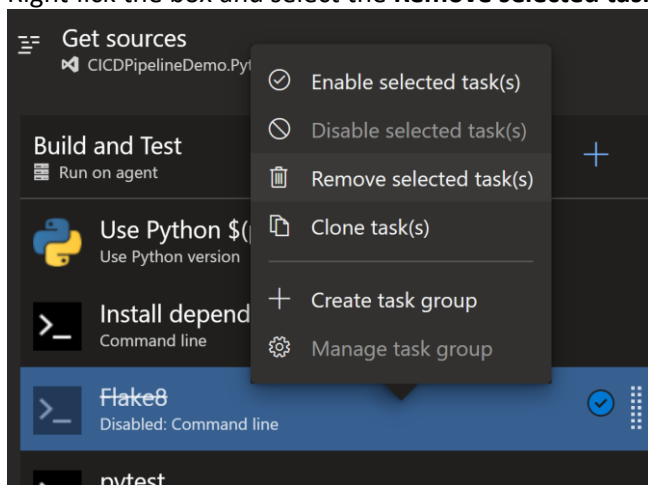
- In ADO, create a new Build pipeline in the **Classic mode** (refer to Lab1-Exercise1)
- After selecting your Python application repo, type **"python"** in the search box. We will use the pre-configured Build template
- Select the Python package template



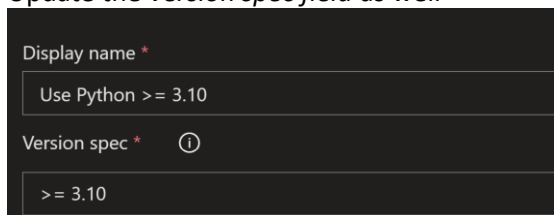
- This templates builds (compiles) your Python application, runs test, creates Python package for deployment and publish the artifacts to the shared fold. There are 2 Jobs within the pipeline process: **Build and Test**, and Publish



- Select the **Pipeline** box on the left
- Specify the *Agent Specification* field as **ubuntu latest**
- Change the *name* of your pipeline appropriately, e.g. **[your shortname]-Python-Web-CI**
- In the *Build and Test Job*, remove the **Flake8** task since we don't use Flake
- Right-click the box and select the **Remove selected task(s)**



- In the *Publish Job*, select the **Use Python >= 3.6** task box
- Update the *Display name* filed as **User Python >= 3.10** (Note: our sample app uses Python 3.12)
- Update the *Version spec* field as well



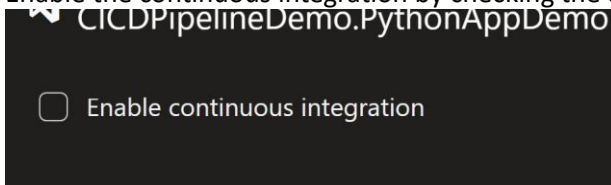
- In the *Publish Job*, remove the **Build sdist** task



In Python, "building" an application often refers to the process of creating a distributable package (like a `.whl` or `.tar` or `.gz` file) that can be installed elsewhere with ***pip***.

However, not all Python applications need to be "built" in this way. If your application is a simple script or a web app that's meant to be run directly from the source code, there might not be a need to create a distributable package. In such cases, **zipping** up the source code and dependencies (as done in your current pipeline) might be all that's needed.

14. We want to set up Continuous Integration (CI), meaning that every time a change is made, the Build process is triggered)
15. Go to the **Triggers** tab at the top
16. Enable the continuous integration by checking the option. Your CI process is now configured

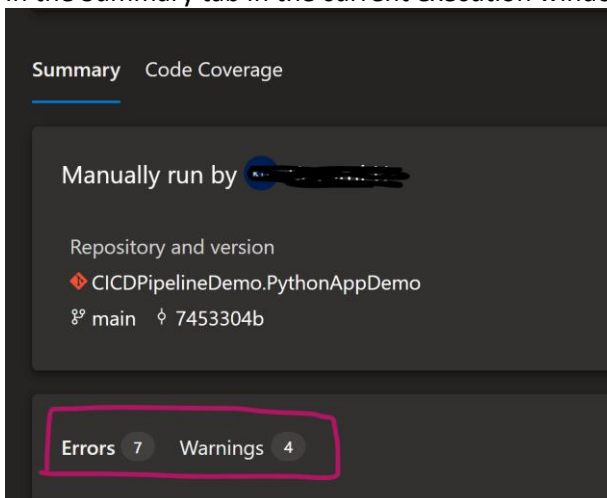


17. Click the **Save & queue** button at the top
18. Enter comments like **"Initial Build pipeline created using the Python template"** and run the process
19. Once the Build pipeline process is completed, you will see several errors in this execution. That is ok. We will troubleshoot it in the next Module

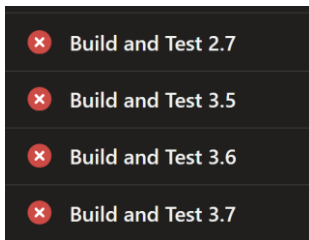
Jobs		
Name	Status	Duration
✖ Build and Test 2.7	Failed	⌚ 16s
✖ Build and Test 3.5	Failed	⌚ 3s
✖ Build and Test 3.6	Failed	⌚ 4s
✖ Build and Test 3.7	Failed	⌚ 14s
ⓘ Publish	Skipped	

Module 2 – Troubleshoot Build pipeline errors

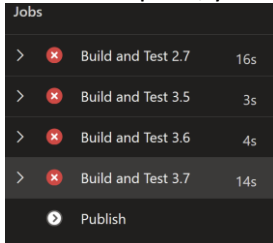
1. In the Summary tab in the current execution window, We have Errors and Warnings



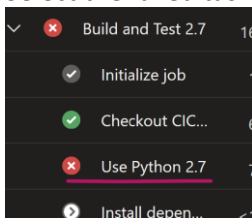
2. Scroll down the page and check what tasks are failed
3. All tasks relating to the *"Build and Test x.x"* failed. But where did those Python versions come from?



4. We have some errors in the Build and Test Job tasks. Let's check it out by selecting one of Errors or failed Job. This will lead to the logs for your pipeline run
5. On the left pane, you can see all the Jobs for the pipeline run



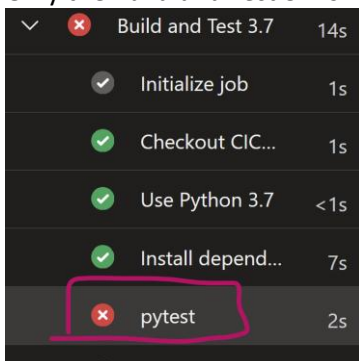
6. Expand the *Build and Test 2.7 Job* and find which task has failed
7. Select the failed task



8. Read the *Warning and Error descriptions*
9. The error means that the task failed to find a Python version that matches the specification "2.7". This could be due to a few reasons:

```
##[warning]You should provide GitHub token if you want to download a python release. Otherwise
##[error]Failed to download Python from the Github Actions python registry (https://github.com
##[error]Version spec 2.7 for architecture x64 did not match any version in Agent.ToolsDirector
```

- a. **Python 2.7 is not available:** Python 2.7 has reached its end-of-life and is no longer maintained, which might be why it's not available in the GitHub Actions python registry²³.
 - b. **Network issues:** There could be network issues preventing the task from accessing the GitHub Actions python registry²³.
 - c. **GitHub token issues:** If the githubToken input is used and is not valid, it could cause download failures¹.
10. In this example, the error means that Python 2.7 is no longer maintained, and it's recommended upgrade to Python 3.x if possible
 11. Check the error(s) for the rest of the *Jobs*
 12. Did you notice differences in errors?
 13. Only the *Build and Test 3.7* shows the different error than the rest of the *Jobs*



14. The error is not related to the Python version, but *pytest task*. The specific Job failed because of Python test task
15. Let's check what the error is about

16. Scroll down to find the error

```

===== no tests ran in
ERROR: file or directory not found: tests
##[error]Bash exited with code '4'.
Finishing: pytest

```

17. `pytest` uses the exit code to indicate the status of the test session

- 0**: All tests were collected and passed successfully
- 1**: Tests were collected and run but some of the tests failed
- 2**: Test execution was interrupted by the user
- 3**: Internal error happened while executing tests
- 4**: `pytest` command line usage error
- 5**: No tests were collected

