

# **Azure DevOps**

## **Hands-on Lab 1**

### **Introduction to Azure DevOps 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 1 - Introduction to Azure DevOps Pipelines .....	4
Prerequisites – Create an App Service in Azure (10min) .....	5
Prerequisite – Set up Demo Python App and Repo in ADO (20min) .....	6
Exercise 1- Cerate pipelines using Classic Mode (20min) .....	11
Module 1 – Create a simple Build pipeline.....	11
Module 2 – Add tasks to your pipeline.....	13
Module 3 – View Build pipeline result.....	16
Module 4 – Create a simple Release pipeline.....	17
Module 5 – View Release pipelines .....	20
Summary.....	21
Exercise 2- Understand Pipeline Features and Processes (1h) .....	22
Module 1 – Set up CICD pipeline .....	22
Module 2 – Update Release pipeline settings .....	24
Module 3 – Understand Variables (Q & As).....	25
Module 4 – Define Pipeline Variables.....	25
Module 5 – Use Variables in Variable Group .....	26
Module 6 – Understand pipeline Stages.....	28
Module 7 – Understand deployment conditions.....	29
Summary.....	30
Exercise 3 – Manage Secrets (15min) .....	32
Exercise 1 - Manage secrets in Key Vault.....	32
Module 2 – Link Key Vault secrets in Variable Group .....	34
Summary.....	35

## Lab 1 - Introduction to Azure DevOps Pipelines

<b>Objective(s)</b>	<ul style="list-style-type: none"> <li>To be able to get the basic understanding of Azure DevOps (ADO) pipeline features</li> <li>To be able to create Build and Release pipelines using the Classic mode</li> <li>To be able to view and manage your pipelines</li> <li>To be able to update the pipeline settings</li> <li>To be able to use variables using Pipeline Variables and Variables Group</li> <li>To be able to integrate with Azure Key Vault for more modern and secure way of managing secrets</li> </ul>
<b>Duration of Lab</b>	<ul style="list-style-type: none"> <li>1.5h + Prerequisites setups</li> </ul>
<b>Prerequisite(s)</b>	<ul style="list-style-type: none"> <li>Azure Subscription and Resource Group</li> <li><b>Contributor</b> Role in a selected Resource Group in Azure</li> <li>Git for Windows installed locally (<a href="#">Git for Windows</a>)</li> <li>Visual Studio Code editor</li> <li>Azure DevOps (ADO) organisation and project</li> <li><b>Contributor</b> role in the selected ADO project</li> </ul>
<b>Tool(s)</b>	<ul style="list-style-type: none"> <li>Azure Portal</li> <li>Azure DevOps (ADO)</li> <li>Visual Studio Code</li> </ul>
<b>Exercises</b>	<ol style="list-style-type: none"> <li>Create a simple Build pipeline using Classic Mode (20m)</li> <li>Understand pipeline Features and Processes (1h)</li> <li>Manage Secrets (15m)</li> </ol>
<b>Subscription</b>	[Selected Subscription]
<b>Resource Group</b>	[Selected RG]
<b>Navigation</b>	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.

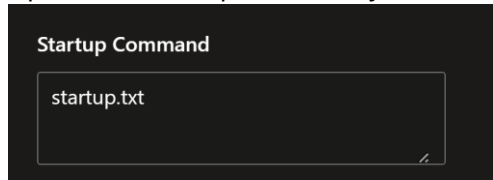
### Naming Convention for Labs

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](#)).

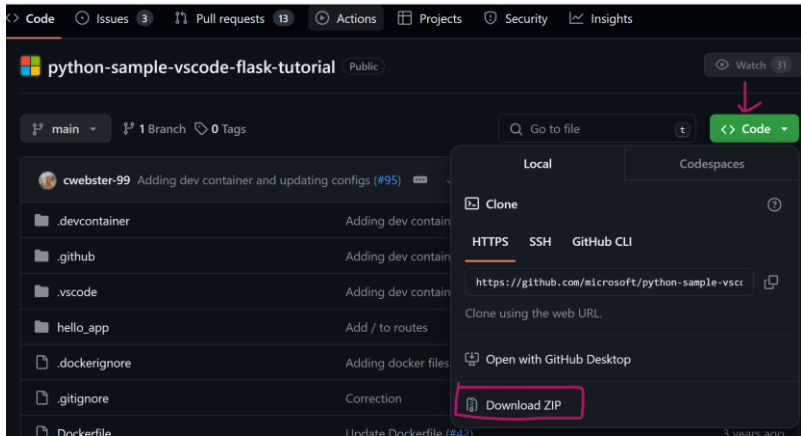
The naming convention below is designed to group your Azure resources together for easy access.

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

## Prerequisites – Create an App Service in Azure (10min)

Prerequisite(s)	<ul style="list-style-type: none"> <li>Existing App Service Plan in Azure - <b>at-asp-adodemo-pythonapps</b></li> <li><b>Contributor</b> role at a selected <i>Resource Group</i> in Azure</li> </ul>
Topics	<p>In this exercise, we will cover the following topics.</p> <ul style="list-style-type: none"> <li>Create a new App Service for your Python App</li> </ul>
Duration	<ul style="list-style-type: none"> <li>10min</li> </ul>
Tool(s)	<ul style="list-style-type: none"> <li>Azure portal</li> </ul>
Lab Scenario	<p>When you want to deploy a web-based application, there are a number of options available.</p> <p>For this exercise, we will provision an <b>App Service</b> to host your python application.</p>
Subscription	[selected subscription]
Resource Group	[selected RG]
Steps – Create App Service	<ol style="list-style-type: none"> <li>Follow the instructions provided in MS Online tutorial: “<i>Step 2 – Create a web app in Azure</i>” using <b>Azure portal</b>  <a href="#">Quickstart: Deploy a Python (Django or Flask) web app to Azure - Azure App Service   Microsoft Learn</a> </li> <li>Name your App Service (web app) as <b>[your short name]-web-python</b></li> <li>In the 3<sup>rd</sup> step – “<i>On the Create Web App page, fill out the form as follows.</i>”, select the <b>existing App Service Plan</b>, <b>at-asp-adodemo-pythonapps</b>, instead of creating a new App Service plan (in the item #5 in the Step 3)</li> <li>Skip the step 4 for specifying App Service Plan sku</li> <li>Create a Web app</li> </ol>
Steps – Configure Startup for Python App	<ol style="list-style-type: none"> <li>The <i>python-sample-vscode-flask-tutorial</i> app has a <b>startup.txt</b> file that contains the specific startup command for the web app. Set the web app startup-file configuration property to startup.txt.</li> <li>In your App Service in Azure, type “<b>config</b>” in the search box for the menus on the right</li> <li>Select the <b>Configuration</b> menu under the <i>Settings</i> section</li> <li>Update the <i>Startup Command</i> field with <b>startup.txt</b></li> </ol> <div data-bbox="502 1458 997 1637" data-label="Image">  <p>The screenshot shows a dark-themed interface with a label 'Startup Command' above a text input field. The input field contains the text 'startup.txt'.</p> </div> <ol style="list-style-type: none"> <li>Click the <b>Save</b> button</li> </ol> <p>Note: This step is explain in the online article - <a href="#">Build and deploy Python web apps with Azure Pipelines - Azure Pipelines   Microsoft Learn</a>. Follow the <i>Create an Azure App Service web app</i> section, <b>Step #4</b></p>

## Prerequisite – Set up Demo Python App and Repo in ADO (20min)

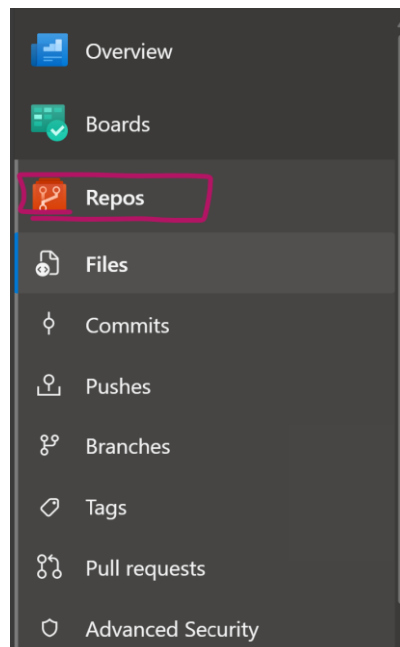
Prerequisite(s)	<ul style="list-style-type: none"> <li>ADO project for the lab</li> <li>Contributor role for the selected ADO project</li> <li>Python 3.9 or higher locally installed</li> </ul>
Topics	<p>In this exercise, we will cover the following topics.</p> <ul style="list-style-type: none"> <li>Download the sample Python app</li> <li>Create your own repo in ADO</li> </ul>
Duration	<ul style="list-style-type: none"> <li>20min</li> </ul>
Tool(s)	<ul style="list-style-type: none"> <li>Azure DevOps (ADO)</li> <li>Visual Studio Code</li> </ul>
Lab Scenario	<p>Throughout the lab exercises, we will use the sample Python app available from MS online.</p> <p>This application uses the “<b>Flask</b>” library for easy application development in Python. It also demonstrates how you can structure, create and manage your web application, pages, routing and all other things for your application.</p> <p>We will be following the sample instructions provided by the references below. There are slightly different steps that you need to follow, but those are instructed below.</p> <p>Note: You can fork the sample repo to your repo in ADO. For this lab, we will download the files and create your own repo manually</p>
References	<ul style="list-style-type: none"> <li><a href="#">Quickstart: Deploy a Python (Django or Flask) web app to Azure - Azure App Service   Microsoft Learn</a></li> <li><a href="#">Build and deploy Python web apps with Azure Pipelines - Azure Pipelines   Microsoft Learn</a></li> </ul>
Subscription	[selected subscription]
Resource Group	[selected RG]
Steps – Download sample app	<ol style="list-style-type: none"> <li>Download the sample file from <a href="https://github.com/Microsoft/python-sample-vscode-flask-tutorial">https://github.com/Microsoft/python-sample-vscode-flask-tutorial</a></li> </ol>  <ol style="list-style-type: none"> <li>Go to your downloaded local file location and unzip the file</li> </ol>

