Hosting Python Packages in Azure DevOps

User Story Ref:

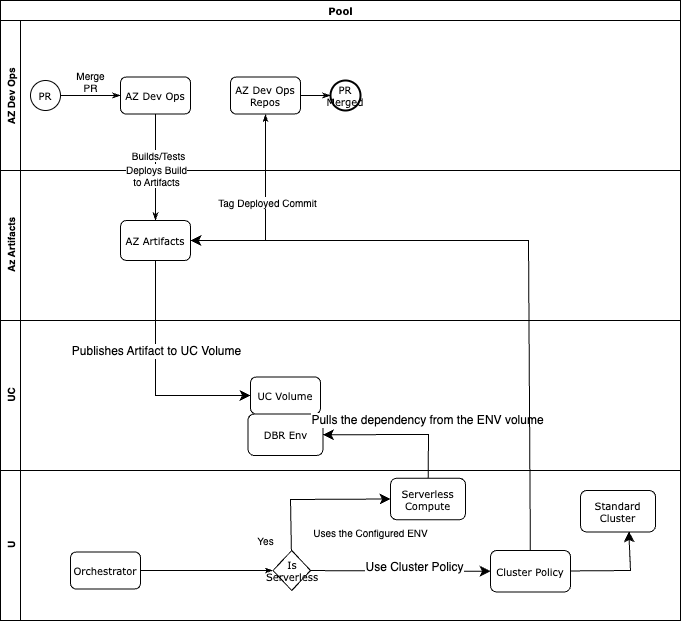
<https://dev.azure.com/adv-dev/Data%20and%20Analytics%20Portfolio/_workitems/edit/252500>

**Objective**

Build a Python wheel package using Visual Studio Code, upload it to a specified volume location, and update the workflow pipeline to use the wheel file.

Execute a Databricks job with the wheel package from the volume location using serverless compute.

Design a process for hosting Python packages in Azure DevOps, including package development, publishing to Azure Artifacts, installation in Databricks, and automation using Azure Pipelines.



**Implementation Part 1 – Completed:**

Doc:

[Python wheel package v1.0.pptx](https://advantagesolutionsnet.sharepoint.com/:p:/r/sites/DataLakePlatform/Shared%20Documents/Data%20Engineering/Foundation/Docs/Python%20wheel%20package%20v1.0.pptx?d=w07427ccbfa124ea28ee27017d1427179&csf=1&web=1&e=NlHKyM)

**Build the Wheel Package**

Utilize Visual Studio Code to implement the necessary code changes.

Generate the wheel (.whl) file using the appropriate build commands.

**Upload to Volume Location**

Transfer the generated wheel file to the designated volume storage for accessibility within Databricks.

**Update the Workflow Pipeline**

Modify the workflow pipeline configuration to reference the new wheel package.

Ensure dependencies and path configurations are correctly set.

**Run Databricks Job**

Execute a Databricks job utilizing serverless compute with the wheel package from the volume location.

Validate that the job runs successfully.

**Testing**

Successfully tested the metadata\_fetcher task, confirming proper execution with the updated package.

Proceed with further validations for additional tasks as needed.

**Python Wheel File for Databricks Workflow**

**Overview**

The solution involves packaging Python code into a wheel file that can be deployed and executed within Databricks workflows, enabling serverless computation capabilities.