18. In this example, we have not defined and created any tests. Therefore, the task cannot find the expected file or directory, called **“tests”**, for your testing(**“ERROR: file or directory not found: tests”**)

Module 3 – Fix Build pipeline errors

- Now, we understand what errors are about. Let's fix the errors
- Go back to the definition of your Build pipeline in ADO by selecting your pipeline on the *breadcrumbs* at the top
- Click the **Edit** button
- Let's look at the *Use Python version task* in the *Build and Test Job*
- We have the errors on Python version. The problem is this:

Display name *

Use Python \$(python.version)

Version spec * ⓘ

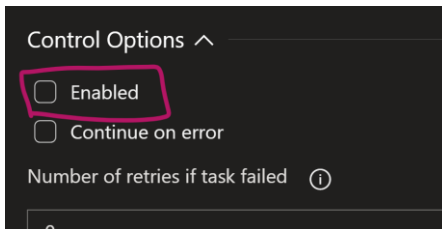
\$(python.version)

- What is **\$(python.version)** syntax here?
- That's the variable expression. So, let's check the *Pipeline variables*
- Select the **Variables** tab at the top
- Find the variable name **“python.version”**. What is the value specified here?
- The versions specified in this variable are wrong

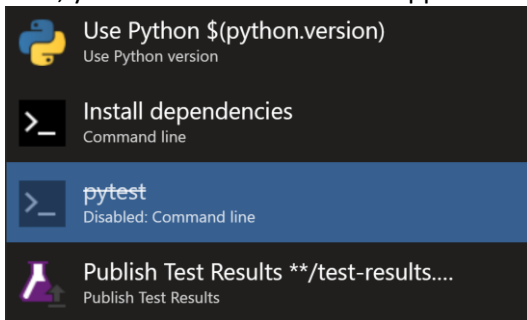
Name ↑	Value
python.version	<u>2.7, 3.5, 3.6, 3.7</u>
system.collectionid	0e3662d8-e6e0-4f556-0076-e57ed4b2

- You can specify the fixed Python version(s) or use a wildcard for the minor version as well
 - 3.10
 - 3.10,3.11,3.12 or
 - 3.x
- We don't want to run build validation and steps for all 3.x versions. So, let's accept the option #1 above
- Update the variable value to **3.10,3.11,3.12**
- You can save (without queueing a new process) here, if you like. So, you don't lose your changes
- Go back to the **Tasks** tab
- The second error was relating to the *pytest task*
- Still in the Editor window for your CI pipeline, select the *pytest task* in the *Build and Test Job*
- We don't have unit tests written yet. So, this task fails because the process cannot find the relevant test folder

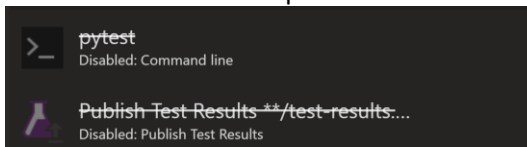
19. You can remove this task, or keep it for future use. Either option is fine
20. To keep the task but not to include in the pipeline execution, disable the task
21. In the editor pane on the right, scroll down to find the **Control Options** section
22. Untick the **Enabled** checkbox. This will disable the selected task when the Job is run



23. Now, you can see the task name appears with the strikethrough



24. Since we don't need to publish the test results, let's disable this task as well (below *pytest* task)



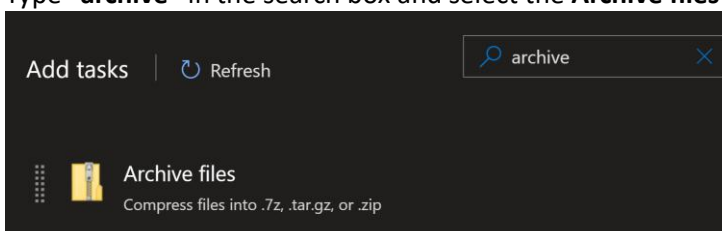
25. **Save and queue** a new process with comments like **"Errors fixed for python versions and pytest disabled"**
26. Did the pipeline run successful?
27. Unfortunately, not
28. Find the cause of the error for your Build pipeline, like you did previously
29. The error indicates that this directory, `/home/vsts/work/1/s/dist`, does not exist or is empty at the time the "Publish Build Artifacts" task is running.

```
=====
##[error]Publishing build artifacts failed with an error: Not found PathtoPublish: /home/vsts/work/1/s/dist
Finishing: Publish Artifact: dist
```

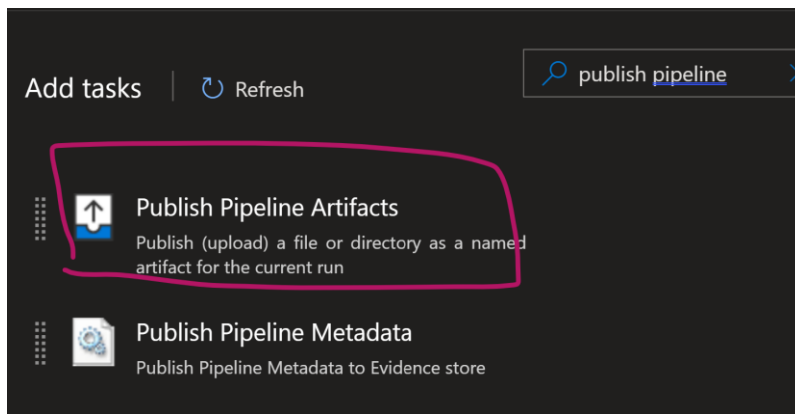
The pre-configure template does not create a Python package (.zip). Thus, we need to create a package and upload it to the shared directory where the Release pipeline process has the access

Module 4 – Create Python Package and Publish (upload) artefacts

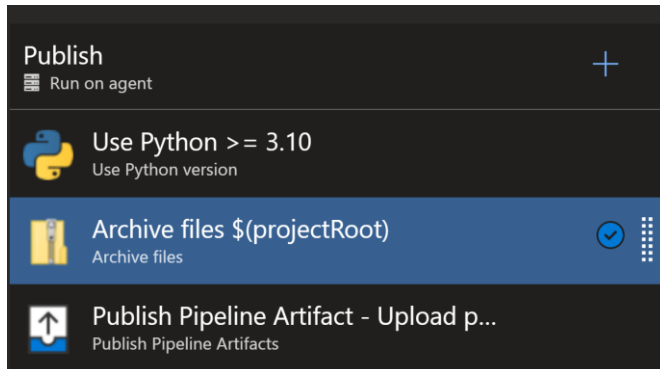
1. In your Build (CI) pipeline editor, remove the **Publish Artifact** task under the *Publish Job*
2. Add a new task by clicking the + button for the *Publish Job*
3. Type **"archive"** in the search box and select the **Archive files** task



4. In the Template pane, enter **"publish pipeline"** in the search box and select the **Publish Pipeline Artifacts** task



5. You have 2 new tasks in the Publish Job



6. Select the **Archive files** task and enter the following details

Area	Value
Task version	2.*
Display name	Archive files \$(System.DefaultWorkingDirectory)
Root folder or file to archive	\$(System.DefaultWorkingDirectory)
Prepend root folder name...	Uncheck
Archive type	Zip
Archive file to create	\$(Build.ArtifactStagingDirectory)/\$(Build.BuildId).zip
Replace existing archive	Checked

7. Select the **Publish Pipeline Artifact** task and enter the following details

Area	Value
Task version	1.*
Display name	Publish Pipeline Artifact - Upload package
Artifact name	drop
Artifact publish location	Azure Pipelines

8. **Save & queue**, and enter comments like **“Added the new tasks for creating Python package and uploading artifacts”**
9. Is your run successful?