Dockerfile  
 LICENSE  
 README.md  
 requirements.txt  
 SECURITY.md  
 startup.py  
 startup.txt  
 test\_test1.py  
 uwsgi.ini  
 .dockerignore  
 .gitignore  
 hello\_app  
 .devcontainer  
 .github  
 .vscode

3. We don't need GitHub reference. Delete the **“.github”** folder

## Steps – Create a new repo in ADO

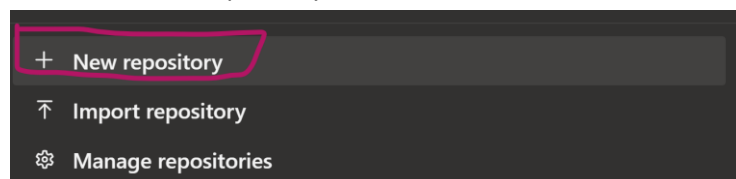
1. Open **ADO** and go to the selected *project*
2. Select the **Repos** menu on the left



3. In the middle at the top, your existing repo is shown in the breadcrumbs e.g.

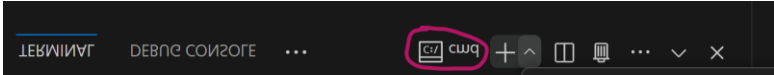
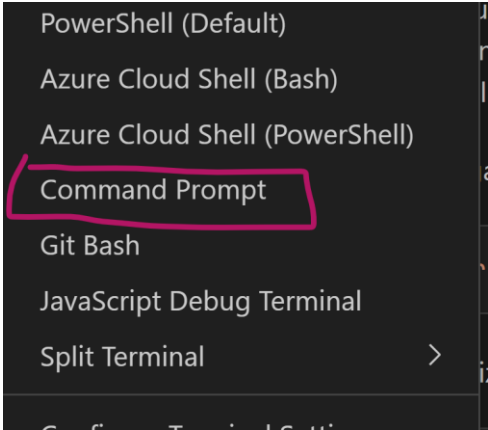


4. Click the down arrow at the end of the repo name
5. Select the New Repository button


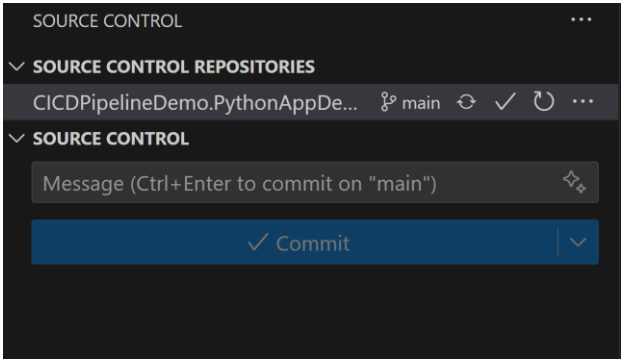


6. Specify the following settings and click the **Create** button

Area	Value
Repository Type	Git

	<table border="1"> <tr> <td>Repository Name</td><td>[your short name]-python-app-demo</td></tr> <tr> <td>Add README file</td><td>(Checked)</td></tr> </table> <p>7. Once your repo is created, it will show on the screen</p> <p>8. Copy the URL of your repo from the Browser's address bar at the top</p>	Repository Name	[your short name]-python-app-demo	Add README file	(Checked)
Repository Name	[your short name]-python-app-demo				
Add README file	(Checked)				
<b>Steps – add existing files to repo</b>	<ol style="list-style-type: none"> <li>We will add the sample app to the newly created repo. You can do several ways:             <ol style="list-style-type: none"> <li>Git bash (command tool) in Windows</li> <li>Git commands from VS Code</li> <li>Git integration using VS Code feature</li> <li>Manually add files to the repo in ADO</li> </ol> </li> <li>We will use Git commands in VS Code to initiate Git locally, add files, commit and push</li> <li>Open <b>Visual Studio (VS) Code</b></li> <li>Select the <b>Open Folder</b> option and select your sample python app folder</li> <li>In VS Code, open the <b>Terminal</b> by:             <ol style="list-style-type: none"> <li>Go to the <b>Terminal</b> menu at the top, select <b>New Terminal</b> option, or</li> <li>Press <b>Ctrl+Shift+`</b></li> </ol> </li> <li>The new Terminal window appears at the bottom (as default)</li> <li>The current Terminal type is indicated with the icon in the middle in the Terminal window. In this instance, the Terminal is <b>Command Prompt</b> below.              </li> <li>If it is not <b>Command Prompt</b>, click the the <b>down arrow</b> next to the <b>+</b> button and select the <b>Command Prompt</b> terminal type              </li> <li>Initialize a new Git repository in your project directory, run the command below:             <pre>git init</pre> </li> <li>Add your files to the repository. If you want to add all files in the current directory, you can use the following command:             <pre>git add .</pre> </li> <li>Commit your changes with a message             <pre>git commit -m "Your commit message"</pre> </li> </ol>				



	<p>12. Now, link your local repository to the remote ADO repository. Replace <b>[your-repo-url]</b> with the URL of your new repository that you copied:</p> <pre>git remote add origin your-repo-url</pre> <p>where “your-repo-url” is replaced with the URL of your repo</p> <p>13. Finally, push your changes to the remote repository:</p> <pre>git push -u origin master</pre>
<p><b>Check your repo</b></p>	<ol style="list-style-type: none"> <li>1. In <b>VS Code</b>, select the <b>Source Control</b> menu on the left (as default layout)</li> </ol>  <ol style="list-style-type: none"> <li>2. You can see your local repository under the <i>SOURCE CONTROL REPOSITORIES</i> section, and any changes made will appear under the <i>SOURCE CONTROL</i> section. e.g.</li> </ol>  <ol style="list-style-type: none"> <li>3. Go back to <b>ADO portal</b></li> <li>4. Check your repo. All files are committed to your repo now</li> </ol>
<p><b>Steps – Run your application locally</b></p>	<ol style="list-style-type: none"> <li>1. Go back to <b>VS Code</b></li> <li>2. In the same <i>Terminal window</i>, run the following commands (one by one)</li> </ol> <pre># Create a virtual environment python -m venv .venv  # Activate the virtual environment .venv\Scripts\activate  # Upgrade pip python -m pip install --upgrade pip  # Install the required packages pip install -r requirements.txt  # Set the environment variables, FLASK_APP with the hello_app.webapp module export set FLASK_APP=hello_app.webapp  # Run the Flask application python3 -m flask run</pre>

Note: If you get an error running “**export set...**” command above, just use “**set...**” without “**export**”

3. When you run the last command above, you will get this kind of message.

```
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
```

4. URL with IP address above is the application instance running locally. Copy the URL (**http://127.0.0.1:5000**)
5. Open Web Browser and paste the URL in the address bar at the top
6. You can see the sample Python app running
7. To **quit**, go back to the *Terminal window in VS Code*, press **CTRL+C**

Note:

The session for the virtual environment you’ve created is kept active while you’re using VS Code for your project.

Every time you want to run your application, run the last command and open the Web Browser for your app.

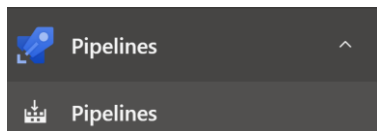
**Python3 -m flask run**

## Exercise 1- Create pipelines using Classic Mode (20min)

<b>Prerequisite(s)</b>	<ul style="list-style-type: none"> <li>Azure DevOps (ADO) organisation</li> <li>ADO project with <b>repo</b> for Python application</li> <li><b>ADO service connection</b> accessing to a selected <i>Azure Resource Group (RG)</i> with the <b>Contributor role</b></li> <li>You are added to the <b>Contributor role</b> at the selected <i>RG</i></li> <li>Visual Studio Code editor</li> <li>Local copy of your Python application, synced from your ADO repo</li> </ul>
<b>Topics</b>	<p>In this exercise, we will cover the following topics.</p> <ul style="list-style-type: none"> <li>Create a simple Build pipeline</li> <li>Add tasks for your pipeline</li> <li>View Build pipeline result</li> <li>Create a simple Release pipeline</li> <li>View Release pipelines</li> </ul>
<b>Duration</b>	<ul style="list-style-type: none"> <li>20 min</li> </ul>
<b>Tool(s)</b>	<ul style="list-style-type: none"> <li>Azure DevOps (ADO)</li> </ul>
<b>Lab Scenario</b>	You want to create CI/CD processes using ADO Pipelines for you to be able to deploy your application, code changes etc...
<b>Subscription</b>	[selected subscription]
<b>Resource Group</b>	[selected RG]