**Clone the repository into VSCode:**

<https://github.com/Advantage-Solutions/dnap-ingestion-framework.git>

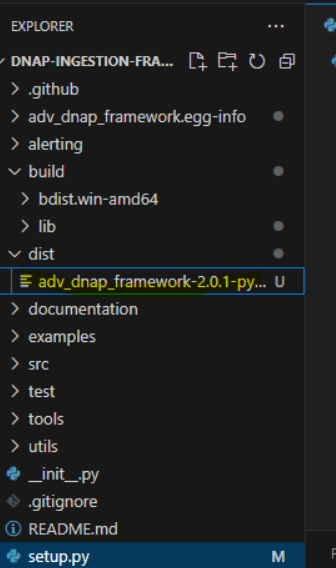
**Change the branch to:**

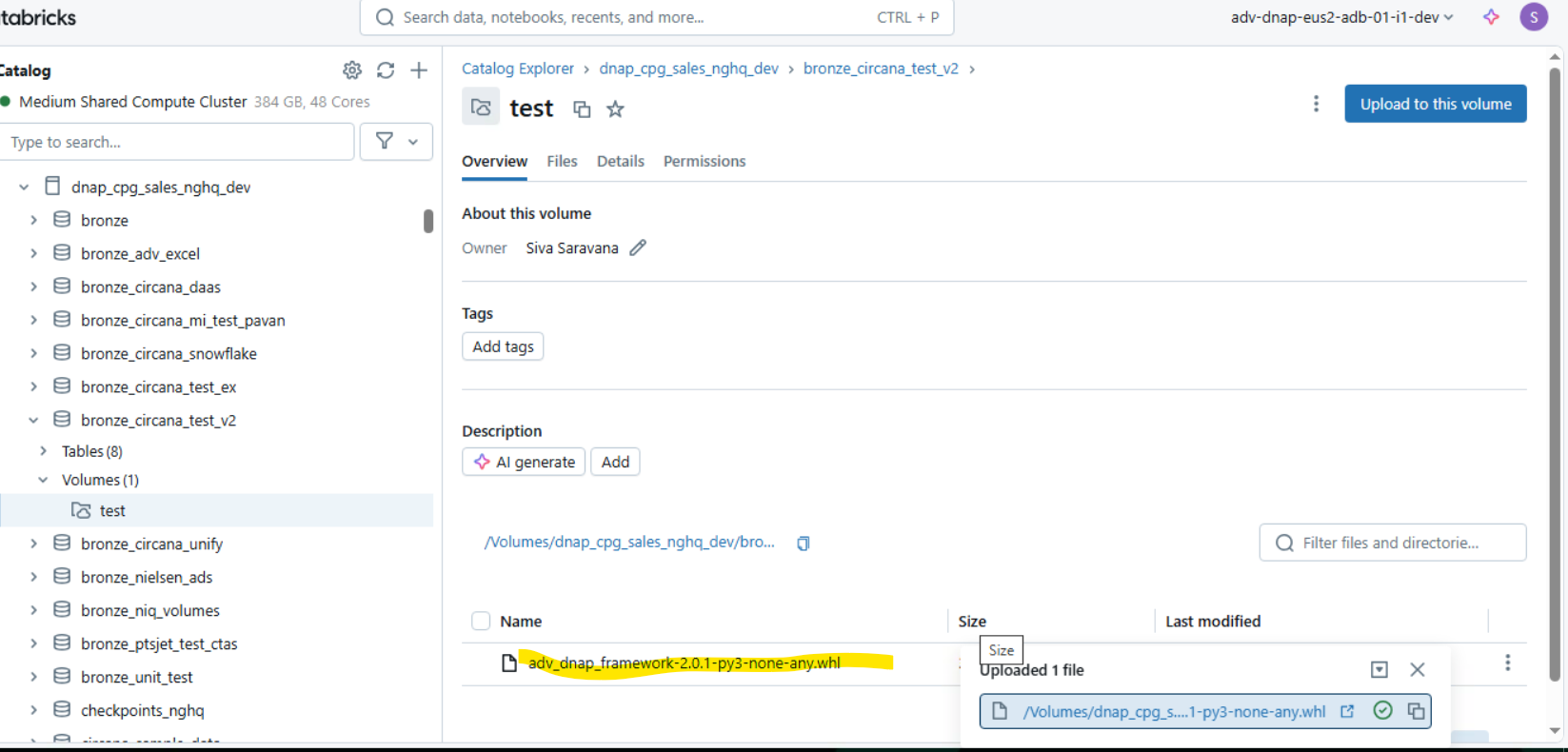
foundation/siva.saravana/252500\_wheel\_package