10. That's the end of the exercise!

Summary



ACHIEVEMENTS

After you have completed the Lab, you are now able to:

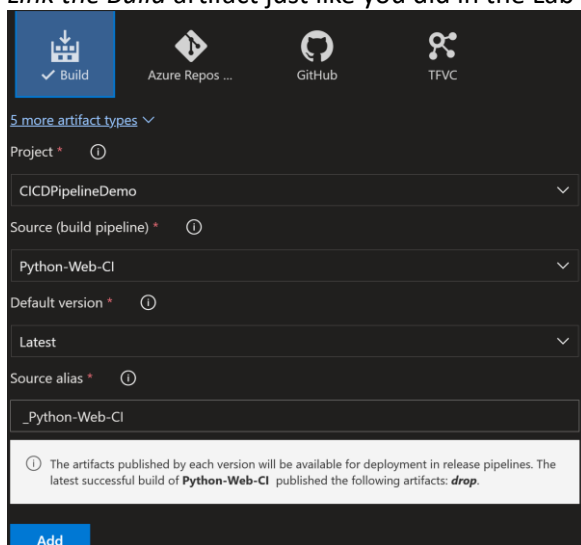
- ✓ Create a Build classic pipeline for Python application
- ✓ View the failed pipeline run
- ✓ Find the error causes
- ✓ Troubleshoot pipeline errors and fix them
- ✓ Create a package (.zip) for Python application and publish (upload) the artifacts

Exercise 2- Deploy Python Application to Azure using Classic Mode (15min)

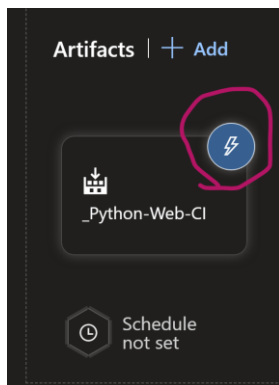
Prerequisite(s)	<ul style="list-style-type: none"> Azure DevOps (ADO) organisation ADO project with repo for Python application ADO service connection accessing to a selected <i>Azure Resource Group (RG)</i> with the Contributor role You are added to the Contributor role at the selected <i>RG</i> Visual Studio Code editor Local copy of your Python application, synced from your ADO repo Azure App Service for Python App running on Linux in Azure
Topics	<p>In this exercise, we will cover the following topics.</p> <ul style="list-style-type: none"> Create a Release pipeline for Python application View your Python App in Azure
Duration	<ul style="list-style-type: none"> 15min
Tool(s)	<ul style="list-style-type: none"> Azure Portal Azure DevOps (ADO) Visual Studio (VS) Code
Lab Scenario	<p>You will deploy your Python application using ADO classic pipelines to Azure App service. This is the basic step for you to be able to create, set up and deploy CICD pipelines for Python applications</p>
Subscription	[selected subscription]
Resource Group	[selected RG]

Module 1 – Create a Release pipeline for Python Application

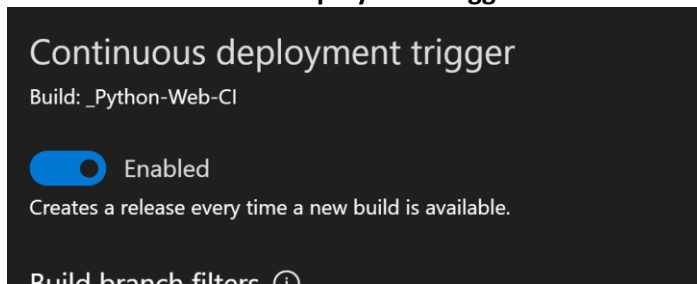
- In **ADO**, select the **Releases** feature under the *Pipelines menu* on the left
- Create a new Release Pipeline by clicking **+New** button on the top left
- Start the **empty job**
- Update the *Stage name* field to “**Dev**” from “*Stage 1*” and close the *Stage editor pane*
- Link the *Build* artifact just like you did in the Lab 1 previously



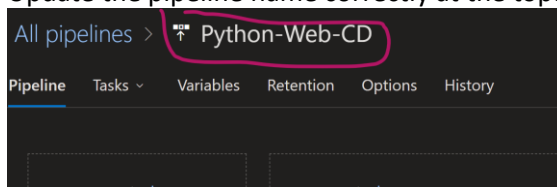
- We want to create Continuous deployment (CD) process for this pipeline, meaning code change are automatically deployed to a specific environment every time CI process is completed
- Click the lightning symbol in the Artifacts



18. Enable the **Continuous deployment trigger** and close the editor pane



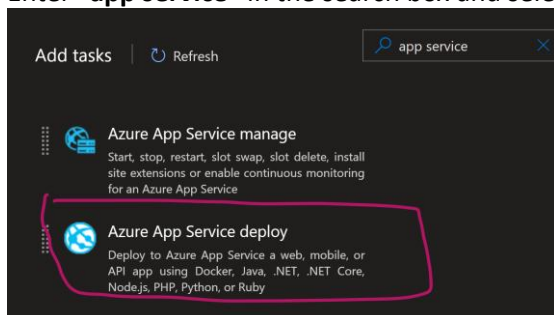
19. Update the pipeline name correctly at the top: **Python-Web-CD**



20. If you don't want to lose your changes, save now. Otherwise, continue
 21. Select the *Dev stage* and we will configure the *Job* and required *Task(s)*
 22. Select the **Agent Job** box on the right
 23. Enter the following details

Area	Value
Display name	Agent job
Agent pool	Azure Pipelines
Agent Specification	Ubuntu latest

24. Add a new task by clicking the + button in the *Agent Job*
 25. Enter "**app service**" in the search box and select **Azure App Service deploy** task

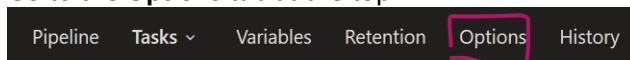


26. Enter the following details

Area	Value
Task version	4.*
Display name	(leave it as it is for now)

Area	Value
Connection type	Azure Resource Manager
Azure subscription	[select your Service Connection for Azure]
App service type	Web app on Linux
App Service name	[select your App Service in Azure that you created in the Lab 1]
Deploy to Slot...	Unchecked
Package or folder	\$(System.DefaultWorkingDirectory)/**/*.zip (leave it as it is)

27. Go to the **Options** tab at the top



28. Change the *Release name* format to `$(date:yyyyMMdd)-Python-Web-Release-$(rev:r)`

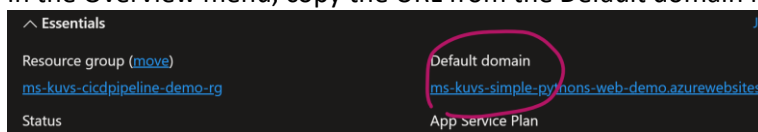
29. **Save** the changes with comments like **"Added Azure web app service deployment task"**

30. Create a *new Release* process

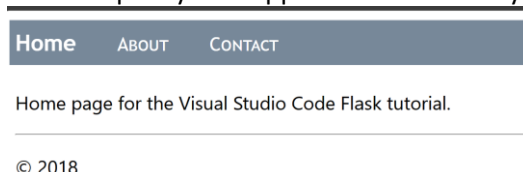
31. Is your Release pipeline run successful?

Module 2 – View your Python App in Azure

1. Go to **Azure portal** (in another browser tab): *portal.azure.com*
2. Select your App Service
3. In the Overview menu, copy the URL from the Default domain field on the right



4. Open another browser tab and paste the URL on the address bar at the top
5. Your sample Python app has been correctly deployed.



6. That's the end of the exercise!

Summary



ACHIEVEMENTS

After you have completed the Lab, you are now able to:

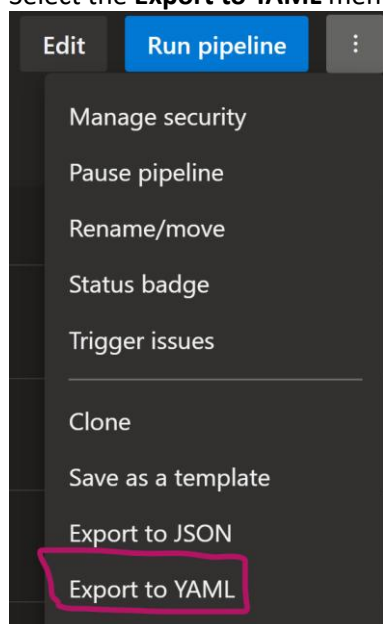
- ✓ Create a Release classic pipeline for Python application
- ✓ Deploy Python application to Azure App Service automatically

Exercise 3- Export Classic pipelines to YAML (5min)

Prerequisite(s)	<ul style="list-style-type: none"> • Azure DevOps (ADO) organisation • ADO project with repo for Python application • ADO service connection accessing to a selected Azure Resource Group (RG) with the Contributor role • You are added to the Contributor role at the selected RG • Visual Studio Code editor • Local copy of your Python application, synced from your ADO repo • Azure App Service for Python App running on Linux in Azure
Topics	<p>In this exercise, we will cover the following topics.</p> <ul style="list-style-type: none"> • Export a Build classic pipeline to YAML
Duration	<ul style="list-style-type: none"> • 5min
Tool(s)	<ul style="list-style-type: none"> • Azure portal • Azure DevOps (ADO) • VS Code
Lab Scenario	<p>You might have created classic pipelines for your deployment before and want to see if you can convert classic pipelines to YAML.</p> <p>You can do so by exporting a classic pipeline to YAML</p>
Subscription	[selected subscription]
Resource Group	[selected RG]

Module 1 – Export a Build classic pipeline to YML

1. In **ADO**, select *you Build (CI) pipeline* for your python app from the *Pipelines menu* on the left
2. Click the **three dots** button at the top right corner
3. Select the **Export to YAML** menu



4. The YAML file is downloaded to your local location
5. Go to the download location and open the file in Notepad
6. That's the YAML steps for your classic pipeline



ADO provides functionality to convert your existing classic pipelines to YAML. But it has some limitations. This functionality works for simple classic pipelines. If pipelines are complicated or customised, or using legacy technologies, this functionality might not be supported.