### Module 1 – Create a simple Build pipeline

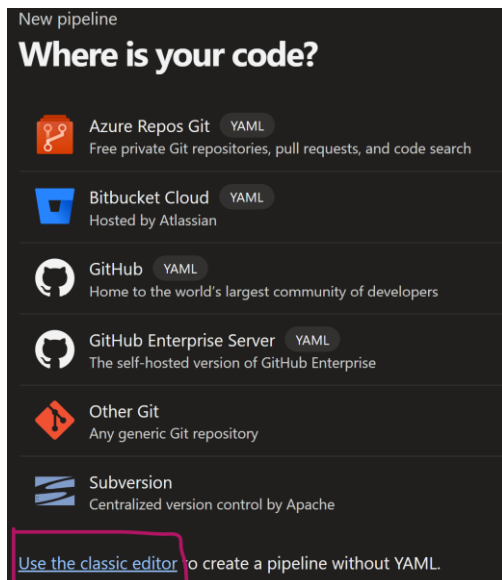
- Go to ADO and select your project
- Select the **Pipelines menu** on the left



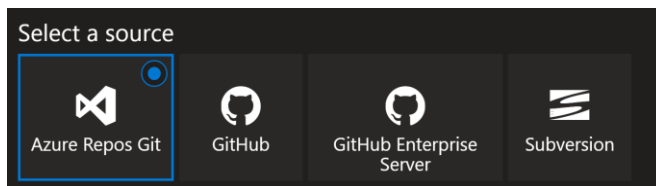
- Click the **New pipeline button** at the top right



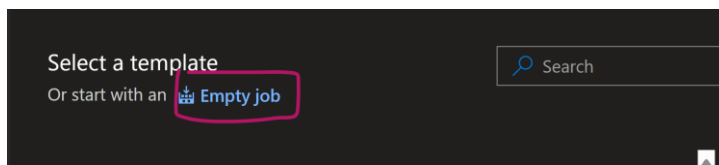
- Scroll down and find the link: **“User the classic editor to create a pipeline without YAML”**



5. In the “Select your repository” pane, select **Azure Repos Git**

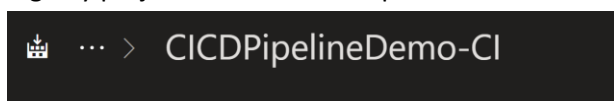


6. “Team Project” is default to your ADO project
7. “Repository” – select your repository
8. “Default branch” is set to **main**
9. Click the **Continue** button
10. In the “Choose a template” pane, select the **Empty job** at the top

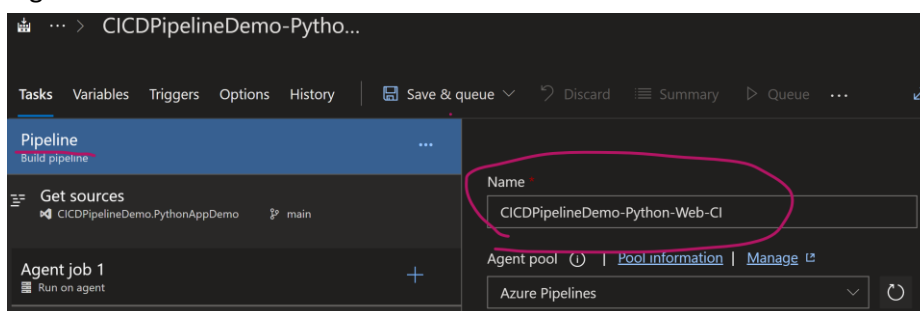


11. At the top, you will see the name of your pipeline. The name is created with the default project name with suffix “CI”

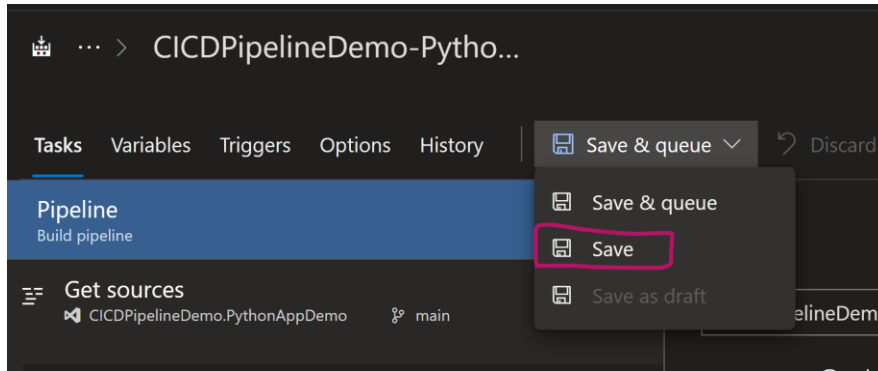
e.g. my project name is “CICDPipelineDemo” like below.



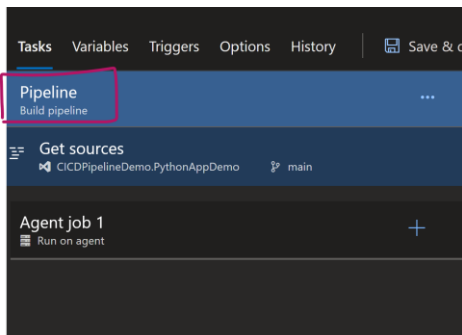
12. Click the text and update the name for your pipeline: **[you short name]-simple-demo-CI**
  13. Alternatively, you can update the name of your pipeline in the main editor pane
- e.g.



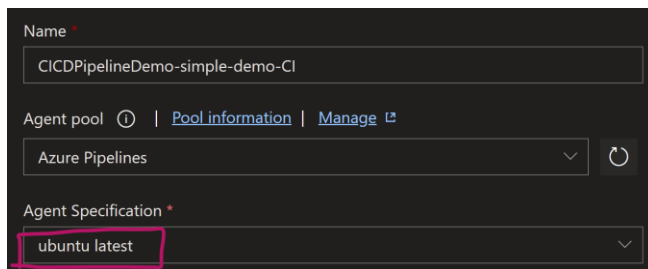
14. At the moment, your pipeline is created, but not physically saved yet. If you want to save your empty pipeline, select the **down arrow next to Save & queue** button, and click **Save**



15. In the *Tasks tab*, select the **Pipeline** box

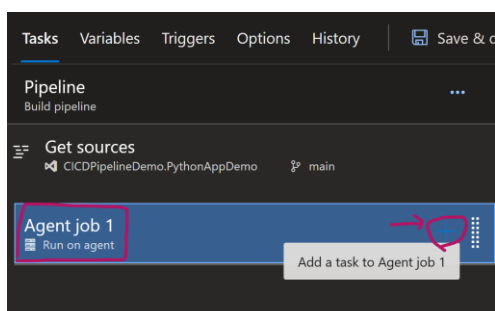


16. The “Agent pool” fields is where you specify which hosting type (VM) type you’re going to use.
  - a. **Azure pipelines:** MS managed pipeline VMs/Machines in Azure. You don’t need to worry about anything and let MS take care of the underlying machines/infrastructure
  - b. **None (Default):** Tasks that do not need a agent (VM/Machine) to run
17. In the *Detail section* on the right, change the “Agent Specification” field to **ubuntu latest**. At the moment, Python app works well with Linux OS on Azure App Service as it is optimised for Linux OS. So, we select *ubuntu* here

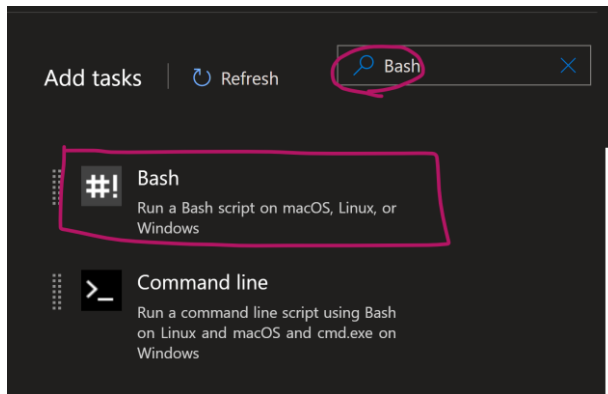


## Module 2 – Add tasks to your pipeline

1. In the same pipeline Editor window, select the **Agent job 1** box on the right
2. Click the **+ button** to add a new task



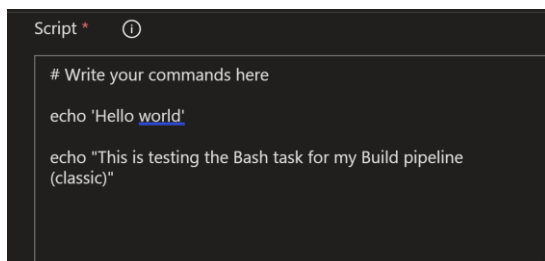
- Pre-built tasks are available for you. Type "Bash" in the search box
- Select the **Bash** task type and click the **Add** button



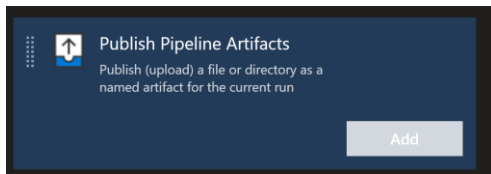
- The new task is added under the "Agent Job 1"
- Select the **Bash Script** task on the right
- Specify the following fields

Area	Value
Task version	3.*
Display name	Bash Script (default value).  <b>Note:</b> You can change and update your task name here. When you creating your steps, it's best to name your task that is descriptive, readable and identifiable. This name appears in our pipeline logs. So, it makes it easier for you to read your logs
Type	Select <b>Inline</b> option
Advanced	(This is where your scripts are included)  Enter the following command under "echo 'Hello world'"  <pre>echo "This is testing the Bash task for my Build pipeline (classic)"</pre>

- The entire script looks like this



- Add another task. Repeat the step 2 above
- Enter "publish" in the search box
- Select the **Publish Pipeline Artifacts** task and click the **Add** button. A new task is added below the *Bash Script* task

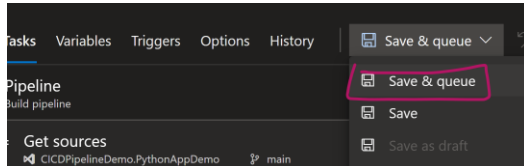


12. Select the newly added task
13. Specify the following fields

Area	Value
Task version	1.*
Display name	(leave it as it is, or you can rename this)
File or directory path	\$(Pipeline.Workspace)  <b>TODO:</b> Click the "... " button next to the path field and check to see your file structure
Artifact name	drop
Artifact publish location	Azure Pipelines
	(leave the rest of the fields as they are)

Note: This exercise is not building and compiling your application. We are simply copying files to the shared artifact location, so your Release pipeline can link some artifacts for deployment process.