**Build the code in VSCode:**

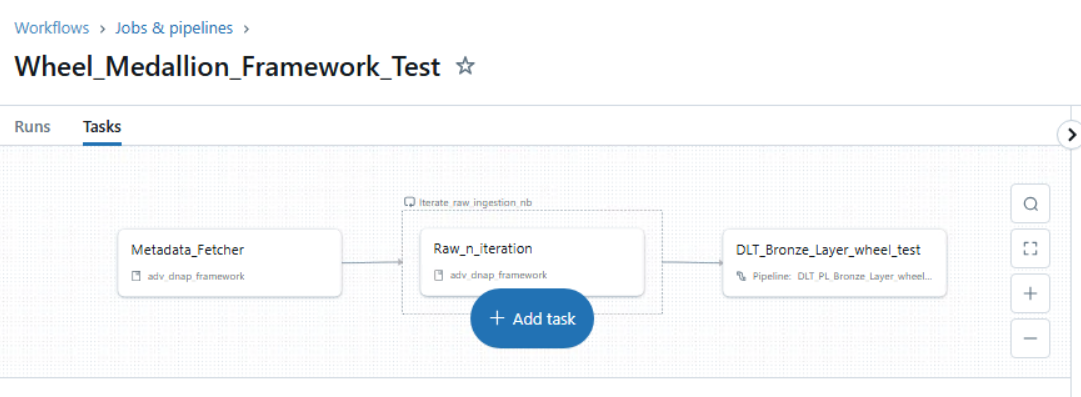
python setup.py bdist\_wheel

**Copy the wheel file from dist and upload it to the volume location:**





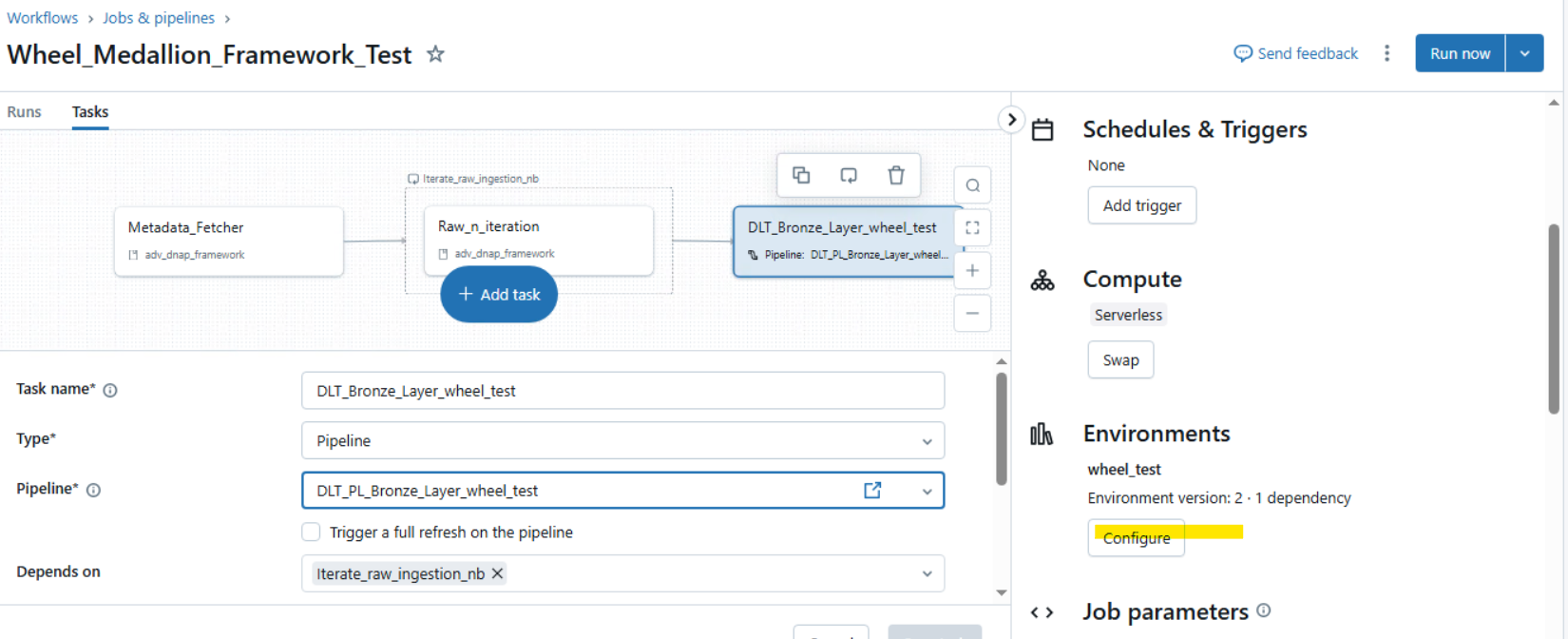
**Databricks Workflow**

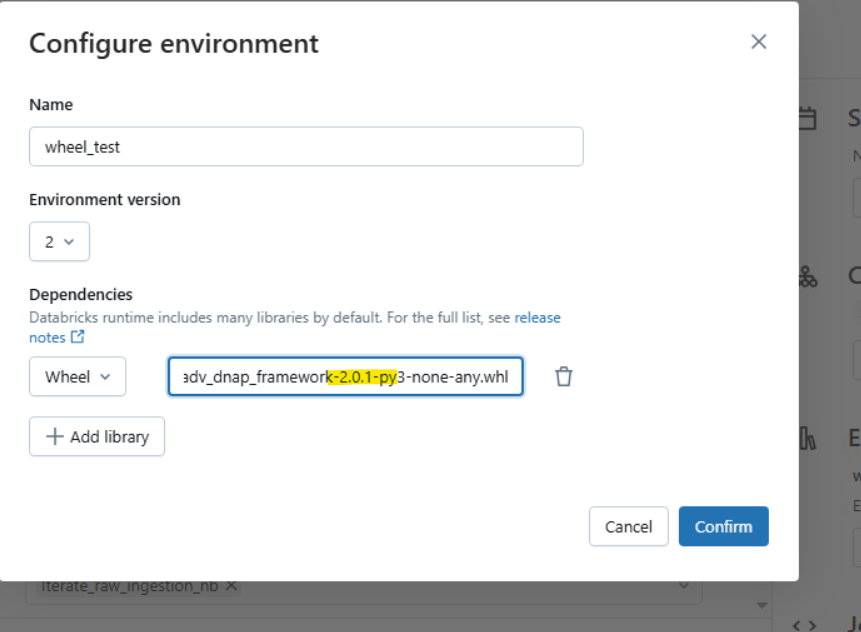


**Databricks Workflow URL**

<https://adb-4661267993302765.5.azuredatabricks.net/jobs/833584566124535/tasks?o=4661267993302765>

**Update wheel package version:**





Run and test the pipeline

**Steps to Host Python Packages in Azure DevOps (part 2)**

1️. Develop & Package the Python Code

1. Write your Python package/module.
2. Use setuptools to create a .whl (wheel) file.

python setup.py bdist\_wheel