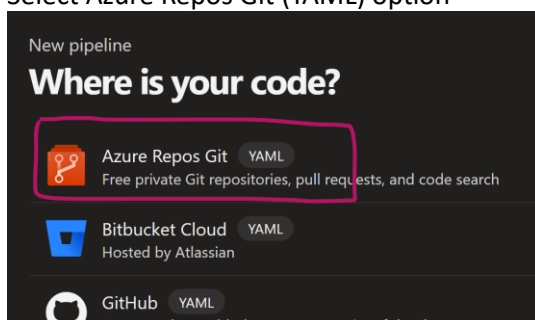
You will not be able to export your Release (CD) pipeline that you created to YAML. It does not look like it's supported from Export functionality

Exercise 4- Create YAML Pipelines with Build definition (30min)

Prerequisite(s)	<ul style="list-style-type: none"> Azure DevOps (ADO) organisation ADO project with repo for Python application ADO service connection accessing to a selected Azure Resource Group (RG) with the Contributor role You are added to the Contributor role at the selected RG Visual Studio Code editor Local copy of your Python application, synced from your ADO repo Azure App Service for Python App running on Linux in Azure
Topics	<p>In this exercise, we will cover the following topics.</p> <ul style="list-style-type: none"> Create YAML pipeline Update YAML definition in ADO YAML pipeline editor Understand Build definition in YAML Run YAML pipeline
Duration	<ul style="list-style-type: none"> 30 min
Tool(s)	<ul style="list-style-type: none"> Azure portal Azure DevOps (ADO) VS Code
Lab Scenario	<p>We have learnt how to create CICD pipelines using the Classic mode. The Classic mode makes it easy for you to create, define and configure your pipeline definition (e.g. processes, steps etc...) as it provides the visual interfaces for you.</p> <p>But a Classic pipeline is not sharable and change history is not tracked.</p> <p>You want to start managing your pipelines in code-first manner, so that pipelines are also managed as part of your code changes, like you do for your application code</p>
Subscription	[selected subscription]
Resource Group	[selected RG]

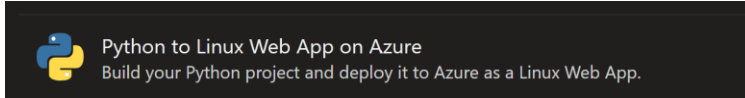
Module 1 – Create a YAML pipeline

1. In ADO, select the **Pipelines** feature in the *Pipelines menu* on the left
2. Click the **New pipeline** button at the top right corner
3. Select Azure Repos Git (YAML) option



4. Select your *Python app repo*

5. A pre-configured YAML pipeline template is available, called *Python Web App on Azure*. It will generate the YAML steps for Build and Release processes for you.

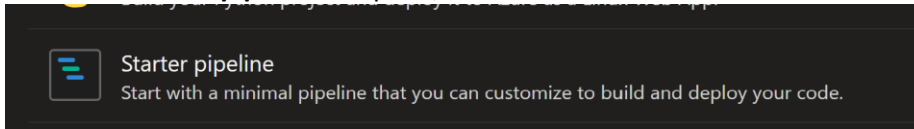


However, this template requires accessing to a selected Azure Subscription and App Service Plan (hosting) directly and tries to create an App Registration in AAD/Entra ID on behalf of you.

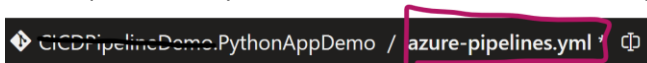
Most of users do not have the right permissions to be able to create App Registrations or configure applications in AAD/Entra ID. If that's the case, you will get an error.

We will not use this template for this Lab

6. Select the **Starter pipeline** template, instead



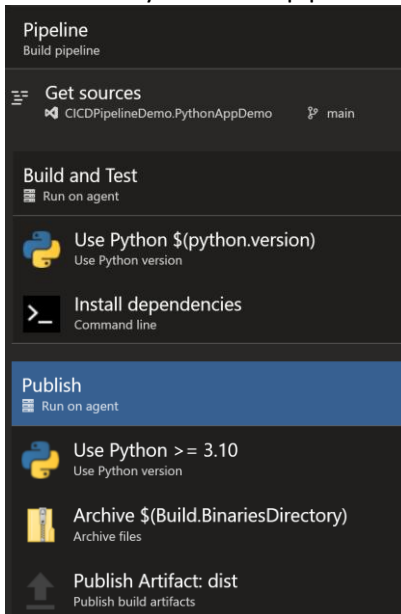
7. You can review the default YAML created the editor window
8. For now, just save the YAML file by clicking the **Save** under the *Save and run button*
9. ADO Pipeline always creates the default YAML file for your selected repo, called **azure-pipelines.yml**



10. Click the **Save** button in the *Save pane*
11. It's **1x YAML to 1x repo relationship** at the moment. We will learn how to create multiple YAML pipelines that are linked to the same repo

Module 2 – Update YAML definition in ADO YAML pipeline editor

1. Click the **Edit** button at the top right corner. This opens the *ADO YAML editor online*
2. Remember your classic pipeline definition for the Build process?



3. We want to follow those processes and steps for the Build stage of your YAML pipeline
4. Delete all the contents in *azure-pipelines.yml*
5. Copy the following scripts

```
# This YAML file is a sample for deploying a simple Python web application to Azure
App Service using Azure DevOps.
# Python application does not use a container and simply gets deployed to Azure App
Service running on Linux OS.
# The pipeline has two stages: Build and Deploy.
# The Build stage installs dependencies, runs tests, and creates a zip file with
the application code.
# The Deploy stage uses AzureRmWebAppDeployment task to deploy the zip file to
Azure App Service, using the Service Connection.

trigger:
- master

stages:
- stage: Build
  displayName: 'Build stage'
  jobs:
  - job: Build
    displayName: 'Build'
    pool:
      vmImage: 'ubuntu-latest'
    steps:
    - task: UsePythonVersion@0
      inputs:
        versionSpec: '3.12'
        addToPath: true

    - script: |
        python -m pip install --upgrade pip
        pip install -r requirements.txt
      displayName: 'Install dependencies'

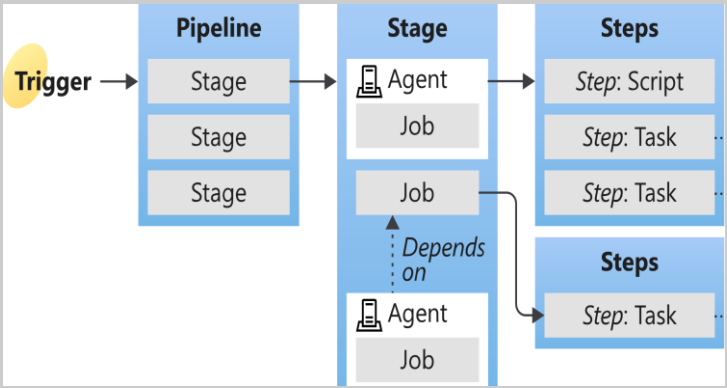
    - task: ArchiveFiles@2
      inputs:
        rootFolderOrFile: '$(System.DefaultWorkingDirectory)'
        includeRootFolder: false
        archiveType: 'zip'
        archiveFile: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
        replaceExistingArchive: true

    - upload: $(Build.ArtifactStagingDirectory)/$(Build.BuildId).zip
      displayName: 'Upload package (zip file)'
      artifact: drop

    - script: |
        echo '##vso[task.setvariable
variable=zipFilePath;]$(Pipeline.Workspace)/drop/$(Build.BuildId).zip'
      displayName: 'Set zip file path'
```