14. Click the Save & queue button



15. Enter the "Save comment" field, e.g. "Initial Build run" and click the **Save and Run** button

Run pipeline

Select parameters below and manually run the pipeline

Save comment

Initial Build run

Agent pool

Azure Pipelines

Agent Specification \*

ubuntu latest

Branch/tag

main

Select the branch, commit, or tag

Advanced options

☐ Enable system diagnostics

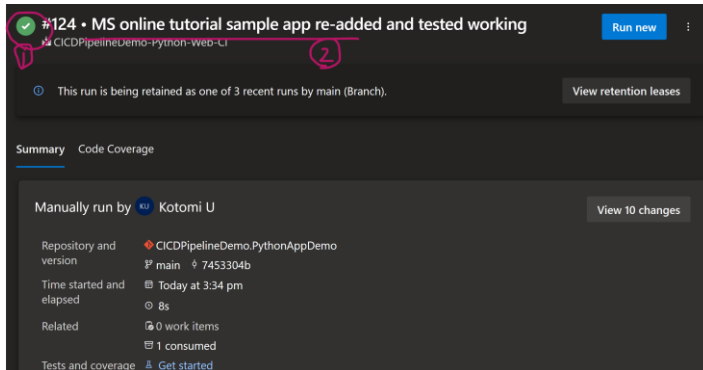
Cancel

Save and run


16. This will save your changes and put a new Build process in the queue

### Module 3 – View Build pipeline result

1. (By default), when you queue your pipeline request, it automatically opens the pipeline window
2. Once the pipeline is completed (success/failure), you will see the colour icon indicating your status



3. The **Green** icon on the left top indicates the successful run of your selected pipeline
4. The description (indicated by #2 above) is the comments from the last changeset (latest check-in change(s))
5. When you scroll down, you can see also the status for the *Agent jobs*

Jobs		
Name	Status	Duration
 Agent job 1	Success	3s

6. Click on the Agent job 1 task result
7. You can see logs for your pipeline steps and processes

← Jobs in run #124

CICDPipelineDemo-Python-Web-CI

Jobs

- ✓ Agent job 1 3s
- Initialize job <1s
- ✓ Checkout CICDPi... 1s
- ✓ Bash Script <1s
- ✓ Post-job: Check... <1s
- Finalize Job <1s
- Report build st... <1s


✓ Agent job 1

```

1 Pool: Azure Pipelines
2 Image: ubuntu latest
3 Agent: Hosted Agent
4 Started: Today at 3:35 pm
5 Duration: 3s
6
7 ▶ Job preparation parameters
29 ▶ fx 1 queue time variable used

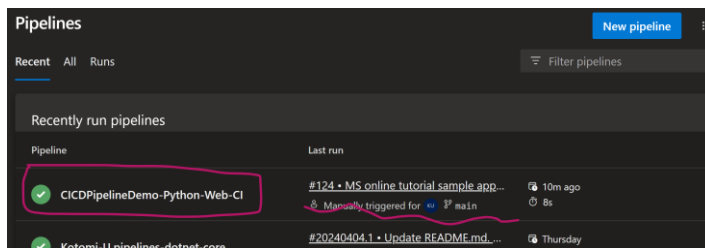
```

8. As can be seen above, the task that you've added to run scripts is logged there correctly as well
9. Go through each task and check what has completed
10. This is where you do troubleshooting if/when your pipeline fails. Failed pipeline as well as jobs/tasks are indicated with the **Red** icon

11. Click the Pipelines menu, , on the left menu panel
12. You will see the list of all pipelines
13. Find your pipeline. You can see the recently run pipeline status as **Green** and the details of the pipeline run

e.g.

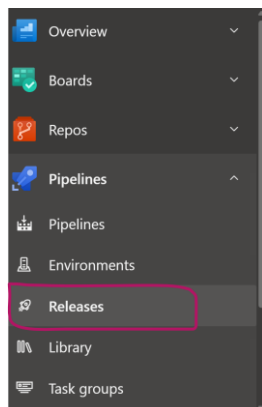




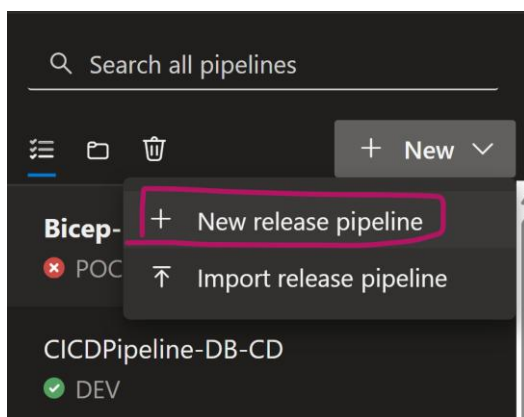
14. Select your pipeline
15. You can see the history of your pipeline executions here

## Module 4 – Create a simple Release pipeline

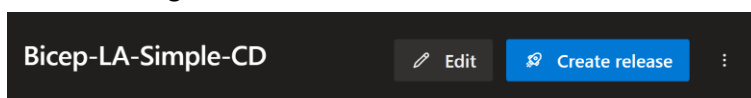
1. Select the Pipelines menu on the left and select Releases menu



2. This is where the classic pipelines are displayed
3. Click the + New button and create the + New release pipeline

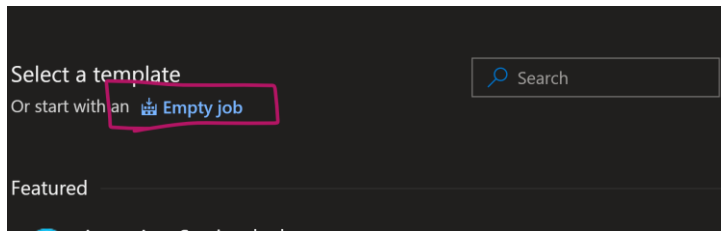


Note: Do not get confused with the **Create release** button on the right pane.

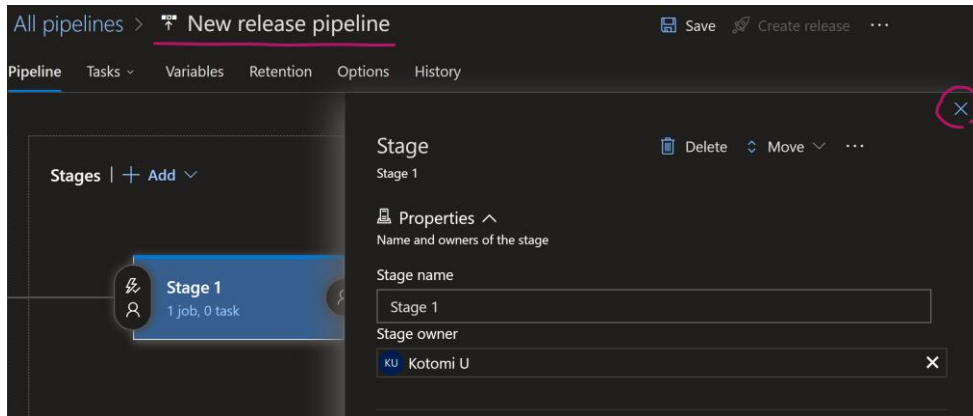


This is to create a new Release pipeline process for the selected pipeline. In this example, a new Release pipeline process is created for “Bicep-LA-Simple-CD” pipeline. So, you can initiate your Release process

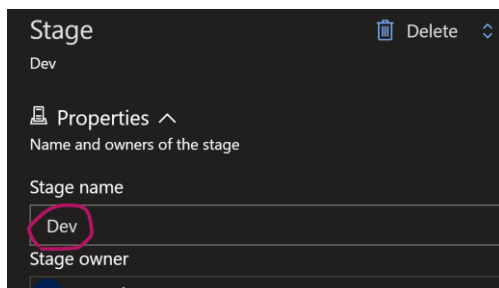
4. The “Select a template” window appears. Select the **Empty job** at the top



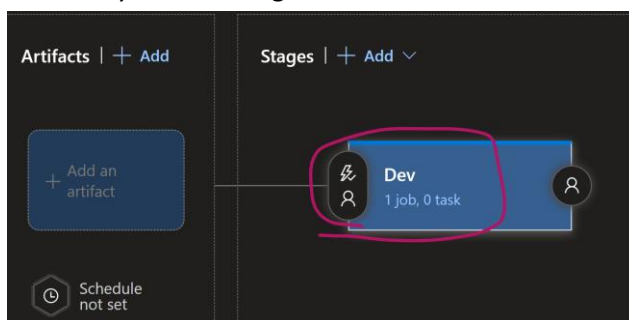
- When the editor pane is first opened, you will see the sidebar pane on the right. Close the editor pane by clicking X button on the top right corner



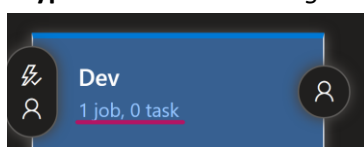
- Change the name of your Release pipeline at the top by clicking the text "New release pipeline":  
**[your short name]-simple-demo-CD**
- At the moment, your Release pipeline is a standalone pipeline, meaning that it does not depend on anything. You can manually trigger the pipeline execution
- Click the **Stage 1** box in the editor. Change the "Stage name" field: **Dev**



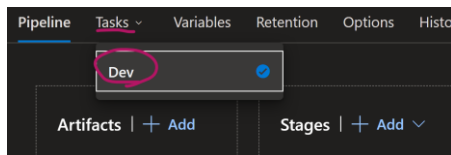
- Close the editor pane by clicking the X button at the top right
- What do you see changed now?



