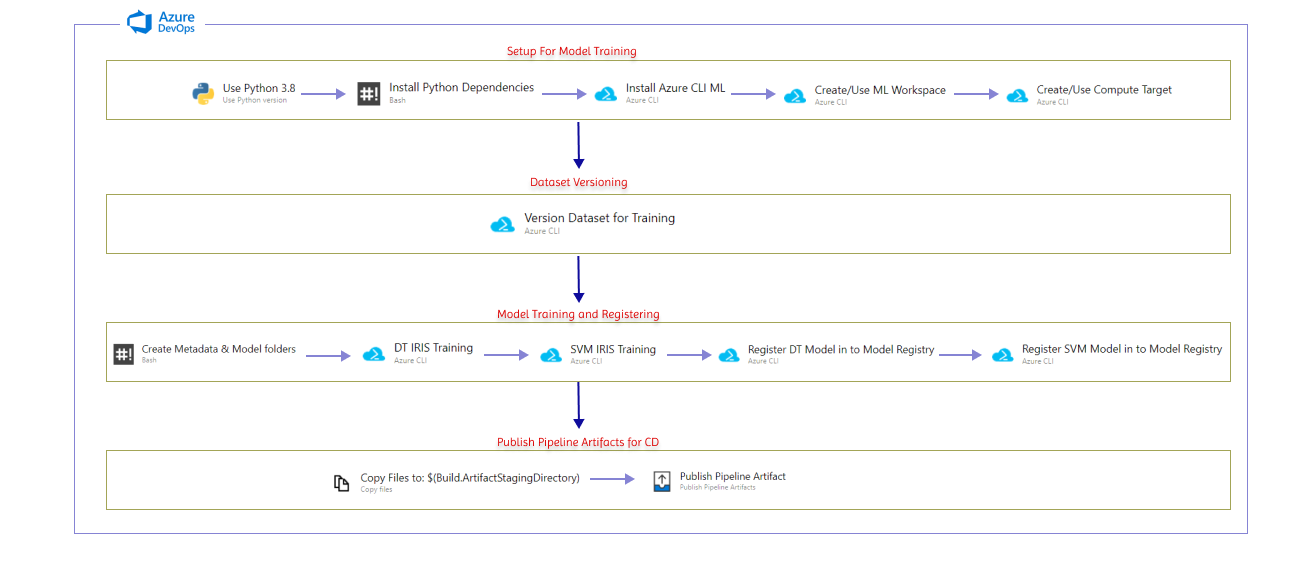
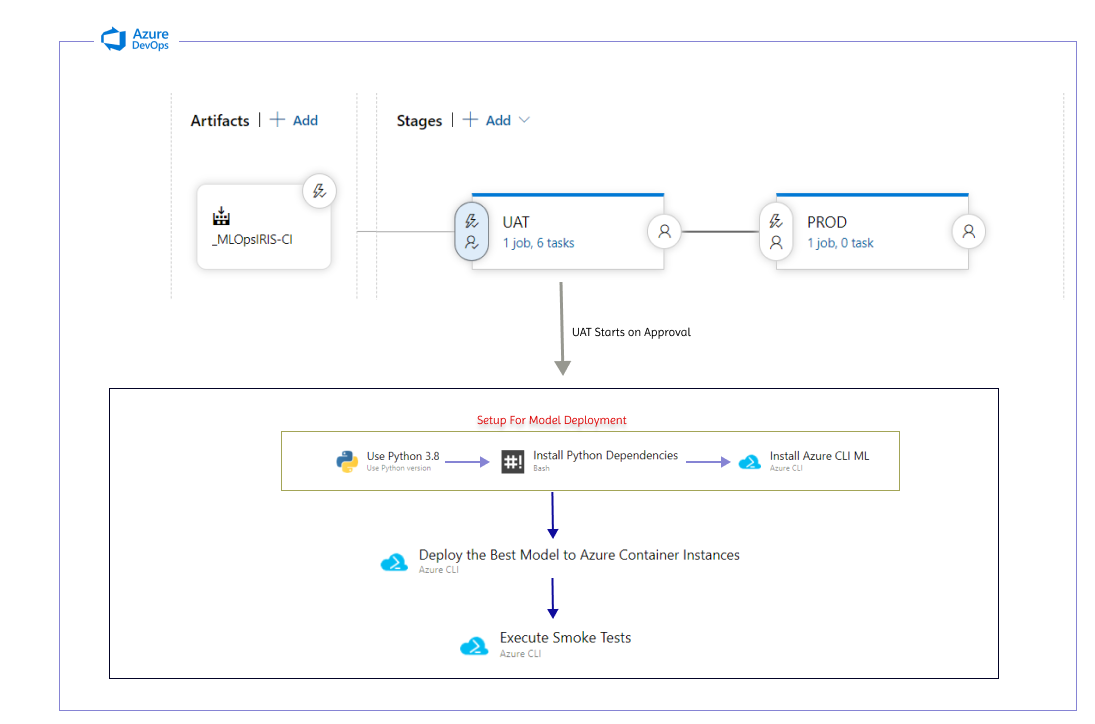
# **The MLOps Architecture**

**CI Pipeline in Azure Devops**