6. Click the **Validate and save** button

Module 3 – Understand Build definition in YAML

YAML	Description
Comments	To add comments in YAML file, use “#” symbol
<pre>trigger: - master</pre>	<p>The trigger section in an Azure Pipelines YAML file specifies which branches will cause the pipeline to run when changes are pushed to them.</p> <p>In your case, trigger: - master means that the pipeline will automatically run whenever changes are pushed to the master branch of your repository.</p>
Pipeline structure	
Stages	<ul style="list-style-type: none"> Build stage (just created) Deploy stage (will create in the later sections)
<pre>displayName: 'Build stage'</pre>	<p>The displayName attribute in an Azure Pipelines YAML file is used to provide a human-readable name for a <i>pipeline</i>, <i>stage</i>, <i>job</i>, or <i>task</i>. This name is displayed in the Azure DevOps web interface.</p> <p>These display names do not affect the functionality of the pipeline, they are purely for making the pipeline easier to understand and manage. You can set them to any string you like.</p>
<pre>pool: vmImage: 'ubuntu-latest'</pre>	<p>We configure the Agent for the selected Job to run on ubuntu-latest OS</p>
<pre>- task: UsePythonVersion@0 inputs: versionSpec: '\${python.version}' addToPath: true</pre>	<p>The UsePythonVersion@0 task in an Azure Pipelines YAML file is used to specify the version of Python that should be used in subsequent tasks in the pipeline.</p> <ul style="list-style-type: none"> It sets up the specified version of Python on the agent where the pipeline is running. If no version is specified, it will use the latest version of Python available on the agent. It adds the Python installation to the PATH so it can be used in subsequent tasks.
<pre>- script: python -m pip install --upgrade p pip install -r requirements.txt -</pre>	<p>The - script: directive in an Azure Pipelines YAML file is used to define a shell script that should be run as a step in the pipeline. This script runs in a new process on the agent where the pipeline is running.</p> <ul style="list-style-type: none"> Single command <ul style="list-style-type: none"> - script: echo Hello, world! Multiline command <ul style="list-style-type: none"> - script

YAML	Description
	<ul style="list-style-type: none"> - echo Add other command here - Echo each command runs in the same shell
<pre>- task: ArchiveFiles@2</pre>	<p>The ArchiveFiles@2 task in an Azure Pipelines YAML file is used to create an archive (zip, tar, etc.) of a set of files. This is often used to package up a build artifact that can be passed to subsequent stages of the pipeline or stored for later use.</p>
<pre>- upload:</pre>	<p>The - upload: directive in an Azure Pipelines YAML file is used to upload build artifacts. These artifacts can be anything produced by your build that you want to keep and use in later stages of the pipeline, or even in other pipelines.</p>
<pre>- script: echo '##vso[task.setvariable variable=zipFile</pre>	<p>The echo '##vso[task.setvariable variable=zipFilePath;]\$(Pipeline.Workspace)/drop/\$(Build.BuildId).zip' command is a special syntax used in Azure Pipelines to set a variable from within a script.</p> <p>After this step, there will be a new pipeline variable named zipFilePath that contains a path to a zip file in the drop directory of the pipeline's workspace. This variable can be used in subsequent steps of the pipeline.</p>

Module 4 – Run YAML pipeline

1. Click the **Run Pipeline** button to trigger the YAML pipeline process and check if it is successful

Summary



ACHIEVEMENTS

After you have completed the Lab, you are now able to:

- ✓ Create YAML pipeline for your repo
- ✓ Add the Build definition, processes and steps
- ✓ Run new pipeline process

Exercise 5- Add Release definition in YAML (15min)

Prerequisite(s)	<ul style="list-style-type: none"> Azure DevOps (ADO) organisation ADO project with repo for Python application ADO service connection accessing to a selected Azure Resource Group (RG) with the Contributor role You are added to the Contributor role at the selected RG Visual Studio Code editor Local copy of your Python application, synced from your ADO repo Azure App Service for Python App running on Linux in Azure
Topics	<p>In this exercise, we will cover the following topics.</p> <ul style="list-style-type: none"> Update YAML pipeline for Release definition Understand YAML definition for Release process
Duration	<ul style="list-style-type: none"> 15min
Tool(s)	<ul style="list-style-type: none"> Azure portal Azure DevOps (ADO) VS Code
Lab Scenario	<p>We have created YAML pipeline and added the Build definition in the previous exercise.</p> <p>We will add the Release definition in YAML pipeline, so you will be able to deploy your Python application to Azure</p>
Subscription	[selected subscription]
Resource Group	[selected RG]

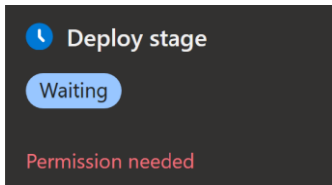
Module 1 – Update YAML pipeline for Release definition

- In your YAML pipeline (azure-pipelines.yml) in ADO, click the **Edit** button at the top right corner
- Scroll down and add the following definition at the bottom in the online editor window

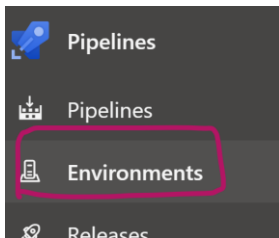
```
- stage: Deploy
  displayName: 'Deploy stage'
  dependsOn: Build
  jobs:
  - deployment: Deploy
    displayName: 'Deploy Python App to Azure App Service'
    environment: 'Dev'
    pool:
      vmImage: 'ubuntu-latest'
    strategy:
      runOnce:
        deploy:
          steps:
            - task: AzureRmWebAppDeployment@4
              inputs:
                ConnectionType: 'AzureRM'
                azureSubscription: '[Your-Service-Connection-Name]'
                appType: 'webAppLinux'
                WebAppName: '[your-App-Service-name]'
                packageForLinux: '$(Pipeline.Workspace)/drop/$(Build.BuildId).zip'
```

where **[Your-Service-Connection-Name]** is replaced with the actual name of your **ADO Service Connection**, and **[your-App-Service-name]** is replaced with the **Azure App Service name**

3. Kick off the updated YAML pipeline by clicking the **Run** button
4. At this stage, your Deploy Stage is pending upon Permission. Or you might get an error indicating that the environment does not exist in ADO before you can use it in a pipeline



5. Select the **Environments** feature under the *Pipelines menu* on the right (in another browser tab)



6. Create a **New environment** called **"Dev" without Resource**

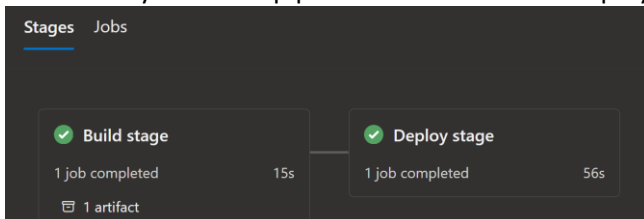


In Azure Pipelines, Environments can have specific security and access policies. By default, when a pipeline attempts to deploy to an environment for the first time, it needs to be approved by someone with manage permissions on the environment.

To resolve this issue, you or someone else with the appropriate permissions needs to approve the deployment.