- You've now created the *Dev Stage* for your pipeline. You will add one or more than Stages later on in the exercise
- Let's add some tasks for your Release pipeline
- Click the **hyperlink** in the *Dev Stage* box, or



14. Select the Tasks tab at the top



15. We have one Agent job for the *Dev Stage*

16. Select the Agent job in the right and specify the following fields

Area	Value
Display name	Agent Job (default)
Agent selection	<p>Azure pipelines</p> <p><b>Note:</b> The "Agent pool" fields is where you specify which hosting type (VM) type you're going to use.</p> <ul style="list-style-type: none"> <li>• <b>Azure pipelines:</b> MS managed pipeline VMs/Machines in Azure. You don't need to worry about anything and let MS take care of the underlying machines/infrastructure</li> <li>• <b>Default (no agents):</b> tasks that do not require an agent (VM/Machine) to run</li> <li>• <b>Agent pool(s):</b> your self-hosted agent pool created by ADO Admin for you</li> </ul>
Agent Specification	Ubuntu latest
	Leave the rest of the fields unchanged

17. Add a new task by clicking the + button in the *Agent Job*

18. Enter "**Bash**" in the **search box**

19. Select the **Bash** task and click the **Add** button

20. The *Bash Script* task is added under the Agent Job now

21. Specify the following field

Area	Value
Task version	3.*
Display name	<p>Bash Script (default value).</p> <p><b>Note:</b> You can change and update your task name here. When you creating your steps, it's best to name your task that is descriptive, readable and identifiable. This name appears in our pipeline logs. So, it makes it easier for you to read your logs</p>
Type	Select <b>Inline</b> option
Advanced	<p>(This is where your scripts are included)</p> <p>Enter the following command under "echo 'Hello world'"</p> <pre>echo "This is testing the Bash task for my Release pipeline (classic)"</pre>

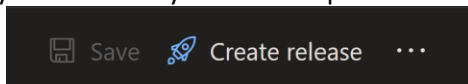
22. The entire script looks like this

```
Script *
# Write your commands here

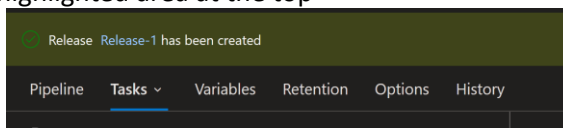
echo 'Hello world'

echo "This is testing the Bach task for my Release pipeline
(classic)"
```

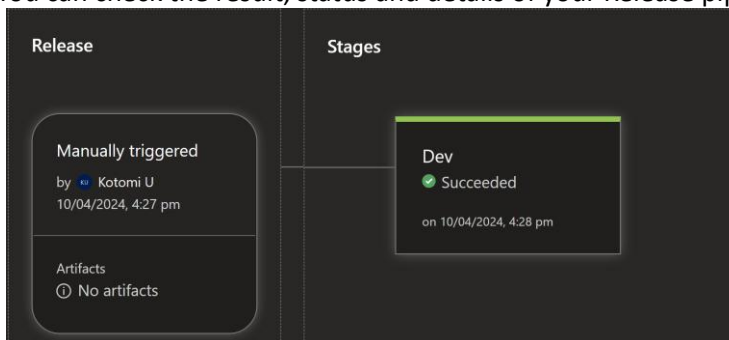
23. Click the Save button to save your changes
24. Enter some comments above your changes in the *Comment textbox*, e.g. “New Release pipeline created”
25. Click the **OK** button
26. When you save the changes, you will see the Create release button appeared at the top. This is for you to initiate your Release process



27. Click the **Create Release** button
28. Enter the “Release description”. The comment entered here will appear in the pipeline history. So, make sure that your comments describe your release purpose (e.g. testing initial run, etc...)
29. Click the **Create** button
30. You can check the progress and status of your Release pipeline execution. Click the URI in the highlighted area at the top




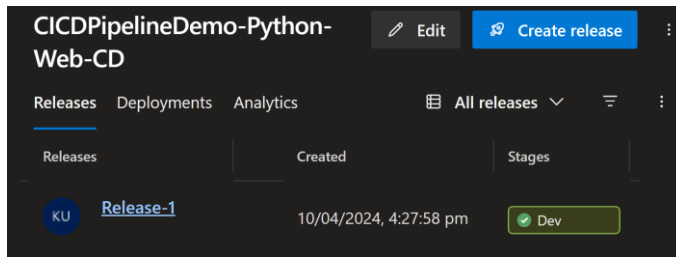
31. You can check the result, status and details of your Release pipeline execution



32. As can be seen, my *Dev Stage* processes/steps are successful. The pipeline was triggered by manual process (not automated way)

## Module 5 – View Release pipelines

1. Click the **Pipelines** menu, , on the left menu panel and select the **Releases** menu
  2. You will see the list of all Release (classic) pipelines on the left pane
  3. Find your pipeline. You can see the recently run pipeline status as **Green** and the details of the pipeline run
- e.g.



4. You can check the details of each execution by selecting a specific run on the right side of the pane
5. This is the end of the Exercise 1!

## Summary



### ACHIEVEMENTS

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

- ✓ Create Build and Release pipelines using the Classic mode in ADO
- ✓ Trigger pipelines manually
- ✓ View all pipelines
- ✓ Check pipeline progress and status

## Exercise 2- Understand Pipeline Features and Processes (1h)

<b>Prerequisite(s)</b>	<ul style="list-style-type: none"> <li>Azure DevOps (ADO) organisation</li> <li>ADO project with <b>repo</b> for Python application</li> <li>ADO service connection accessing to a selected Azure Resource Group (RG) with the <b>Contributor role</b></li> <li>You are added to the <b>Contributor role</b> at the selected RG</li> <li>Visual Studio Code editor</li> <li>Local copy of your Python application, synced from your ADO repo</li> </ul>
<b>Topics</b>	<p>In this exercise, we will cover the following topics.</p> <ul style="list-style-type: none"> <li>Set up CICD pipeline</li> <li>Update Release pipeline settings</li> <li>Understand Variables (Q &amp; As)</li> <li>Define Pipeline Variables</li> <li>User Variables in Variable Group</li> <li>Understand pipeline Stages</li> <li>Understand deployment conditions</li> </ul>
<b>Duration</b>	<ul style="list-style-type: none"> <li>60 min</li> </ul>
<b>Tool(s)</b>	<ul style="list-style-type: none"> <li>Azure portal</li> <li>Azure DevOps (ADO)</li> </ul>
<b>Lab Scenario</b>	<p>You will explore various feature available for ADO pipelines in this exercise.</p> <p>At the end of the exercise, you should be comfortable to create and manage your pipeline stages and processes, and be able to understand the basic security considerations.</p>
<b>Subscription</b>	[selected subscription]
<b>Resource Group</b>	[selected RG]

### Module 1 – Set up CICD pipeline

- At the moment, your Release pipeline is independent of code changes and runs based on manual trigger. We can set up continuous deployment (CD) process to deploy your changes to an environment(s), every time changes are checked in to your selected repo
- Go to **ADO** and select the **Pipelines** menu on the left
- Select the **Releases** menu under the Pipelines menu
- Select your Release pipeline
- Click the **Edit** button at the top right
- In the *Pipeline tab*, in the Artifacts box, click the **Add an artifact** button

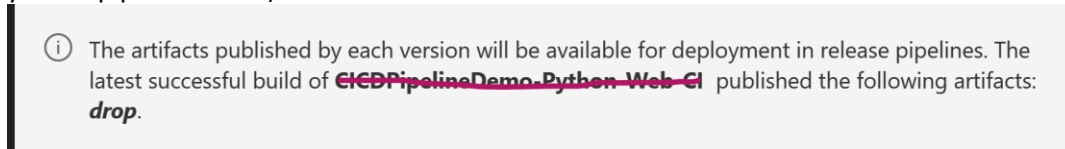


- In the Add an artifact pane, specify the following fields

Area	Value
Source Type	Build
Project	[your Python app project]

Area	Value
Source (build pipeline)	[your Build pipeline for simple-demo-ci]
Default version	Latest
Source alias	<p>(leave it as it is)</p> <p>Note:</p> <p>The Source alias value has your CI pipeline name and is prefixed with the underscore “_” e.g. _[your ci pipeline name]</p> <p>This is the file location created by your CI process. So, do not remove the underscore symbol</p>

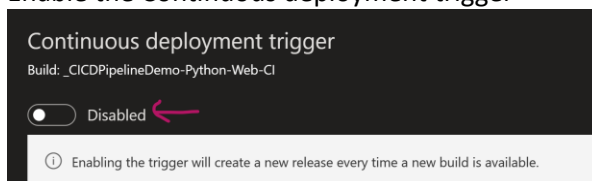
8. You might have some information displayed like below. (e.g. the text with strikethrough in Purple is your CI pipeline name)