If build is not installed, install it using:

pip install build

2️. Upload Package to Azure Artifacts

1. Install required dependencies:

pip install twine keyring artifacts-keyring

1. Authenticate to Azure DevOps.
2. Upload the .whl file to Azure Artifacts.

3️. Install the Package in Databricks

1. Install it directly in Databricks using pip install --index-url.
2. Optionally, install it at the cluster level via UI.

4️. Copy the Package to Unity Catalog Volume

1. Define a Unity Catalog Volume (dbfs:/Volumes/...).
2. Use dbutils.fs.cp to copy the .whl file.

5️. Install from Unity Catalog Volume

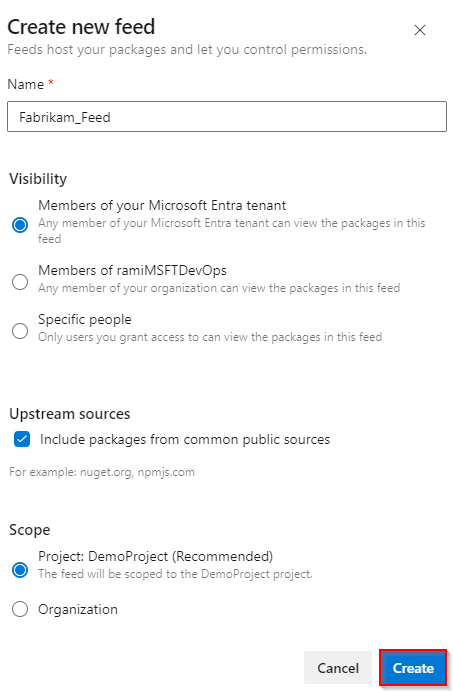
1. Install the package from the volume in any Databricks notebook or cluster.