If you want to avoid this manual approval step in the future, you can change the security and access policies for the environment to automatically approve deployments from this pipeline. **Be aware that this can potentially increase the risk of unauthorized deployments, so it should be done with caution.**

7. In *Environment definition* window for the Dev, click the **three dots** at the top right corner and select the **Security**
8. Scroll down to the **Pipeline permissions** section
9. Click the **+** button to add YAML pipeline to allow accessing this resource
10. Select your YAML pipeline (azure-pipelines.yml in your repo)
11. As soon as you have allowed your YAML pipeline to use this resource *"Dev" environment*, the pending process for your YAML pipeline run will kick off and deploy your Python application
12. Go back to your YAML pipeline and check if the deployment is successful



Module 2 – Understand YAML definition for Release process

YAML	Description
<code>dependsOn: Build</code>	<p>The dependsOn: directive in an Azure Pipelines YAML file is used to specify that one job or stage should not start until another job or stage has completed. This is useful when you have a job or stage that relies on the output of another job or stage.</p> <p>In your case, dependsOn: Build means that the current job or stage will not start until the Build job or stage has completed. If the Build job or stage fails, the current job or stage will not run.</p> <p>By default, a stage will only run if all the stages in <i>depends on</i> have completed successfully. You don't need to explicitly define a condition that the Deploy state runs only if the Build stage is successful here</p>
<code>jobs: - deployment: Deploy</code>	<p>The deployment job is a special type of Jobs.</p> <p>The name following - deployment: (in this case, Deploy) is just an identifier for the job and can be any valid string.</p> <p>For the Build stage, we defined Jobs as:</p> <p><i>Jobs:</i></p> <p>- job: Build</p> <p>For the deployment job, we define it as:</p> <p><i>Jobs:</i></p> <p>- deployment: Deploy</p>
<code>environment: 'Dev'</code>	<p>Required in the deployment job.</p> <p>The environment: directive in an Azure Pipelines YAML file is used to specify an environment that the pipeline will target.</p> <p>Environments in Azure Pipelines represent a collection of resources that can be targeted by deployments from a pipeline.</p>
<code>strategy: runOnce: deploy:</code>	<p>The strategy: directive in an Azure Pipelines YAML file is used to specify the strategy that should be used to run the jobs in a stage. There are several strategies available, including runOnce, rolling, and canary.</p> <p>In your case, runOnce is being used. This is the simplest strategy and is the default if no strategy is specified. It means that all the jobs in the stage will be run once.</p> <p>In this example, the Deploy job will be run once.</p>

Summary



ACHIEVEMENTS

After you have completed the Lab, you are now able to:

- ✓ Update the existing YAML file with additional Stage, Jobs, and Steps for Release process
- ✓ Add a new Environment for YAML pipelines to use

- ✓ Allow automatic approval for a specific environment, so selected YAML pipeline(s) can access this resource
- ✓ Understand the Release process

Exercise 6- Manage YAML scripts (1h)

Prerequisite(s)	<ul style="list-style-type: none"> Azure DevOps (ADO) organisation ADO project with repo for Python application ADO service connection accessing to a selected Azure Resource Group (RG) with the Contributor role You are added to the Contributor role at the selected RG Visual Studio Code editor Local copy of your Python application, synced from your ADO repo Azure App Service for Python App running on Linux in Azure
Topics	<p>In this exercise, we will cover the following topics.</p> <ul style="list-style-type: none"> Use variables Create YAML templates Create another YAML pipeline from your own YAML files Configure permissions for your YAML pipeline Run pipeline to deploy your application to Azure
Duration	<ul style="list-style-type: none"> 60min
Tool(s)	<ul style="list-style-type: none"> Azure portal Azure DevOps (ADO) VS Code
Lab Scenario	<p>Having a single YAML file is easy.</p> <p>But what if you want to create different YAML pipelines for repo?</p> <p>What if your YAML file gets long and complicated?</p> <p>You want to make sure that your YAML files are manageable and maintainable</p>
Subscription	[selected subscription]
Resource Group	[selected RG]

Module 1 – Use Variables

- Your YAML script works fine. But it's quite long and a lot of configuration values are hardcoded. That's not a good coding practice.
Those configuration values should not be hardcoded.
In addition, hardcoded values are repeated in the script or in various places
- You can create the **Pipeline variables** using the *Variable button* (similarly to what you did previously in the Classic pipelines), or define local variables within YAML file
- We will define variables within YAML file
- For this Lab, we also want to include the existing Variable Group that is created in the Lab 1
- When running YAML pipelines, by default, YAML pipelines are not allowed to access some resources.
i.e. Environment resource in Environments or Variable Group in the Library feature
- You need to configure required permissions for your YAML pipeline
- In ADO (in another browser tab), open the **Library** feature under the *Pipelines menu* on the left
- Select your *Variable group* you created in the Lab 1
- Select the Pipeline permissions button at the top



- Add your Python app YAML pipeline (*azure-pipelines.yml* in your repo) by clicking the **+** button, and close the pop-up window

11. Go back to the YAML online editor for your pipeline in ADO, add the following scripts to define variables, after **tigger:** and before **stages:** directives

```
variables:
- group: '$(variableGroupForKv)'
- name: python.version
  value: '3.12'
- name: webAppName
  value: '[your-app-service-name]'
- name: serviceConnectionName
  value: '[your-service-connection-name]'
- name: vmImageName
  value: 'ubuntu-latest'
```

12. It looks like this

```
trigger:
- master

# Note:
# 1) versionSpec in UsePythonVersion@0 task
# 2) To access a specific Variable Group, yo
# 3) appType in AzureRmWebAppDeployment@4 do
variables:
- group: '$(variableGroupForKv)'
- name: python.version
  value: '3.12'
- name: webAppName
  value: '[your-app-service-name]'
- name: serviceConnectionName
  value: '[your-service-connection-name]'
- name: vmImageName
  value: 'ubuntu-latest'

stages:
- stage: Build
  displayName: 'Build stage'
  jobs:
```



In Azure Pipelines, you can access variables defined in a Variable Group directly by their names: ***`$(your variable name defined in Variable Group)`***

13. Let's update the YAML script to use variables in the online editor

```
- stage: Build
  displayName: 'Build stage'
  jobs:
  - job: Build
    displayName: 'Build'
    pool:
      vmImage: '$(vmImageName)'
```

```
- task: UsePythonVersion@0
  inputs:
    versionSpec: '$(python.version)'
    addToPath: true
```

```
jobs:
- deployment: Deploy
  displayName: 'Deploy Python App to Azure App S
  environment: 'Dev'
  pool:
    vmImage: '$(vmImageName)'
```

14. Validate and save

Note: if validating pipeline is taking too long, simply save without validating

15. Run the pipeline

Module 2 – Create YAML templates

1. YAML file that you have created (*azure-pipelines.yml*) is getting longer with complex logic inside. This makes it difficult to manage and maintain your script. We want to split each Stage into separate files and only reference the relevant files from the main YAML pipeline process. How do we do that?

We can create separate YAML template files for the Build and Release processes separately and reference templates from the main YAML pipeline script.

2. Open **VS Code** with your Python application folder
3. At root, add a new YAML file
 - a. **python-app-templates-main.yml**
4. Create a new folder at the root, called **yaml**
5. Create 2x files in this folder
 - a. **python-app-templates-build.yml**
 - b. **python-app-templates-release.yml**



QUICK TIPS You can create YAML templates at subfolder level. However, your *pipeline* (main) YAML file always needs to be the **root** of your repo.

6. From your ADO YAML pipeline (*azure-pipelines.yml*) that you created previously using ADO YAML pipeline editor, copy YAML script and paste it to both **python-app-template-build.yml** and **python-app-template-release.yml** files in VS Code
7. In *python-app-templates-build.yml* file, only keep the block of codes from **jobs:** directive for the *Build stage*, and remove everything else.

This file only contains the definition of the *Build job*

```
jobs:
- job: Build
  pool:
    vmImage: '${{ parameters.vmImageName }}'

steps:
- task: UsePythonVersion@0
```

8. In *python-app-template-releases.yml* file, only keep the block of codes **jobs:** directive for the *Deploy stage*, and remove everything else.