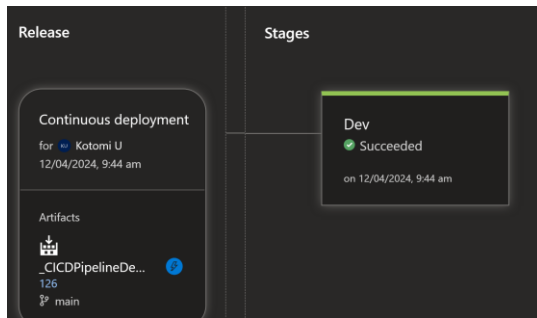
9. Click the **Add** button
10. Now, your CD pipeline is linked with your CI process (code change on your repo)
11. Click the **lightning symbol** on the Artifact box



12. Enable the Continuous deployment trigger

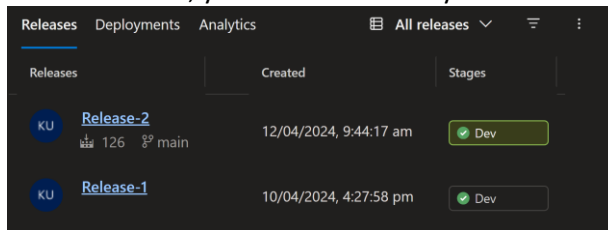


13. Close the side pane by clicking the X button on the top right corner
14. Click the **Save** button at the top, add some comment, and click the **OK** button
15. Go back to your *CI pipeline* under the Pipelines menu
16. Click the **Run pipeline** button at the top to manually trigger the Build process, and click the **Run** button
17. On successful completion, go back to your *CD pipeline* under the *Releases menu*
18. In the history of your CD pipeline, you can see the newly executed run history. Check the timestamp
19. Select the newly executed history by clicking the URL
20. You can see that continuous deployment was successful triggered upon your Build completion  
e.g.

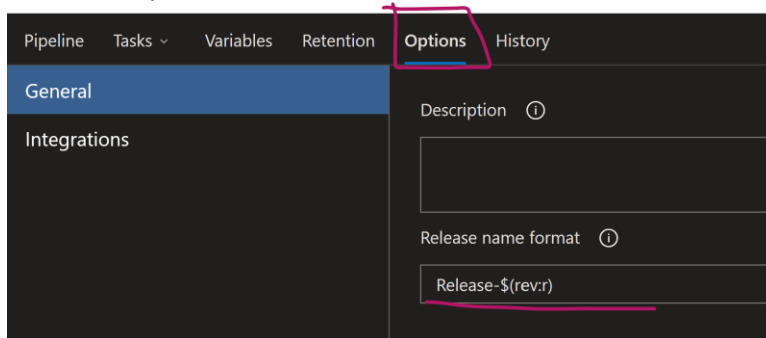


## Module 2 – Update Release pipeline settings

1. At the moment, your Releases history looks like this

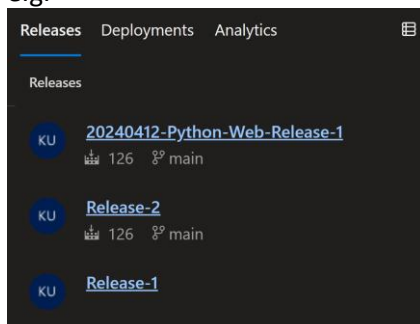


2. You can configure the settings for your pipeline to make it readable or descriptive as well
3. Click the **Edit** button at the top
4. Select the *Options* tab



5. In the *General* section, update the “Release name format” field to: **\$(date:yyyyMMdd)-Python-Web-Release-\$(rev:r)**
6. Click the **Save** button at the top, enter some comment and click the **OK** button
7. Click the Create release button at the top to trigger the Release pipeline process
8. Go back to the summary/history pane by clicking the breadcrumb at the top
9. Check if your newly created Release has the different value

e.g.



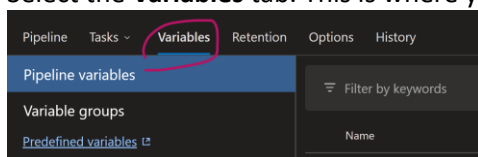


## Module 3 – Understand Variables (Q & As)

1. Take a quick look at and read through the article to understand variables: [Define variables - Azure Pipelines | Microsoft Learn](#) (5min)
2. How many variable types are described in the article?
3. The answers are below:
  - a. User-defined variables
  - b. System variables
  - c. Environment variables
4. What is the difference between the syntax of “`${{variables.[name]}}`” and `$([namr])`, where `[name]` indicates the name of variable you defined?
5. The answer is:
  - a. `${{variables.[name]}}` returns the value set at the initiate state
  - b. `$([name])` returns the value at runtime. If the variable value is set or changed during the process, it shows the update value here
6. Where can you define your variables in YAML pipelines?
7. The answers are below:
  - a. At the root
  - b. At the Stage level
  - c. At the Job level
8. What data type is supported for variables?
9. The answer is – String and it is mutable
10. When you define the same variable in multiple places with the same name, how does it work?
11. The answer is – the most locally scoped variable wins
12. How can you share variables across multiple pipelines in your project?
13. The answer is – Variable groups
14. What is Secret variables?
15. The answer is – they are encrypted at rest with a 2048-bit RSA key

## Module 4 – Define Pipeline Variables

1. In ADO project, select the *Pipelines menu* on the left and select the *Releases*
2. Select your *Release pipeline*
3. Click the Edit button at the top right corner
4. Select the **Variables** tab. This is where you define your variables



5. In the **Pipeline variables** section, click the **+Add** button in the main pane
6. Enter the *name*: **vSample**`[your short name]`
7. Enter any arbitrary value
8. Click the **Save** button at the top, enter some comments and click the **Save** button
9. At the moment, your variable value is simple string. Anyone can see the value
10. Select the **Tasks** tab and update our step to use the variable
11. In the *Bash Script* task, update your *Script* to include the command below
 

```
echo "I am accessing the pipeline variable: $([your variable name])"
```

 where `[your variable name]` is replaced with the name of your variable
12. Click the **Save** button, enter comments and click **OK**
13. Click the **Create release** button to kick off your Release pipeline
14. At the bottom of the pane, enter comments like “variable added in the script” and click **Create**
15. Let’s check your result out by clicking the hyperlink in the **Green** bar at the top
16. Select the Stages box, and your specific stage (in this case “Dev”). Is it **Green**?

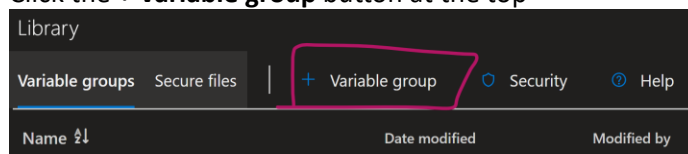
17. Let's check the history of the "Bash Script" task
18. Did the result show your variable value within the script task?

```

1:32:51.5713063Z =====
1:32:51.5713223Z Task           : Bash
1:32:51.5713285Z Description  : Run a Bash script on macOS, Linux, or Windows
1:32:51.5713398Z Version    : 3.237.1
1:32:51.5713464Z Author     : Microsoft Corporation
1:32:51.5713544Z Help       : https://docs.microsoft.com/azure/devops/pipelines/
1:32:51.5713683Z =====
1:32:51.9900746Z Generating script.
1:32:51.9915499Z ===== Starting Command Output =====
1:32:51.9926280Z [command]/usr/bin/bash /home/vsts/work/_temp/7a31f5d9-9bdd-45d5-8
1:32:51.9980591Z Hello world
1:32:51.9981047Z This is testing the Bash task for my Release pipeline (classic)
1:32:51.9981571Z I am accessing the pipeline variable: SampleVariableValue
1:32:51.9992230Z
  
```

## Module 5 – Use Variables in Variable Group

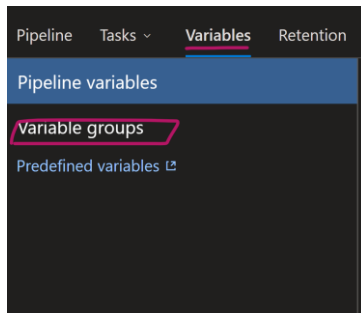
1. When you have a group of variables that you want to share across multiple pipelines in your project, you can use *Variable Group*. Then, team members do not need to create, copy and paste the same variables repeatedly in their pipelines
2. You can enhance security for managing secrets using *Variable Group* as well. The *Library* feature in ADO Pipelines that provides Variable Group functionality has its own security controls to manage user permission.  
For instance, only an authorised team member(s) can create and update variables including secret variables. Other team members are given Read-Only permission to use those variables. So, secrets values are not exposed to other team members
3. Select the *Pipelines menu* on the left and select **Library**
4. Click the **+ Variable group** button at the top



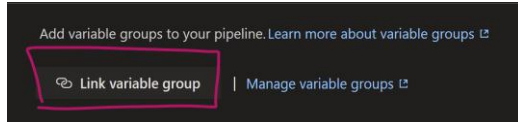
5. Enter the following fields:

Area	Value
Variable group name	vg-[your short name]-python-app-pipelines
Description	Variable group used for python app pipelines demo
Link secrets from Azure key vault	Disabled
Variables	<p>Add 2 variables: non-secret and secret variables.</p> <ol style="list-style-type: none"> <li>1) Non secret variable               <ol style="list-style-type: none"> <li>a. Name: <b>vgSample1</b></li> <li>b. Value: [any arbitrary value]</li> </ol> </li> <li>2) Secret variable               <ol style="list-style-type: none"> <li>a. Name: <b>vgSecretVar1</b></li> <li>b. Value: [any arbitrary value that indicates it's a secret]</li> <li>c. Click the Lock icon to make it "secret" variable</li> </ol> </li> </ol>