6️. Automate with a Databricks Job

1. Create a Databricks notebook to handle the deployment.
2. Schedule a Databricks Job to automate future installations.

**Creating a New Artifact Feed in Azure DevOps**

1. Sign in to your Azure DevOps organization and navigate to your project.
2. Go to Azure DevOps > Artifacts > Create New Feed.
3. Provide a Name for your feed and set its visibility:
   1. Private (for internal use)
   2. Public (for broader access)
4. Enable "Include packages from common public sources" if needed.
5. Choose whether the feed should be scoped to the project or entire organization.

Click Create.

**Creating an Azure Pipeline for Artifact Creation**

1. Navigate to Pipelines > Create a new pipeline.
2. Follow the 4-step setup process:

Connect: Link your code repository (e.g., GitHub).

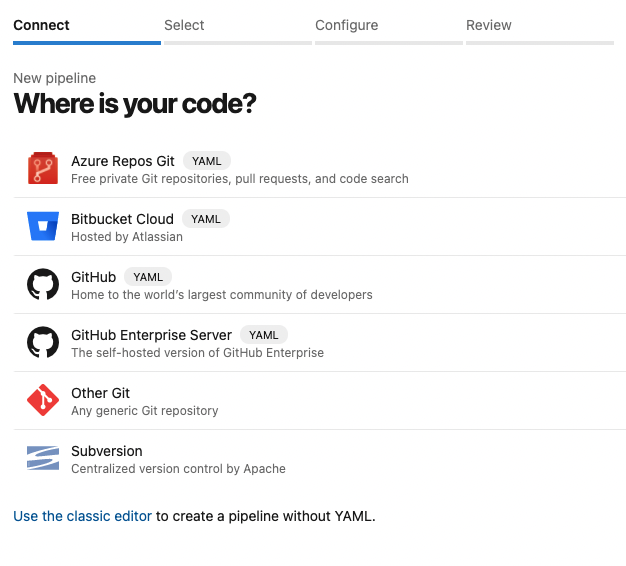
Select: Choose the repository containing your Python package.

Configure: Use the YAML script provided below.

Review: Verify settings and run the pipeline.

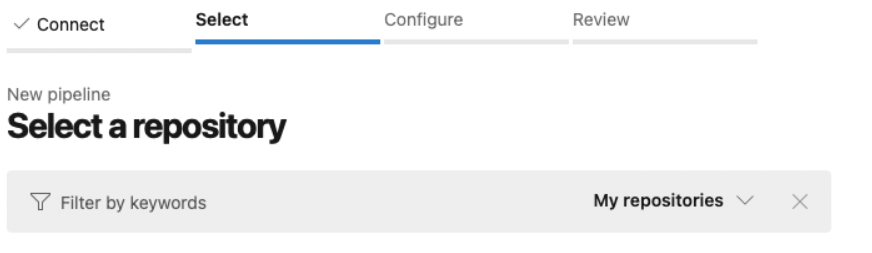
**Connect**

The python package which I am planning to host is in my GitHub. So I am going to connect my GitHub with Azure DevOps and authorise the same.



Select

The required repository is selected.



**Configure**

Under Configure tab, select Starter pipeline and then In the Azure Pipelines, use the yml script

trigger:

- master

pool:

vmImage: 'ubuntu-latest'

strategy:

matrix:

Python36:

python.version: '3.6'

steps:

- task: UsePythonVersion@0

inputs:

versionSpec: '$(python.version)'

displayName: 'Use Python $(python.version)'

- script: |

python -m pip install --upgrade pip

displayName: 'Install dependencies'

# - script: |

# pip install pytest pytest-azurepipelines

# pytest

# displayName: 'pytest'

- script: |

python setup.py sdist

displayName: 'Artifact creation'

- task: CopyFiles@2

inputs:

targetFolder: $(Build.ArtifactStagingDirectory)

- task: PublishBuildArtifacts@1

inputs:

PathtoPublish: '$(Build.ArtifactStagingDirectory)'

ArtifactName: 'dist'

publishLocation: 'Container'