This file only contains the definition of the *Deploy (release) job*

```
jobs:
- deployment: Deploy
  displayName: 'Deploy Python App to Azure'
  environment: '${{ parameters.environmentName }}'
  pool:
    vmImage: '${{ parameters.vmImageName }}'

strategy:
```

9. In **python-app-templates-main.yml** file, copy and paste the following YAML definition

```
# This YAML file is used to define the pipeline stages and the parameters that are
# used in the pipeline.
# The pipeline stages are defined using the build.yml and deploy.yml templates.
# The parameters are defined using the variables section.
# The parameters are used to define the Python version, the Azure App Service name,
# and the Azure service connection name.
trigger:
- master

variables:
- name: python.version
  value: '3.12'
- name: webAppName
  value: '[your-app-service-name]'
- name: serviceConnectionName
  value: '[your-ado-service-connection-name]'

stages:
- stage: Build
  displayName: 'Build stage'
  jobs:
  - template: yaml/python-app-templates-build.yml
    parameters:
      vmImageName: 'ubuntu-latest'
      pythonVersion: '$(python.version)'

- stage: Release
  displayName: 'Release stage'
  dependsOn: Build
  jobs:
  - template: yaml/python-app-templates-release.yml
    parameters:
      vmImageName: 'ubuntu-latest'
      webAppName: '$(webAppName)'
      azureSubscription: '$(serviceConnectionName)'
      environmentName: 'Dev'
```

10. From the main YAML, we need to pass values to parameters, so each process can run accordingly

11. In *python-app-templates-build.yml* file, copy and paste the following parameter declaration at the top, before **jobs:** directive

```
parameters:
- name: vmImageName
  type: string
  default: 'ubuntu-latest'
  values:
    - 'windows-latest'
    - 'ubuntu-latest'
    - 'macOS-latest'
- name: pythonVersion
  type: string
  default: '3.10'
```



Use **default:** (optional) and **values:** (optional) directives for setting default value and adding other options for your parameter

12. In *python-app-templates-release.yml* file, copy and paste the following parameter declaration at the top, before **jobs:** directive

```
parameters:
- name: vmImageName
  type: string
  default: 'ubuntu-latest'
  values:
    - 'windows-latest'
    - 'ubuntu-latest'
    - 'macOS-latest'
- name: webAppName
  type: string
- name: azureSubscription
  type: string
- name: environmentName
  type: string
  default: 'Dev'
  values:
    - 'Dev'
    - 'SIT'
    - 'Prod'
```

13. *python-app-templates-main.yml* file references 2x YAML template files: *python-app-templates-release.yml* file and *python-app-templates-release.yml* file respectively. The main YAML script passes required parameter values to each YAML process.

This way, you don't need to repeat or duplicate the same codes within template files. This makes your YAML files reusable, readable and maintainable with clear separation of concerns.

14. Update both Build and Release YAML template scripts to using parameters



Parameter expression: `${{ parameters.[parameter name] }}` where you replace [parameter name] with the actual parameter name

15. `python-app-templates-build.yml` file looks like this with parameters:

```
pool:
  vmImage: '${{ parameters.vmImageName }}'

- task: UsePythonVersion@0
  inputs:
    versionSpec: '${{ parameters.pythonVersion }}'
    addToPath: true
```

16. `python-app-templates-release.yml` file looks like this with parameters:

```
pool:
  vmImage: '${{ parameters.vmImageName }}'

- task: AzureRmWebAppDeployment@4
  inputs:
    ConnectionType: 'AzureRM'
    azureSubscription: '${{ parameters.azureSubscription }}'
    appType: 'webAppLinux'
    WebAppName: '${{ parameters.webAppName }}'
    packageForLinux: '$(Pipeline.Workspace)/**/*.zip'
```

17. In VS Code, check in your changes to ADO repo

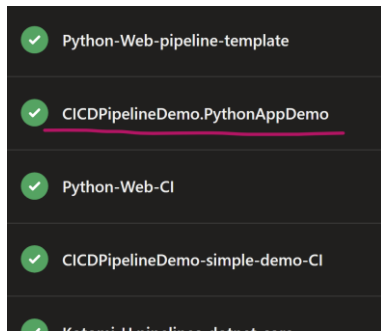
Module 3 – Create another YAML pipeline from your own YAML files

1. In **ADO**, select **Pipelines** features under the *Pipelines menu* on the left
2. We have the default YAML pipeline linked to your repo: `azure-pipelines.yml`. This default pipeline gets triggered every time a change is made in your “main (or master)” branch. Your new pipeline will also be linked to the same repo with the same default branch (master).

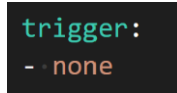
We do not want to have multiple pipelines for your repo running simultaneously or concurrently. We need to disable the default `azure-pipelines.yml` here

3. In the *Pipelines feature*, select your default YAML pipeline (usually the name of your repo) in the list and click the **Edit** button.

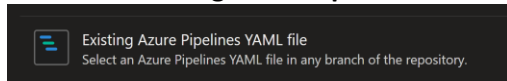
e.g. My repo name is “CICDPipelineDemo.PythonAppDemo” and my YAML pipeline reflects that below.



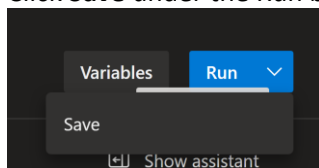
4. Change **trigger:** to **none**. This will change the pipeline to be triggered manually



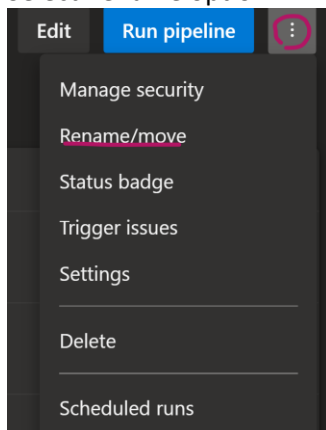
5. Go back to the *Pipelines window*
6. Click the **New pipeline** button at the top right corner to create a new pipeline process
7. Select **Azure Repos Git (YAML)** option
8. Select your *python application repo*
9. Select the **Existing Azure Pipelines YAML file** option



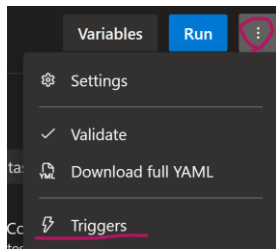
10. Select your YAML file in your repo for the **Path** field from the dropdown list
11. Click **Continue**
12. Click **Save** under the Run button



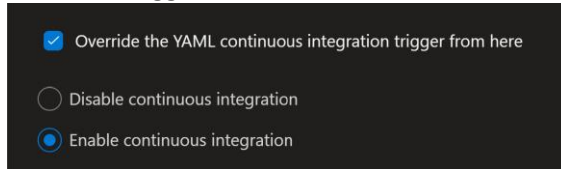
13. Select newly created your YAML pipeline in the Pipeline summary list
14. Click the **three dots** button at the top right corner
15. Select **Rename** option



16. Rename your pipeline like "**[your-short-name]-python-web-pipeline-template**"
17. We want to set up Continuous Integration (CI) process, meaning that changes in your repo automatically triggers the pipeline processes
18. Click **Edit** button and click **three dots** button
19. Select **Triggers** option



20. Enable CI trigger



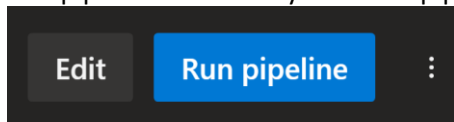
21. If you make changes in your YAML file(s), this allows you to override the existing YAML definition and allows you to set up CI proces

Module 4 – Configure permissions for your YAML pipeline

1. Your newly created YAML pipeline is not allowed to access the Environment resource(s) to be able to execute. This was briefly covered in the previous module about configuring permissions for YAML pipelines
2. Select **Environments** feature under the *Pipelines menu* (in another browser tab)
3. Select **Dev** environment
4. Click the **three dots** button at the top right corner
5. Select **Security** option
6. Scroll down to the Pipeline permission section
7. Click + button and select your newly created pipeline here

Module 5 – Run pipeline to deploy your application

1. Go back to your Pipeline window (in another tab)
2. Run pipeline to kick off your YAML pipeline process for deployment



3. Once the pipeline process is completed successfully, go to Azure portal (in another browser tab)
4. Find your Web Service for Python app
5. Open the Python App by clicking the **URL** in the *Overview*
6. Can you see your Python app?



AzureRmWebAppDeployment@4 task has some gotcha.

Microsoft team created an auto-detect feature that tries to guess the best method for code deployment to your Web App: *Web Deploy*, *Kudu REST APIs*, *Container Registry*, *Zip Deploy*, *RunFromPackage*.

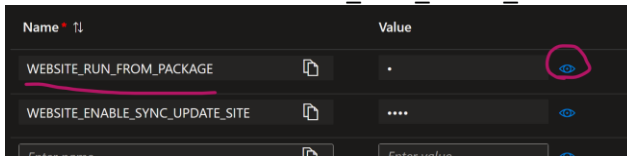
There are some occasions that Python application deployment using *AzureRmWebAppDeployment@4* task automatically selects **RunFromPackage** deployment method, instead of *Web App Deploy*.