6. Click the **Save** button at the top
7. Now, we need to make this Variable Group available in your pipeline
8. Go to the *Releases* under the *Pipelines menu* on the left
9. Select your *Release pipeline* and click the Edit button at the top right corner
10. Select the *Variables tab* and select the **Variable groups** section on the right pane



11. Click the **Link variable group** button



12. Select your *Variable group*
13. At the moment, the scope is set to the **"Release"**. That means any Release pipeline can use this Variable group
14. Click the **Link** button
15. Let's use those variables in the pipeline process
16. Select the *Tasks* tab at the top
17. In the *Bash Script* task, update your script

```
18. echo "This is the variable from Variable Group: $(vSample2)"
19.
20. echo "This is the secret variable from Variable Group: $(vSample3)"
```

where you replace *"vSample2"* and *"vSample3"* with your variable names

21. Click the **Save** button and add comments like "Variables in Variable group added"
22. Click the **Create release** button to kick off your Release pipeline process
23. Check the result and history, and check variables in Variable group is correctly used

```
073Z ===== Starting Command Output =====
019Z [command]/usr/bin/bash /home/vsts/work/_temp/e41e9c48-3989-46b0-ba6
001Z Hello world
015Z This is testing the Bash task for my Release pipeline (classic)
071Z I am accessing the pipeline variable: SampleVariableValue
089Z This is the variable from Variable Group: VariableGroupSampleValue
022Z This is the secret variable from Variable Group: ***
030Z
```

24. Did you notice your secret variable doesn't show the value?



Microsoft makes an effort to mask secrets from appearing in Azure Pipelines output, but you still need to take precautions.

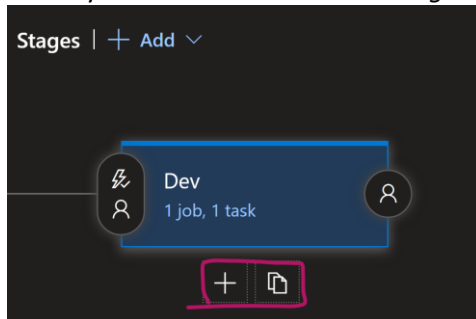
- **NEVER** echo secrets as output
- Some operating systems log command line arguments. **NEVER** pass secrets on the command line. Instead, we suggest that you map your secrets into environment variables.

Microsoft never masks substrings of secrets. If, for example, "abc123" is set as a secret, "abc" isn't masked from the logs.

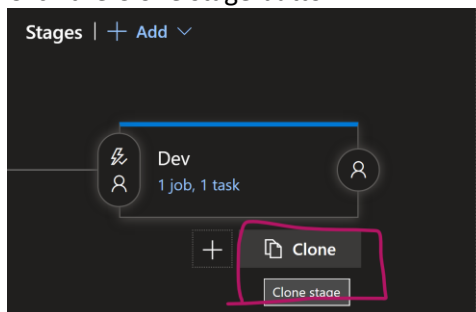
This is to avoid masking secrets at too granular of a level, making the logs unreadable. For this reason, secrets should not contain structured data. If, for example, "{ \"foo\": \"bar\" }" is set as a secret, "bar" isn't masked from the logs

## Module 6 – Understand pipeline Stages

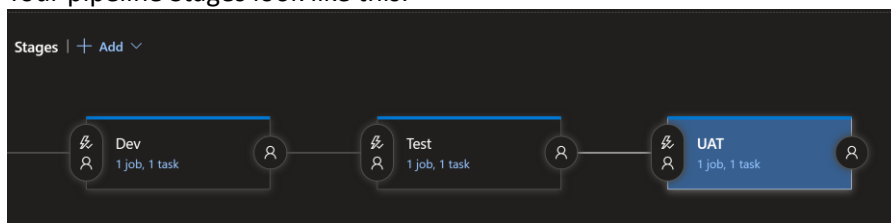
1. Select your Release pipeline in the *Releases* under the *Pipelines menu* on the left
2. Click the **Edit** button
3. We add 2 more Stages in the Release pipeline: **“Test”** and **“UAT”**
4. There are 2 options to add a new Stage: **Add** (a brand new Stage) or **Clone** (copy existing Stage)
5. Hover your mouse over the *Dev Stage box*. Add and Clone buttons will appear



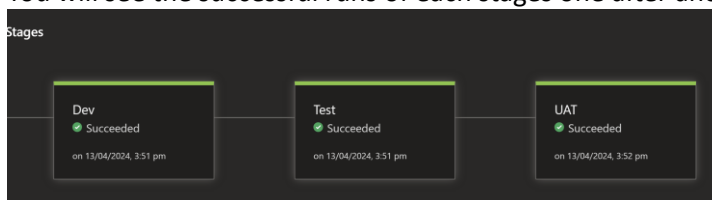
6. Hover your mouse over each button
7. Click the Clone stage button



8. A new Stage is added after the *Dev Stage*, as **“Copy of Dev”**
9. Update the name of the stage as **“Test”**
10. Add another stage for **“UAT”**
11. Your pipeline Stages look like this:



12. At the moment, your pipeline process runs in a linear manner, meaning that each stage runs after the successful run of the previous stage
13. Let's see how it works. **Click** the Save button at the top and create a new Release
14. Click the hyperlink for your Release pipeline process and check the results
15. You will see the successful runs of each stages one after another

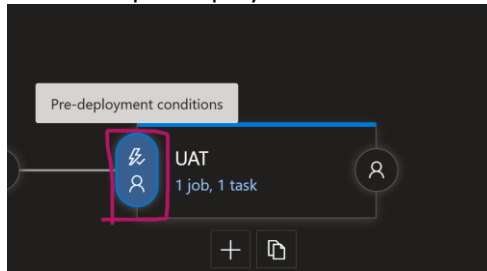


16. Let's check the pipeline histories by clicking the name of your Release pipeline on the breadcrumbs at the top
17. The history of your newly updated pipeline has 3 stages (in this case, 3 environments)

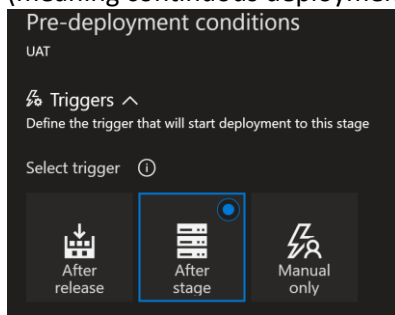
KU	<a href="#">20240413-Python-Web-Release-3</a>	13/04/2024, 3:50:54 pm	<span>🟢 D..</span> <span>🟢 T..</span> <span>🟢 U..</span>
KU	<a href="#">20240413-Python-Web-Release-2</a>	13/04/2024, 3:00:14 pm	<span>🟢 D..</span>
KU	<a href="#">20240413-Python-Web-Release-1</a>	13/04/2024, 1:32:32 pm	<span>🟢 D..</span>

## Module 7 – Understand deployment conditions

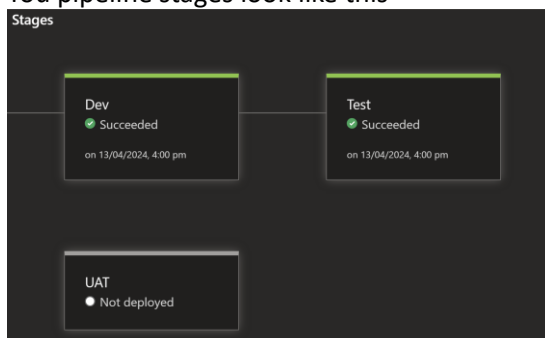
1. What if you do not want to create continuous deployment for the UAT environment? You can set that in manual trigger mode
2. Go back to the **Edit** screen
3. Select the pre-deployment condition for the UAT stage



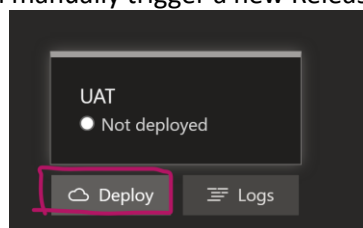
4. As you can see, the current setting for the Pre-deployment conditions is set to “After stage” (meaning continuous deployment). Change it to **Manual only**



5. Save the change and trigger a new Release
6. You pipeline stages look like this

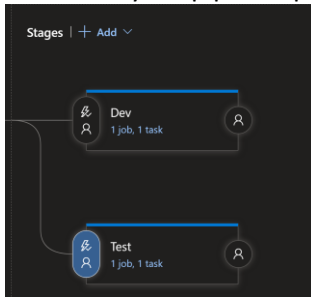


7. The UAT stage is disconnected because the stage is only triggered manually
8. You can manually trigger a new Release for the UAT stage at any time



9. What if you want to deploy to Dev and Test simultaneously (in parallel)?
10. Select the **Pre-deployment conditions** for the *Test stage*
11. Change the trigger to **After release** option

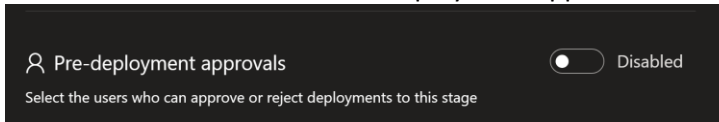
12. How does your pipeline process look like?