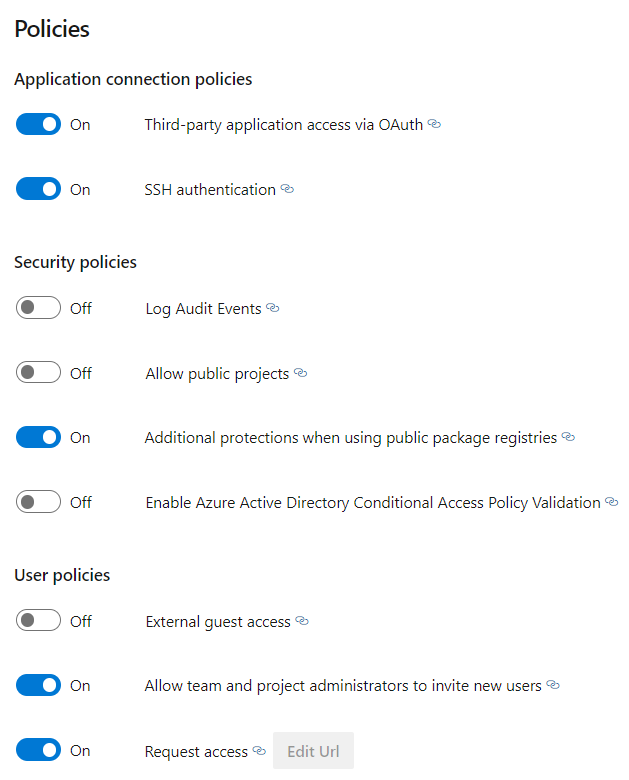


**CD Pipeline in Azure Devops**

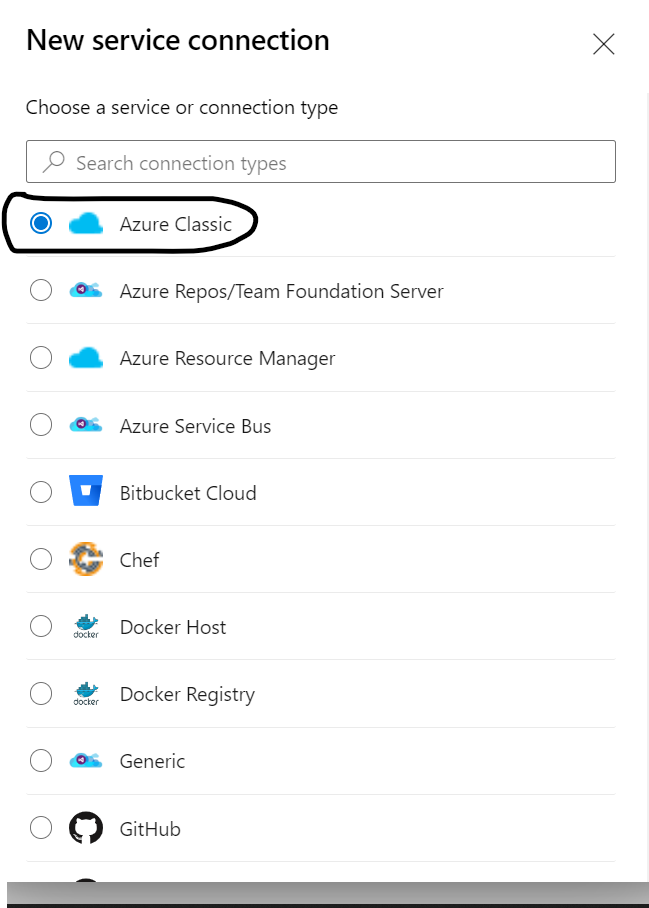