**Final Steps**

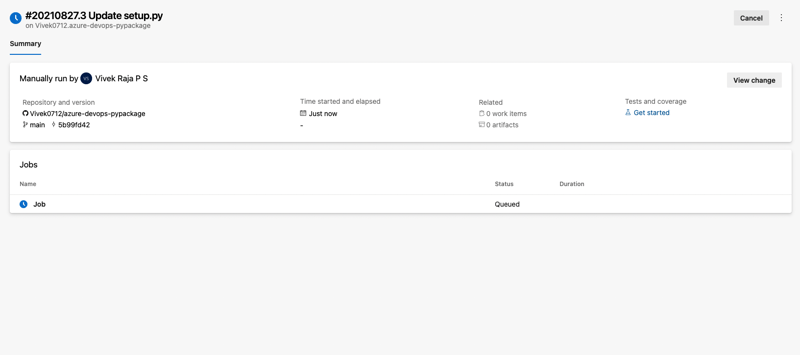
Review the pipeline configuration.

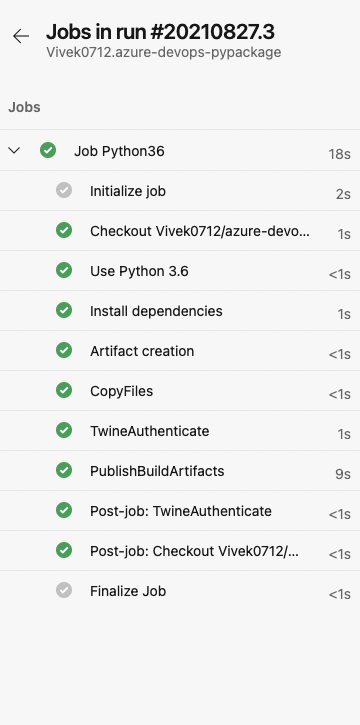
Run the pipeline to build and publish the package.

Use the package in your applications.

**Review**

Review and Run the pipeline.





**Setting up release Pipeline**

Once the pipeline is run, it creates a Dist folder in the code repo which has the artifact.

To set up a release pipeline to release the artifact we had just created, go to Azure DevOps > Pipelines > Release Pipelines > Create New Release Pipeline.

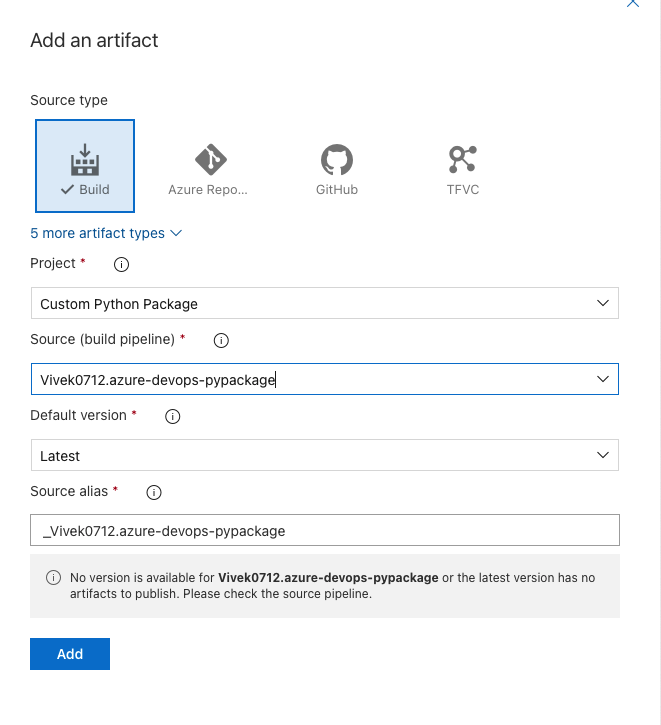
Start with an empty template.

A release pipeline takes an artifact and runs through several stages to publish the artifact as per the pipeline configuration.

To publish the python package, we are going to select the Build Pipeline artifact and create a stage with one jobs with three tasks.

**Selecting the Artifact source**

Select the pipeline which we had created before as the source of artifact.