13. Now, we want to add an approver before Test release

14. Select the **Pre-deployment condition** for the *Test stage*

15. Scroll down and enable the Pre-deployment approvals

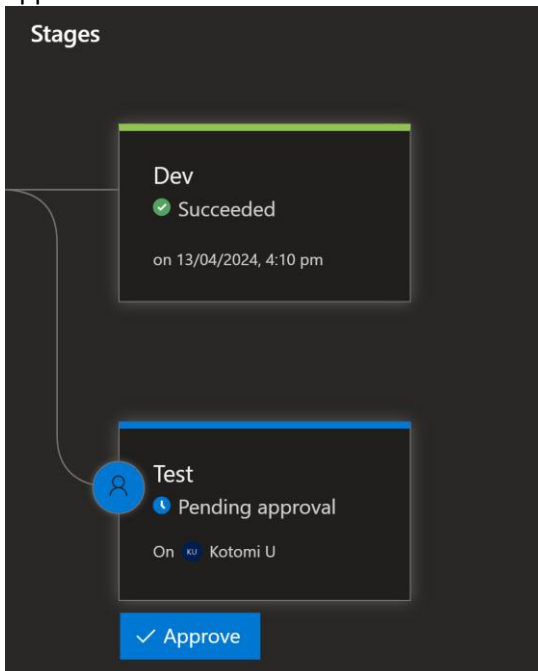


16. Select at least one approvers. Enter your name

17. Save the change and trigger a new Release

18. What happened?

19. At the moment, the Release process for the Test stage (environment) is awaiting on the Pending approval



20. An approver(s) will receive the email notification typically contains information about the release pipeline that requires approval, including the name of the pipeline, the stage that requires approval, and the changes being deployed.

The email also includes links to the ADO portal where the approver can review the changes and either approve or reject the deployment.

21. **Approve** the release

22. This is the end of the Exercise 2!

## Summary



### ACHIEVEMENTS

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

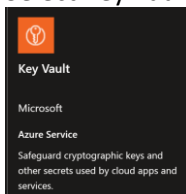
- ✓ Create continuous deployment (CD) processes automatically triggered by your CI process
- ✓ Create pipeline variables and manage
- ✓ Create Variable Group and manage
- ✓ Manage secrets better
- ✓ Create more Stages in the pipeline and manage them in various pre-deployment conditions
- ✓ Set up an approver(s) for the Release process

## Exercise 3 – Manage Secrets (15min)

<b>Prerequisite(s)</b>	<ul style="list-style-type: none"> <li>Azure DevOps (ADO) organisation</li> <li>ADO project with <b>repo</b> for Python application</li> <li>ADO service connection accessing to a selected Azure Resource Group (RG) with the <b>Contributor role</b></li> <li>You are added to the <b>Contributor role</b> at the selected RG</li> <li>Visual Studio Code editor</li> <li>Local copy of your Python application, synced from your ADO repo</li> </ul>
<b>Topics</b>	In this exercise, we will cover the following topics. <ul style="list-style-type: none"> <li>Manage secrets in Key Vault</li> <li>Link Key Vault secrets in Variable Group</li> </ul>
<b>Duration</b>	<ul style="list-style-type: none"> <li>15 min</li> </ul>
<b>Tool(s)</b>	<ul style="list-style-type: none"> <li>Azure portal</li> <li>ADO</li> </ul>
<b>Lab Scenario</b>	In this exercise, we will learn more modern and secure approach to manage your secrets using Azure Key Vault.
<b>Subscription</b>	[selected subscription]
<b>Resource Group</b>	[selected RG]

### Exercise 1 - Manage secrets in Key Vault

- Go to **Azure portal** (or open another browser tab, *portal.azure.com*) and select your *Resource Group*
- Click the **+ Create** button at the top in the main pane
- Type “key vault” in the search box
- Select Key Vault and click the **Create** button



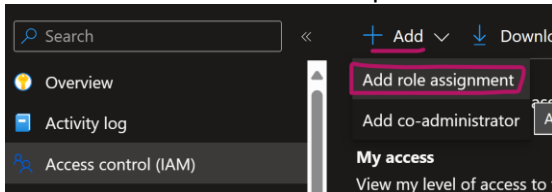
- Enter the following details

Area	Value
<b>Subscription</b>	[your subscription]
<b>Resource Group</b>	[your Resource Group]
<b>Key vault name</b>	at-kv-[your short name]-python-web
<b>Region</b>	Australia East
<b>Pricing tier</b>	Standard
	(Leave the rest as they are)
<b>Access configuration tab</b>	
<b>Permission model</b>	Azure role-based access control
<b>Networking tab</b>	

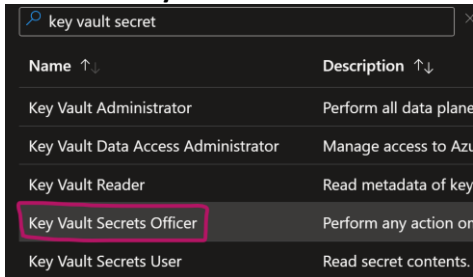


Area	Value
	(Leave as they are)
Review, and create	

- We will create a Key Vault: **at-kv-[your short name]-python-web**
- Once the resource is created, go to the Key Vault resource
- Since we chose to use Azure RBAC for permission, at this point, you are not allowed to create new secrets. This is because Azure Key Vault separates management and data operations. Thus, you need to add yourself to Key Vault data operation role
- Select the **Access control (IAM)** menu on the left
- Click the **+Add** button at the top and select the **Add role assignment**



- Type **"key vault secret"** in the search box
- Select the **Key Vault Secrets Officer** role and click **Next**



- Add yourself to this role by clicking the **+ Select members** link
- Click the **Review + assign** button
- You are now successfully added to this role
- Select the **Secrets** menu under the *Objects section*
- Click the **Generate/Import** button on the top
- Enter the following details

Area	Value
Upload options	Manual
Name	kvSecretSample1
Secret Value	[any arbitrary value]

- For the next module, you also need to add *ADO Service Connection (App Registration created in AAD)* to be added to the **Key Vault Secrets User** role this gives READ access to the Key Vault
- Repeat the step 9 & 10 above and find the Key Vault Secrets User role
- Assign the App Registration (created for ADO Service Connection) to that role

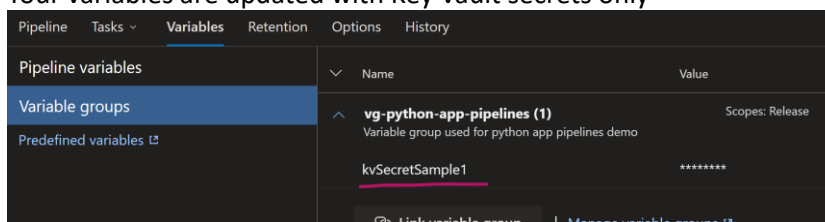
**Note:** You can not use the **hyphen (-)** for your secret name. ADO Variable Group does not support the hyphen separated secret name from Key Vault. Instead, if you need to separate wordings, you need to use the **underscore (\_)** or camel/pascal case.

## Module 2 – Link Key Vault secrets in Variable Group

1. In ADO, select the **Library** in the *Pipelines menu* on the left
2. Select the Variable Group that you have created in the previous exercise
3. We will update this configuration to include Azure Key Vault
4. Enable the **Link secret from an azure Key vault as variables** option
5. Enter the following details

Area	Value
Azure subscription	[Your service connection name]
Key vault name	[your KV name]
Variables	Click the <b>+Add</b> button. If you have more than one secrets, you can add all secrets or select a specific secret here.

6. Click the **Save** button for Key Vault details
7. Click the Save button to save changes for your Variable Group
8. When you integrate Variable Group with Azure Key Vault, the existing variables are overridden and disappear, because your secret management and source of the data becomes Azure Key Vault, not ADO Library
9. Let's check what happens to the Pipeline Variables that you have set up for your pipeline previously
10. Select your *Release pipeline* in the *Releases* under the *Pipelines menu* on the right
11. Select the *Variables tab* and the *Variable groups section* (if not refreshed, refresh the browser)
12. Your variables are updated with Key Vault secrets only

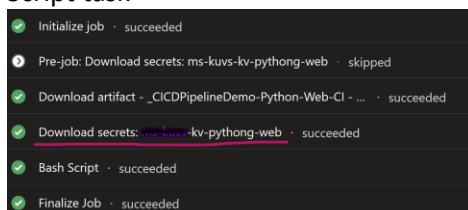


13. You can always create another Variable Group for variables that are not linked to Azure Key Vault, add that to your Pipeline Variables
14. We need to fix the existing pipeline steps
15. Select the **Tasks-Dev** tab (you might want to set the pre-deployment conditions for Test and UAT to be manual trigger, to avoid continuous deployment execution)
16. Remove the last 3 commands echoing the variables from the script
17. Include the new command below

18. `echo "This is the secret linked from Azure KV: $(vSample4)"`

where "vSample4" is replaced with your secret name in Key vault - **kvSecretSample1**

19. Save the changes and trigger a new Release pipeline process
20. Success?
21. Check the Release logs by selecting the **Logs** tab for your Release
22. The new task is added to download secrets from the specific Key Vault before the existing Bash Script task



23. Your script task uses the variable that is linked to the Key Vault secret

```
2024-04-13T05:09:24.4816731Z Hello world
2024-04-13T05:09:24.4817180Z This is testing the Bash task for my Release pipeline (classic)
2024-04-13T05:09:24.4818354Z This is the secret linked from Azure KV ***
2024-04-13T05:09:24.4822166Z
```

24. That's the end of the exercise3!

## Summary



### ACHIEVEMENTS

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

- ✓ Create Azure Key Vault with secrets
- ✓ Link your Variable Group with Key Vault
- ✓ Use secrets as variables in your pipelines