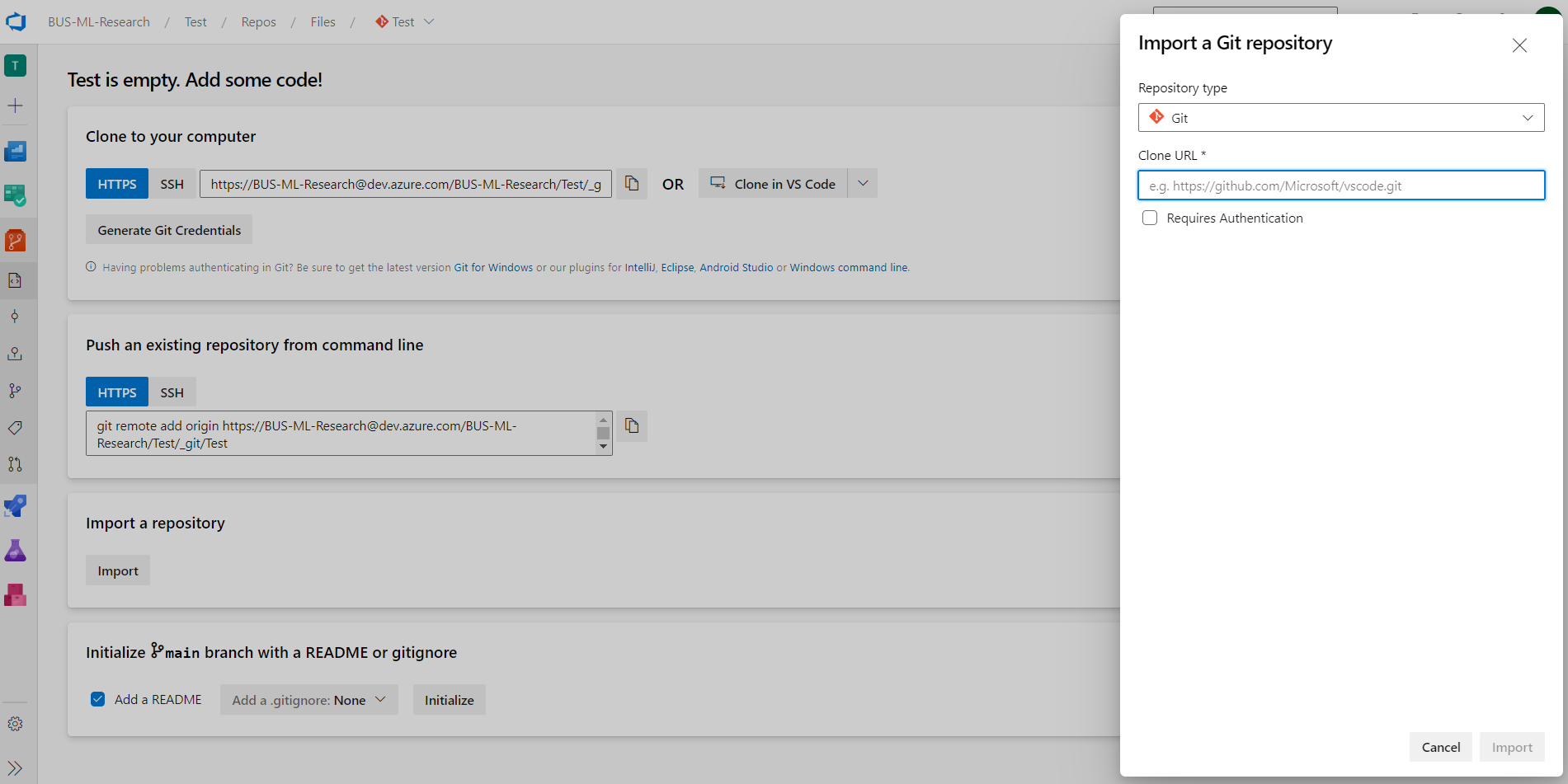


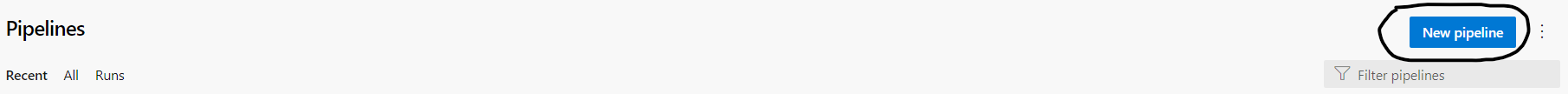
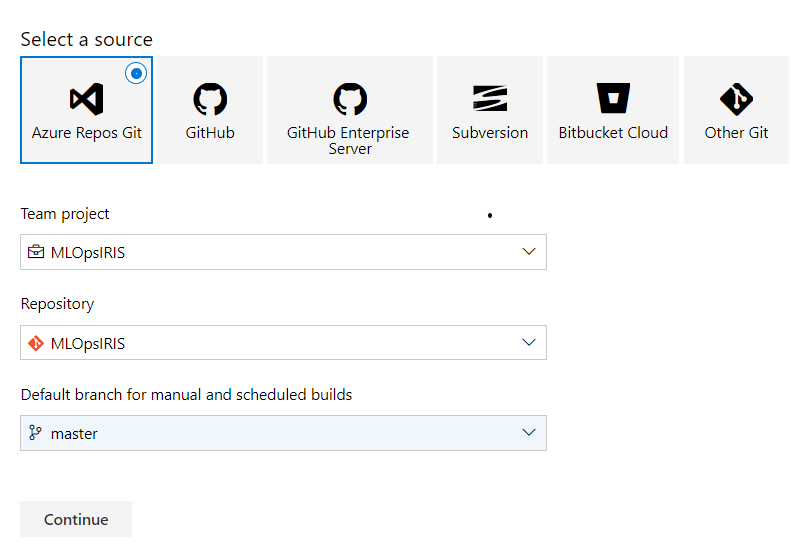
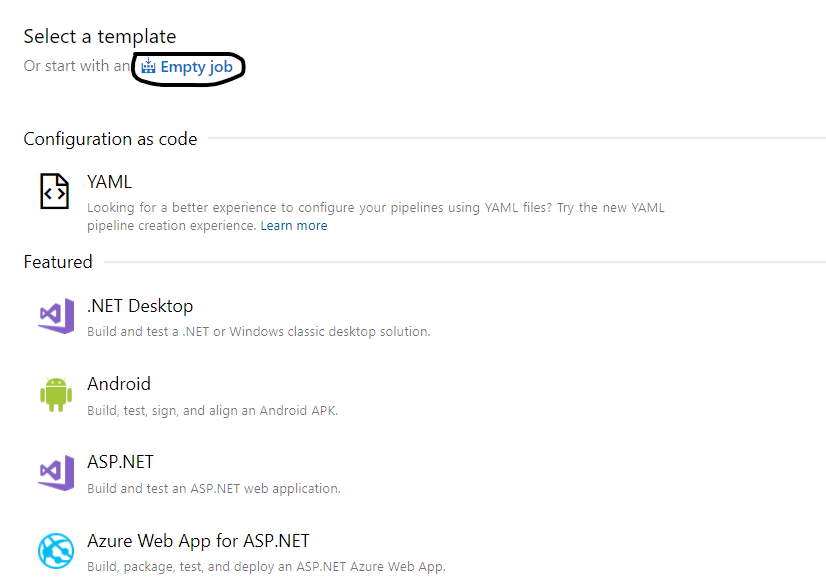