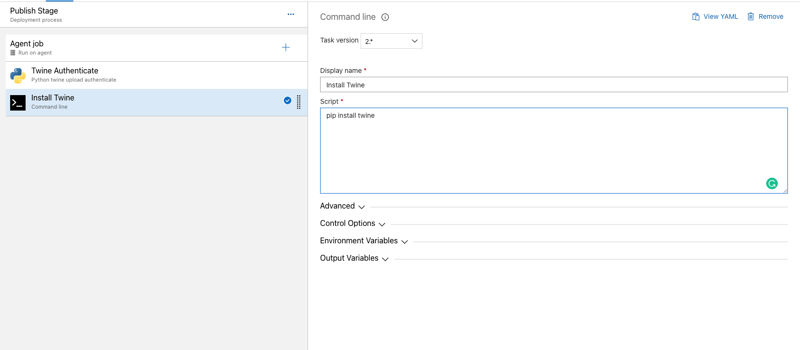


**Pipeline Stage**

Click on the stage and give a suitable name for the same. Click on Jobs and then we are going to create three tasks here. Add Agent Job.

**Task 1: Twine Authenticate**

Search for Twine Authenticate on the right side and add the same.

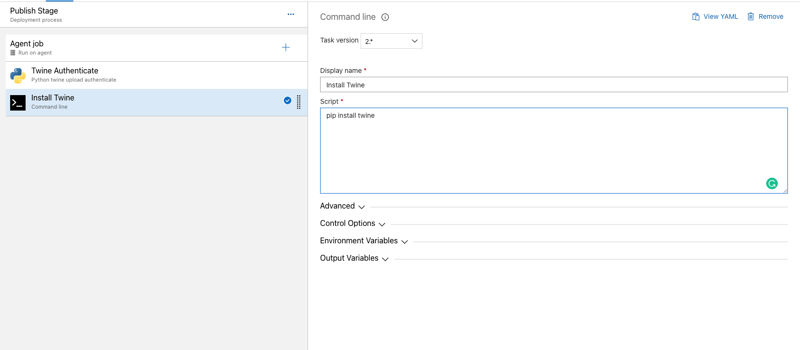


**Task 2: Install Twine**

Search for Command line and add it. Name the command script as Install Twine and add the following to the script

bash

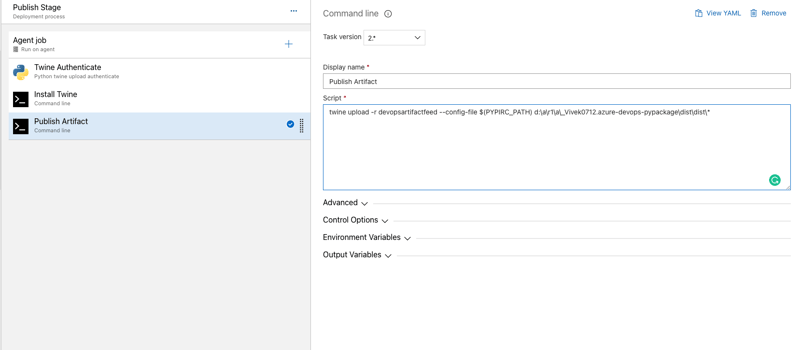
pip install twine



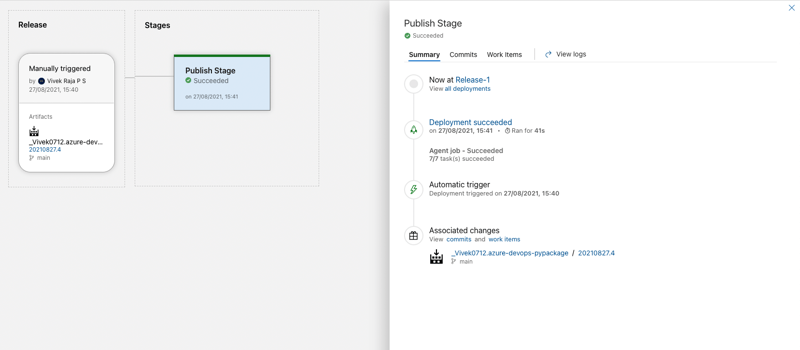
**Task 3:**

Search for Command line and add it. Name the command script as Publish Artifact and add the following to the script. Replace with the name of the artifact feed which we had created first and with the pipeline name