The problem of *RunFromPackage* is that the default location for the application files (*/site/wwwroot*) can get mismatched with .zip package files. If/when that happens, you might get an error not to be able to run your application.

You need to force the Web App Deploy method, so Azure task doesn't automatically change the method to *RunFromPackage* method.

Ensure the default environment variable, *WEBSITE_RUN_FROM_PACKAGE*, is set to 0 (instead of 1)

7. In your *App Service* in *Azure portal*, type “**environment**” in the *search box* for menus
8. Select **Environment variables** feature under the *Settings section*
9. Check the value for *WEBSITE_RUN_FROM_PACKAGE*, and make sure the value = 0



10. Go back to **VS Code**
11. Update **AzureRmWebAppDeployment@4** task in *python-app-templates-release.yml* file like below

```
- task: AzureRmWebAppDeployment@4
  inputs:
    ConnectionType: 'AzureRM'
    RenameLockedFiles: true # Rename locked files during deployment
    UseWebDeploy: true # Explicitly set to Web Deploy
    TakeAppOfflineFlag: true # Take app offline during deployment
    azureSubscription: '${ parameters.azureSubscription }'
    appType: 'webAppLinux'
    WebAppName: '${ parameters.webAppName }'
    packageForLinux: '$(Pipeline.Workspace)/**/*.zip'
```

12. Check in changes to trigger CI/CD processes for your YAML pipeline
13. Go back to your YAML pipeline in ADO, and check if your pipeline run is successful

Summary



ACHIEVEMENTS

After you have completed the Lab, you are now able to:

- ✓ User variables in YAML file
- ✓ Create YAML templates for reusability and maintainability
- ✓ Configure pipeline permissions for accessing to the Environment resource in ADO
- ✓ Run YAML pipeline that uses separate YAML templates files for deployment
- ✓ Configure Azure deployment method to using Web App Deployment, rather than *RunFromPackage* method

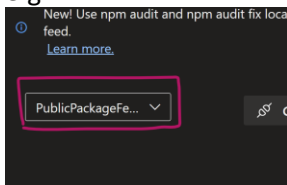
Exercise 7 (Optional) – Azure Artifacts (30min)


Prerequisite(s)	<ul style="list-style-type: none"> Azure DevOps (ADO) organisation ADO project with repo for Python application ADO service connection accessing to a selected Azure Resource Group (RG) with the Contributor role You are added to the Contributor role at the selected RG Visual Studio Code editor Local copy of your Python application, synced from your ADO repo Azure App Service for Python App running on Linux in Azure
Topics	<p>In this exercise, we will cover the following topics.</p> <ul style="list-style-type: none"> Set up Upstream feed in ADO Artifacts Link your Artifacts feed for CI Process Check CICD pipelines
Duration	<ul style="list-style-type: none"> 30min
Tool(s)	<ul style="list-style-type: none"> Azure portal Azure DevOps (ADO) VS Code
Reference(s)	<ul style="list-style-type: none"> Upstream sources overview - Azure Artifacts Microsoft Learn
Lab Scenario	
Subscription	[selected subscription]
Resource Group	[selected RG]

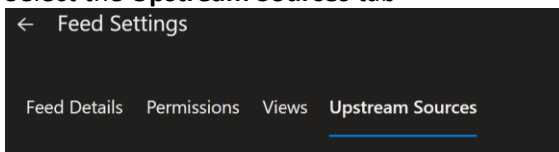
Module 1 – Set up Upstream feed in ADO Artifacts

- Create a new Upstream feed in ADO Artifacts and name it like “[your-short-name]-upstream-feed]”. You will create a feed as “*project-scoped*” feed, instead of “*organisation-scoped*” feed here.
- Follow the instruction from [Set up upstream sources for your feed - Azure Artifacts | Microsoft Learn](#). Complete the first section “Enable upstream sources in a new feed” only
- Check the newly created feed by selecting the dropdown list










e.g.



- Check the **settings** for this feed by clicking the cog button  on the right
- Select the **Upstream Sources** tab



- Initially, publicly available common feeds are included in the upstream feed

Feed Details	Permissions	Views	Upstream Sources
<input type="checkbox"/>		npmjs	https://registry.npmjs.org/
<input type="checkbox"/>		NuGet Gallery	https://api.nuget.org/v3/index.json
<input type="checkbox"/>		PowerShell Gallery	https://www.powershellgallery.com/api/v2
<input type="checkbox"/>		PyPI	https://pypi.org/
<input type="checkbox"/>		Maven Central	https://repo.maven.apache.org/maven2/
<input type="checkbox"/>		Google Maven Reposi...	https://dl.google.com/android/maven2/
<input type="checkbox"/>		JitPack	https://jitpack.io/
<input type="checkbox"/>		Gradle Plugins	https://plugins.gradle.org/m2/
<input type="checkbox"/>		crates.io	https://index.crates.io/

7. We don't need to manage all of feed sources like above
8. Select everything, except for PyPI, above and click the **Delete** button
9. Click **Save**
10. Copy the URL

Module 2 – Link your Artifacts feed for CI process



Private feed in ADO Artifacts requires correct authentication. You can use:

- a. **PAT** (Personal Access Token)
- b. **PIPAuthenticate** task

Example 1: Using PAT Authentication

1. Create a new PAT and give "Read" permission for ADO Artifacts ([Use personal access tokens - Azure DevOps | Microsoft Learn](#))
2. To index ADO Artifact feed, you need to set the correct URL like this:
`https://[PAT]@[Feed URI]`

where **[Feed URI]** is replaced with either:

`pkgs.dev.azure.com/[ADO Org Name]/[Project Name]/_packaging/[Your Feed Name]/pypi/simple/` for **project-scoped feed**, or

`pkgs.dev.azure.com/[ADO Org Name]/_packaging/[Your Feed Name]/pypi/simple/` for **org-scoped feed**

Example 2: Using PIPAuthenticate task in YAML

1. PAT requires a proper secret management. Instead of using PAT, you can also use YAML task to automatically get authenticated
2. Use **PIPAuthenticate** task in your YAML Job
`task: PipAuthenticate@1`
`inputs:`
`artifactFeeds: 'YourProjectName/YourFeedName'`



The **PipAuthenticate** task takes care of the authentication with the ADO feed.

The **`pip install -r requirements.txt`** command will install the packages from your authenticated feed without needing to specify the index URL.

Steps

1. Previously, our pipelines (both Classic and YAML) are upgrading your Python libraries from publicly available Python gallery, not from your ADO Artifacts feed
2. You want to use your managed feed to update and upgrade your dependencies for your solution. Thus, we need to update our CI process
3. Go back to **VS Code**
4. Select ***python-app-templates-build.yml*** file
5. We need to add a new task for **PIPAuthenticate**

```
steps:
- task: UsePythonVersion@0
  inputs:
    versionSpec: '${{ parameters.pythonVersion }}'
    addToPath: true

- task: PipAuthenticate@1
  inputs:
    artifactFeeds: 'CICDPipelineDemo/PublicPackageFeed'
    # If you have service connections for feeds from external organizations, list them here
    # serviceConnection: 'YourServiceConnectionName'
    onlyAddExtraIndex: true # Set to true to use PyPI as the primary index

- script: |
  python -m pip install --upgrade pip
```

6. Copy the following code and paste it in your YAML file

```
- task: PipAuthenticate@1
  inputs:
    artifactFeeds: [ADO org name]/[your project name]'
    onlyAddExtraIndex: true # Set to true to use PyPI as the primary index
```

where **[ADO org name]** is replaced with your ADO org name, and **[your project name]** with your ADO project name.

7. Check in changes to your repo. This will kick off a YAML pipeline process

Module 3 – Check CICD pipelines

1. In ADO, select the *Pipelines feature* in the *Pipelines menu* on the right
2. Select your YAML that uses templates (e.g. python-web-pipeline-template.yml)
3. Check if the CI process is triggered by your change, as well as CD process for your YAML pipeline
4. Is it successful?

Module 4 – Let's make changes

1. You have completed the basic tasks and activities so far. You are now able to create YAML pipelines to set up CI/CD processes
2. Let's make some changes and kick off another CI/CD process and see if your changes are reflected in the web application or not
3. Update or make some changes in one of *.html templates*, under ***/hello_app/templates/*** folder
4. Check in changes and kick off the CICD pipeline process
5. Check your Python application in Azure
6. Is it all good?
7. This is the end of the exercise