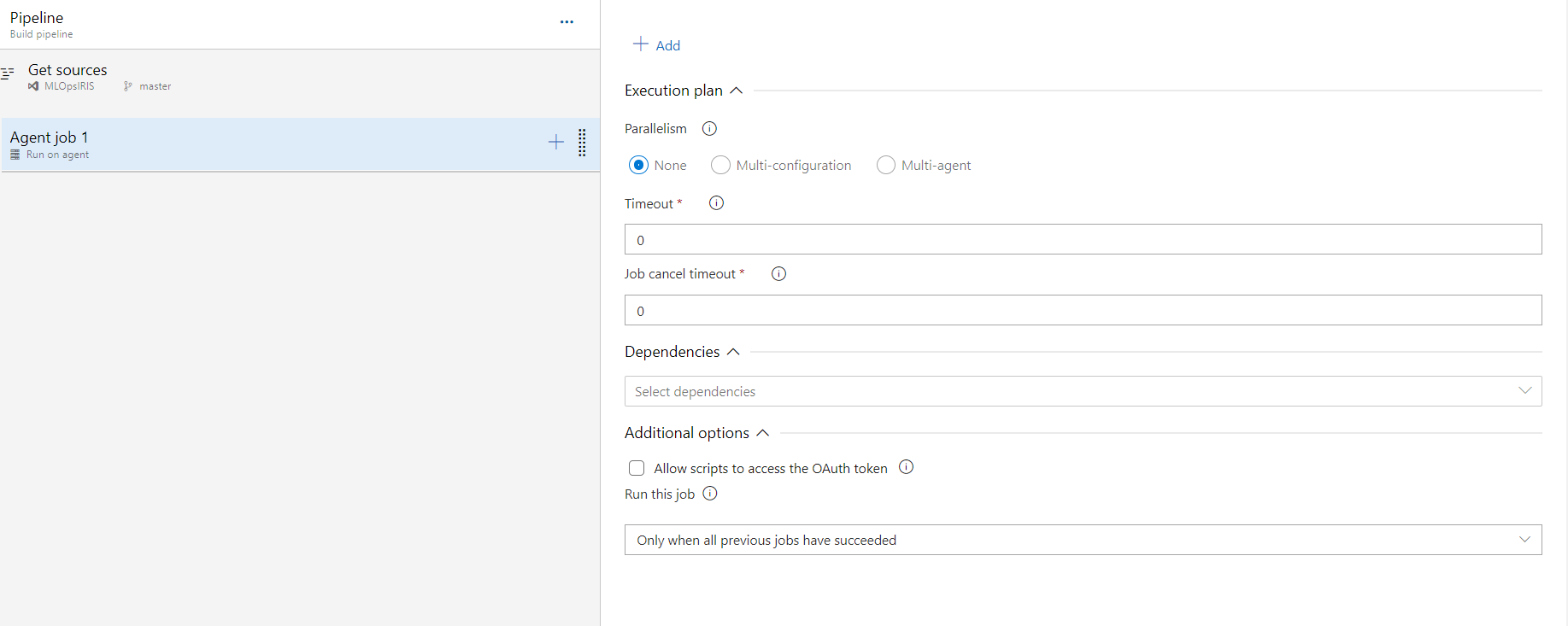
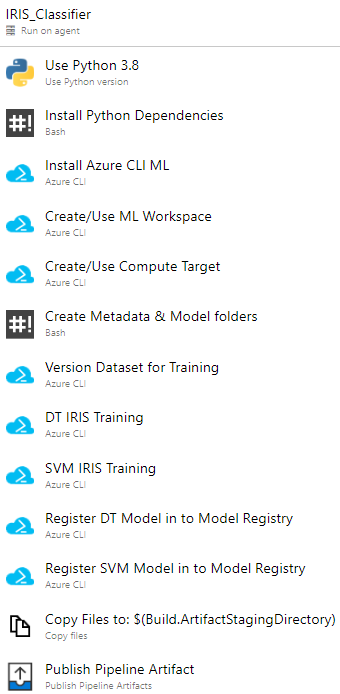
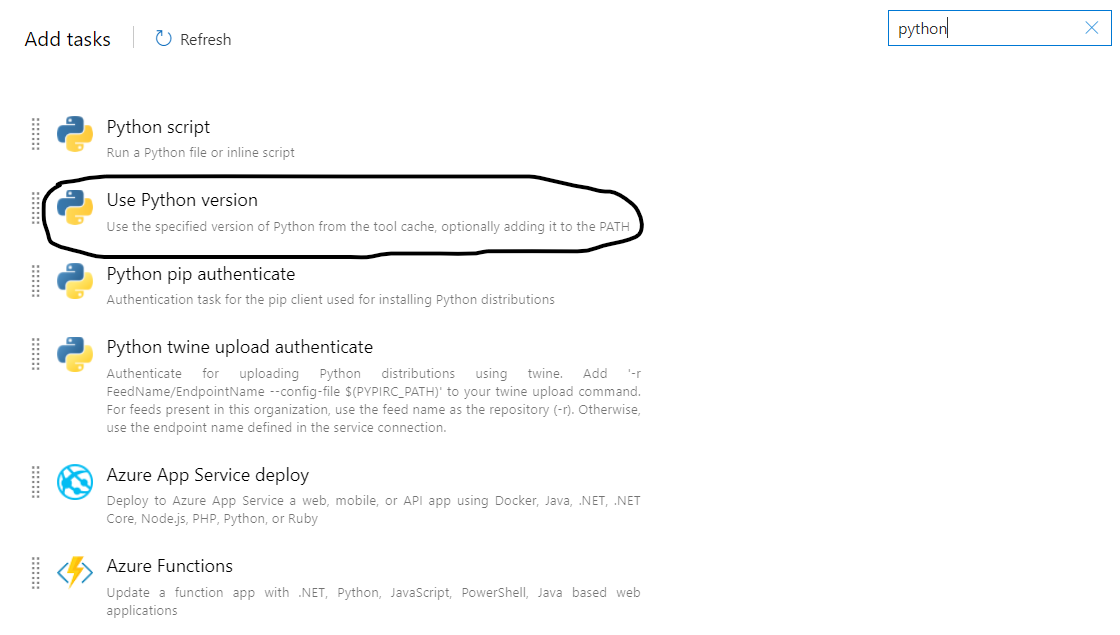