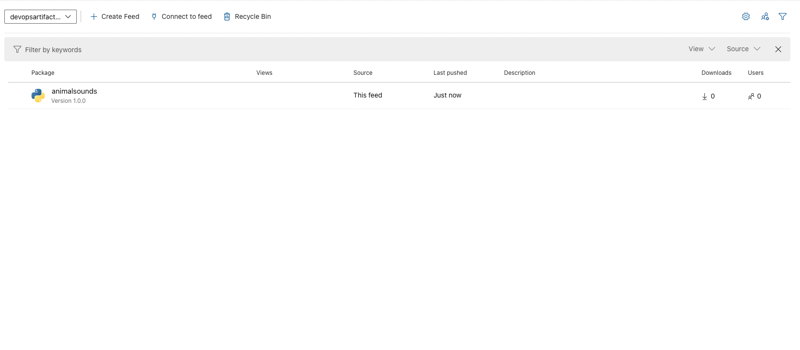
twine upload -r <Artifact Feed Name> --config-file $(PYPIRC\_PATH) d:\a\r1\a\<Build Pipeline Name>\dist\dist\\*



Once your release pipeline is ready, save it and Run the release pipeline. It may take a couple of minutes to run.



Once the release pipeline is successfully run, we can see the published artifact in the Artifact.



Using the Python Package

Now that our python package is published, we will see how to use it.

Create a personal token

On the right top corner, click your account and create a personal token. This token is assured that the package is only accessed by authorised personals since it is private to the organisation.

**Connect to a feed**

Once the personal token is generated, go to the Artifact and click on the our python package artifact. Click connect to feed and choose either Twine or PyPI as publisher for the same. Follow the steps given there.

**twine**

Sign in to your Azure DevOps organization, and then navigate to your project.

Select Artifacts, select your feed from the dropdown menu, and then select Connect to feed.

Select twine under the Python section.

If this is your first time using Azure Artifacts with twine, select Get the tools and follow the steps to install the prerequisites.

Add a pypirc file to your home directory and paste the provided snippet. Your file should look like this:

[distutils]

Index-servers =

FEED\_NAME

[FEED\_NAME]

Repository = <https://pkgs.dev.azure.com/ORGANIZATION_NAME/PROJECT_NAME/_packaging/FEED_NAME/pypi/upload/>

**Pip**

Sign in to your Azure DevOps organization, and then navigate to your project.

Select Artifacts, select your feed from the dropdown menu, and then select Connect to feed.

Select pip under the Python section.

If this is your first time using Azure Artifacts with pip, select Get the tools and follow the steps to install the prerequisites.

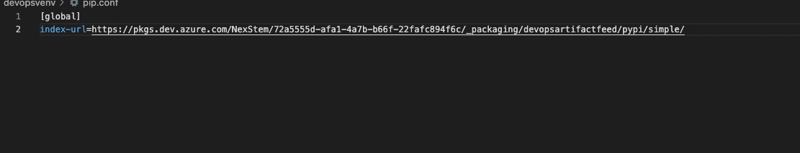
Create a virtual environment.

python -m venv /path/to/new/virtual/environment

**Install the python package**

Once the steps are followed, with access token and the feed is connected, you will be able to install and access the python package.

Add a pip.ini file (Windows) or pip.conf file (Mac/Linux) to your virtual environment and paste the provided snippet into it. Your file should resemble the following:



[global]

index-url=https://pkgs.dev.azure.com/ORGANIZATION\_NAME/PROJECT\_NAME/\_packaging/FEED\_NAME/pypi/simple/

**Publish packages**

Run this command in your project directory to create source and wheel distributions:

python setup.py sdist bdist\_wheel

Run the following command to publish your package. Use the -r FEED\_NAME flag to ensure your private packages are not accidentally published to PyPI.

twine upload -r <FEED\_NAME> dist/\*

**Install packages**

Run this command in your project directory to install your packages:

pip install

To install a specific package, run the following command, replacing the placeholder with the package name from your feed.

Command

pip install <PACKAGE\_NAME>