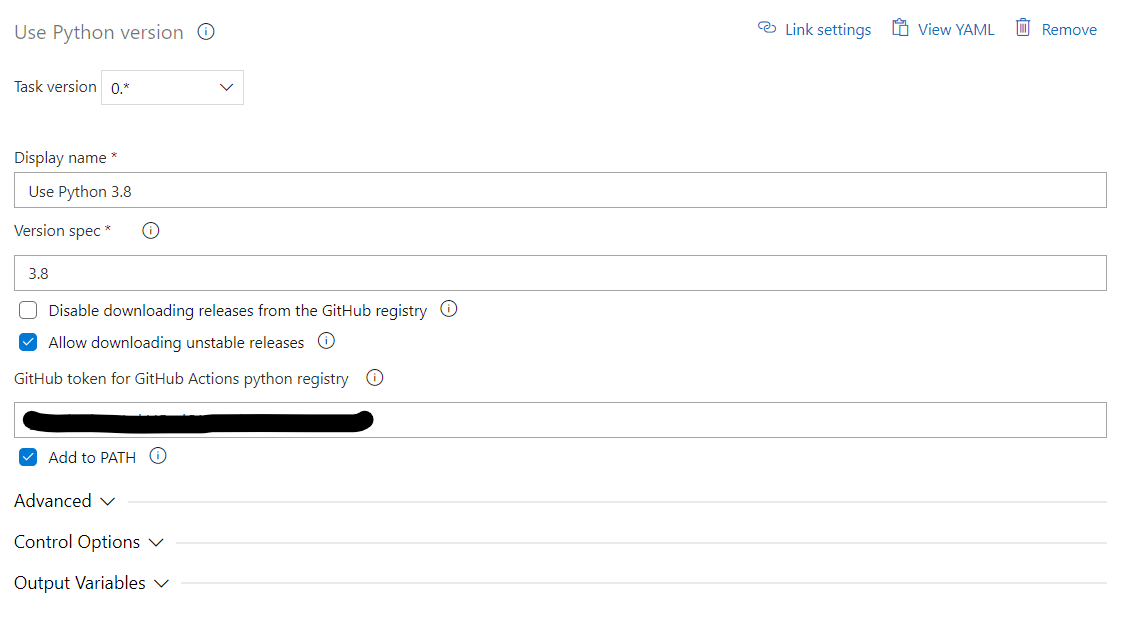
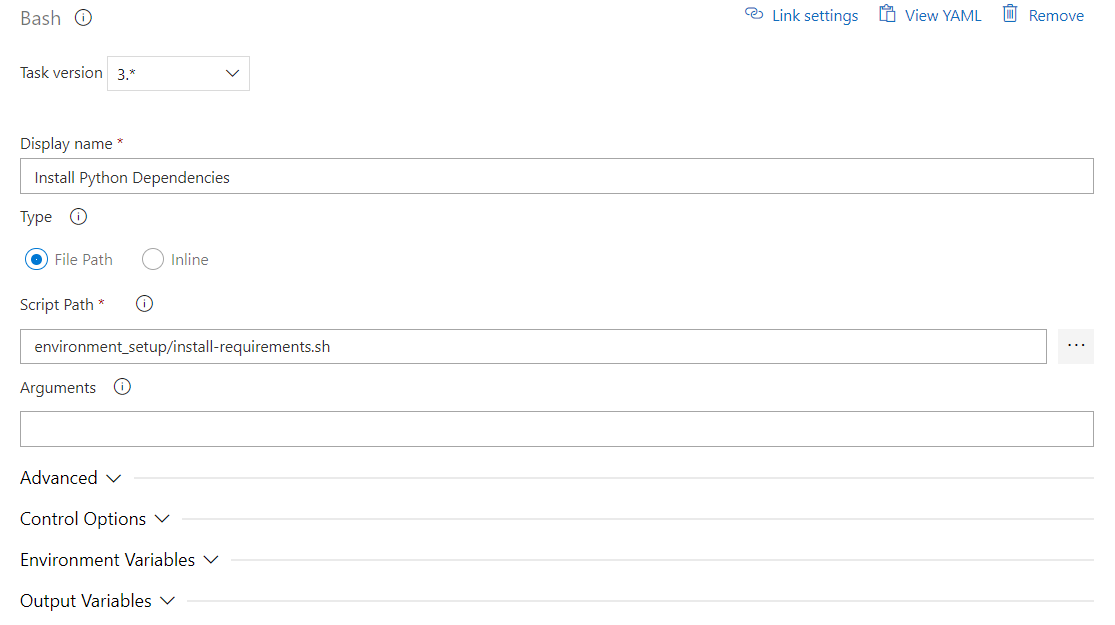
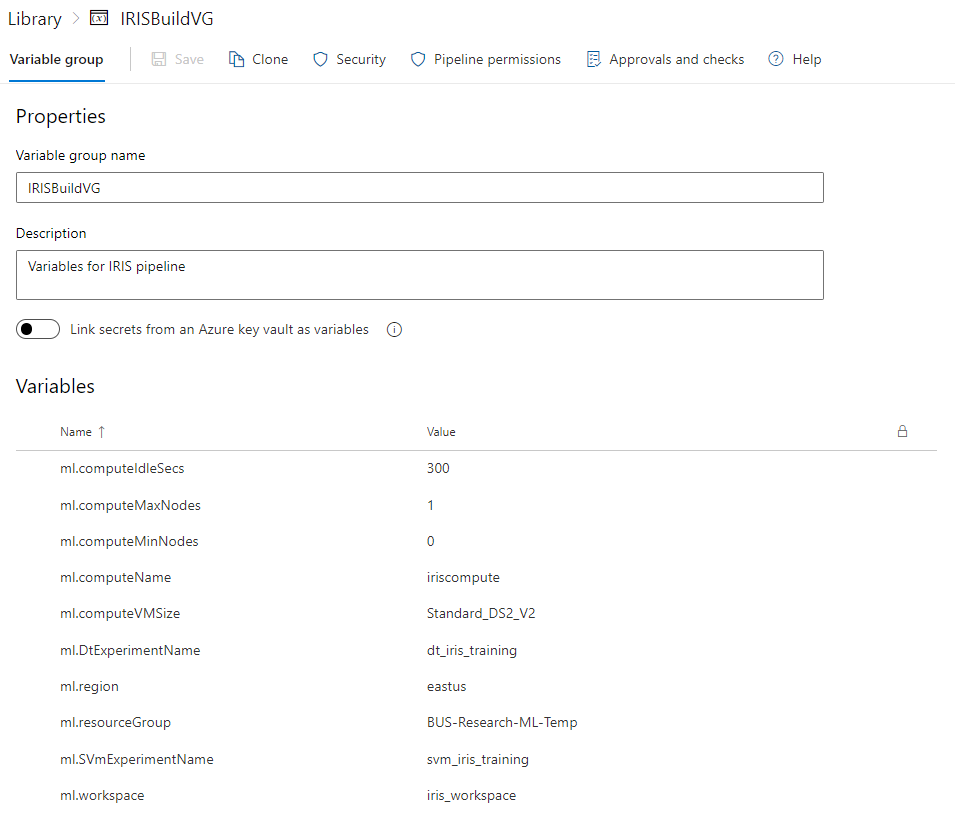
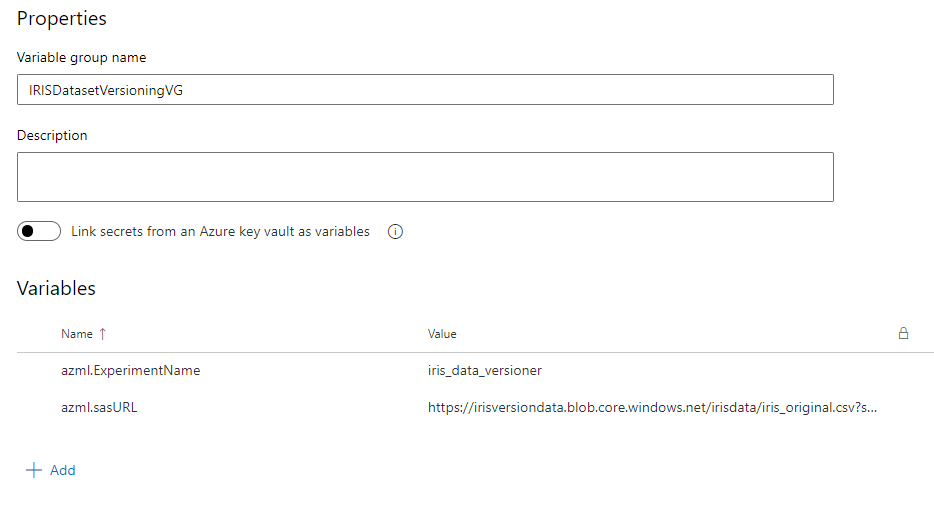
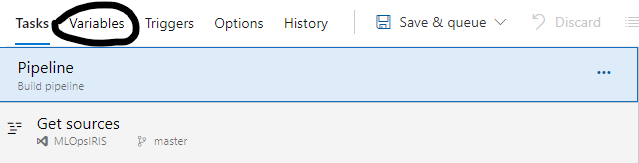
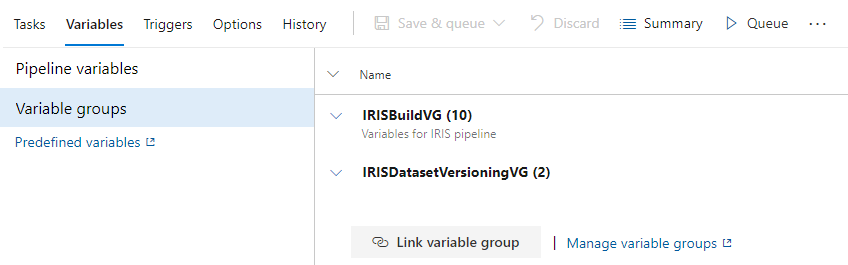
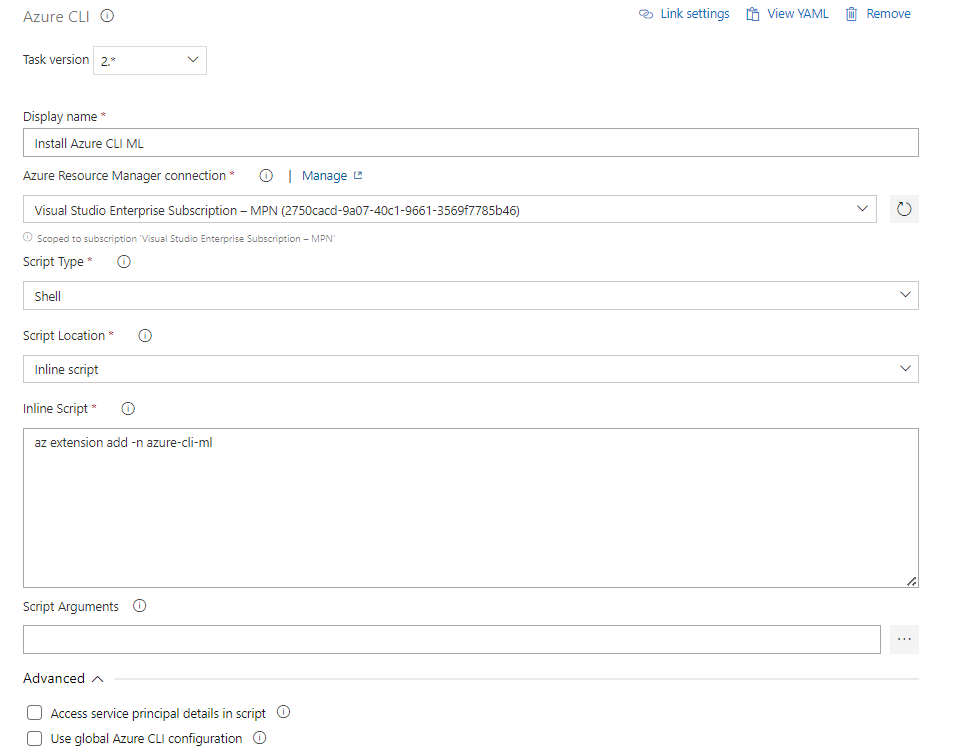
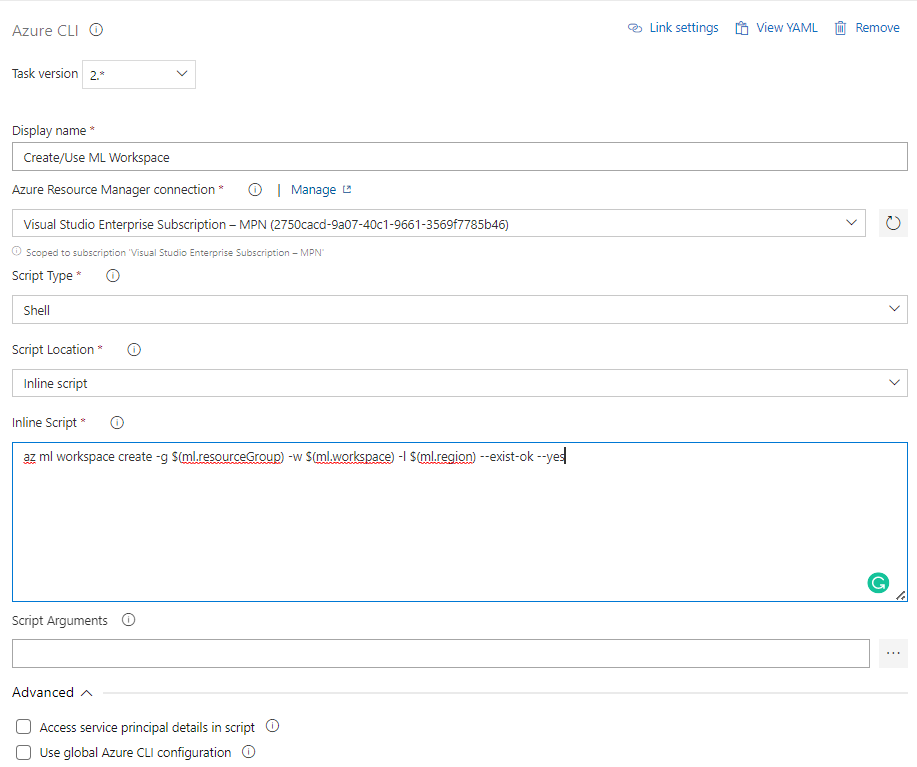
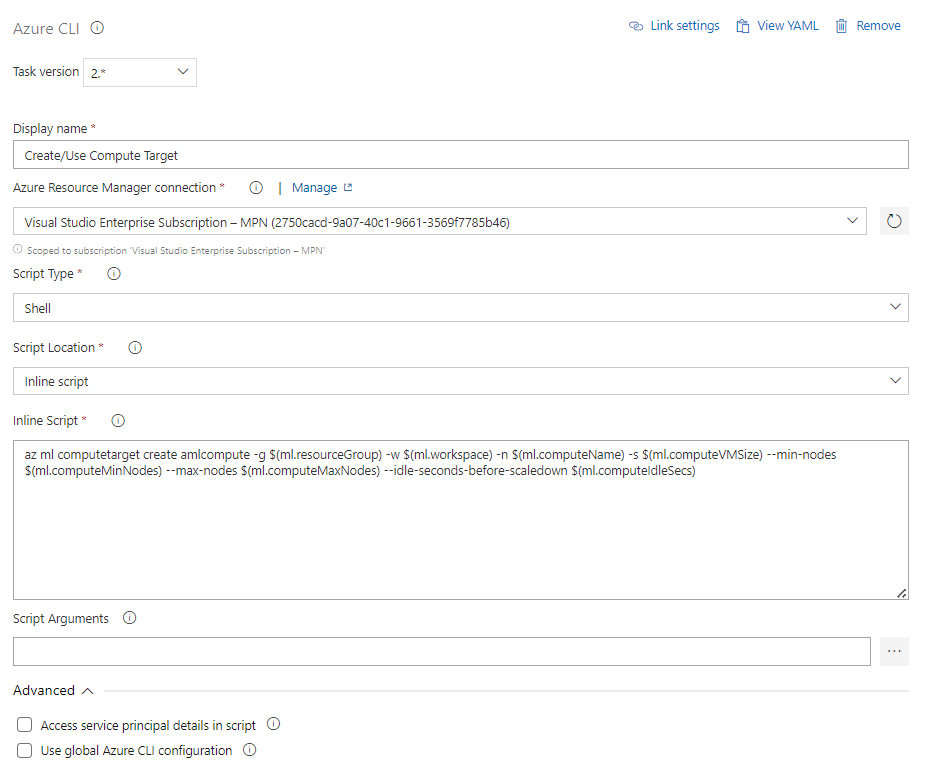
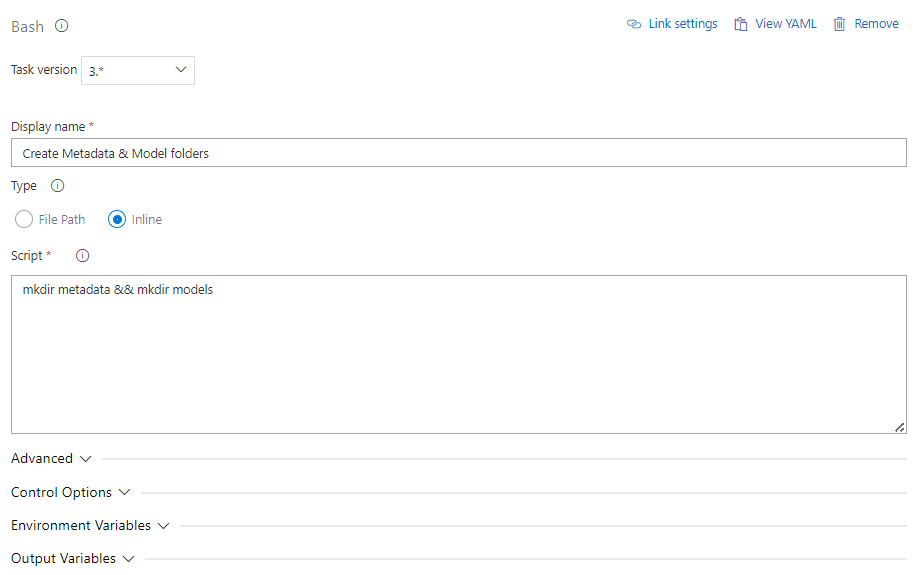
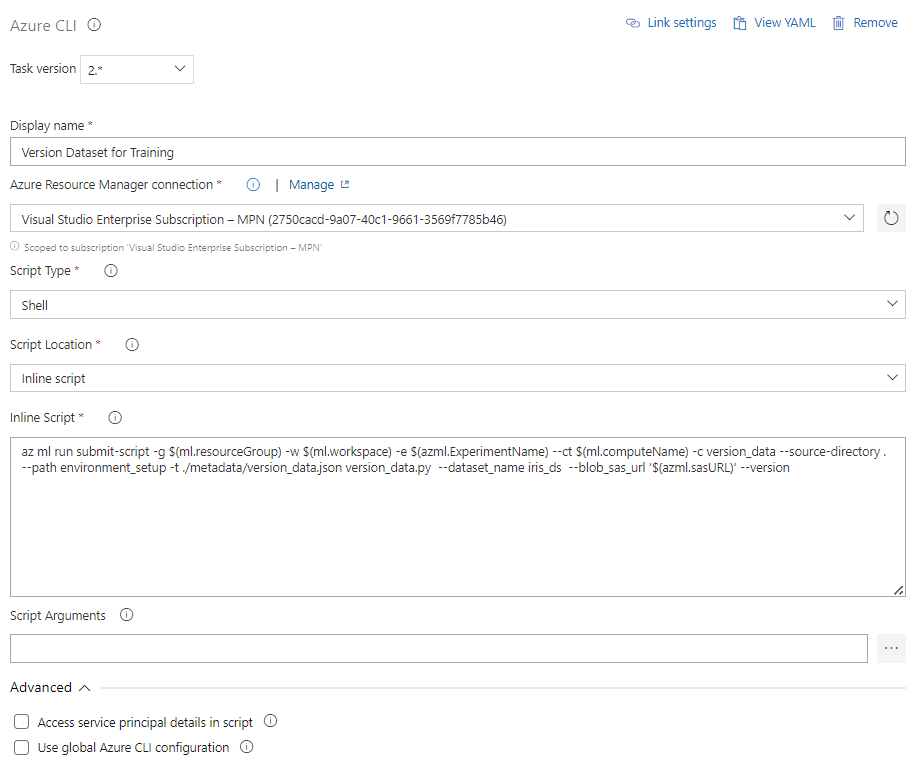
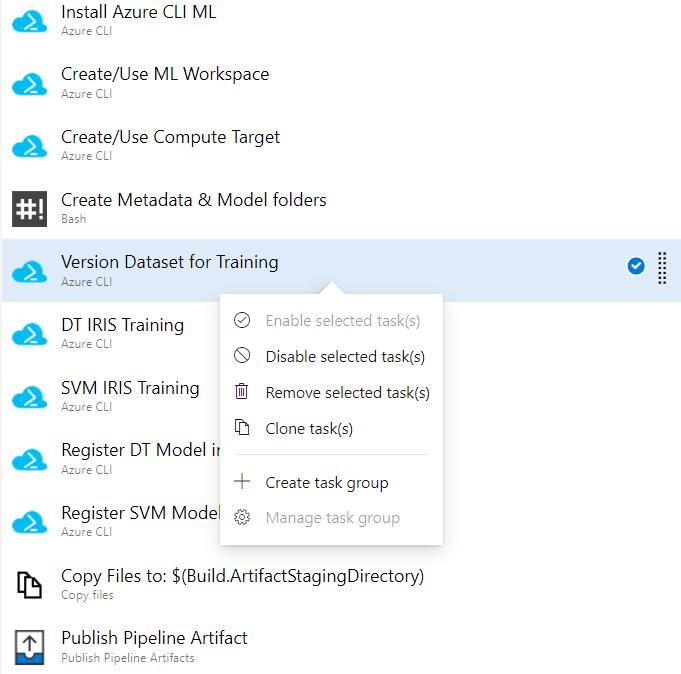
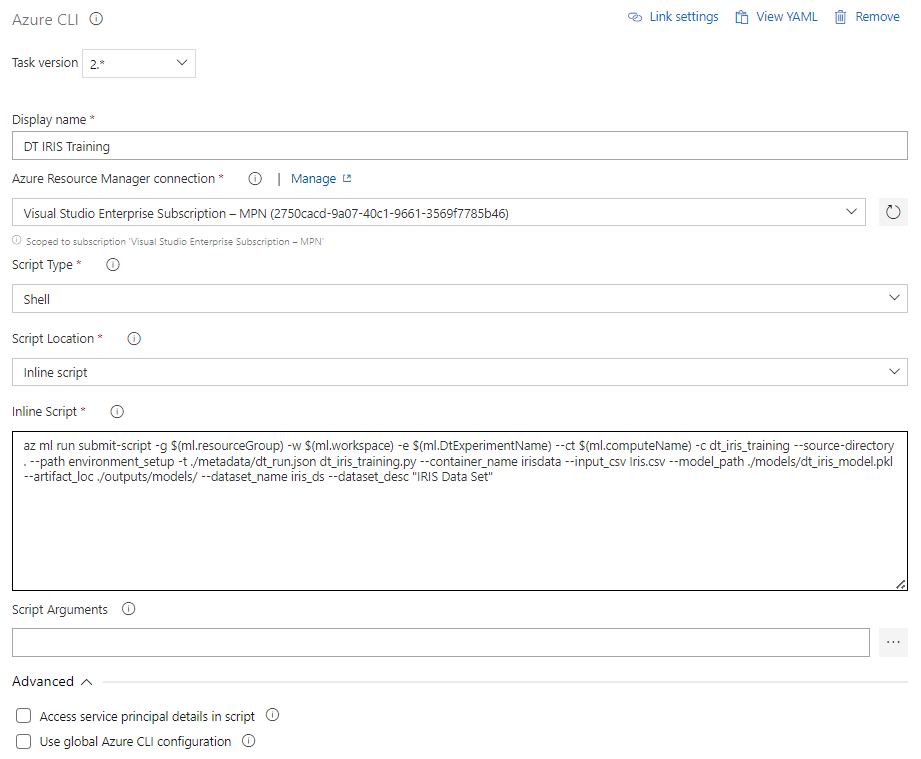
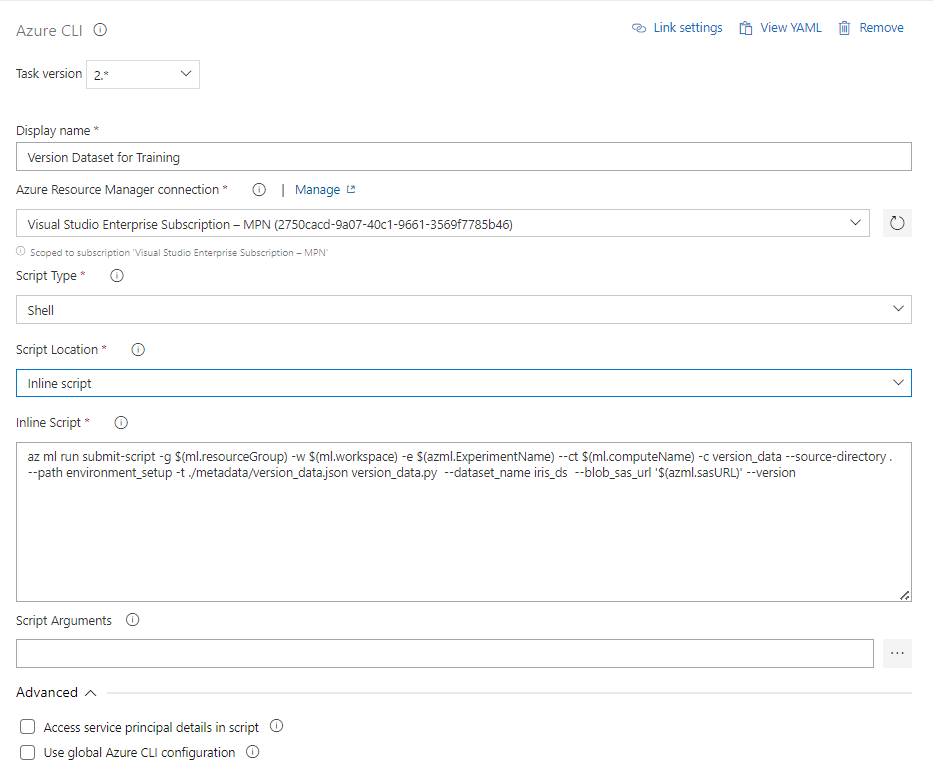
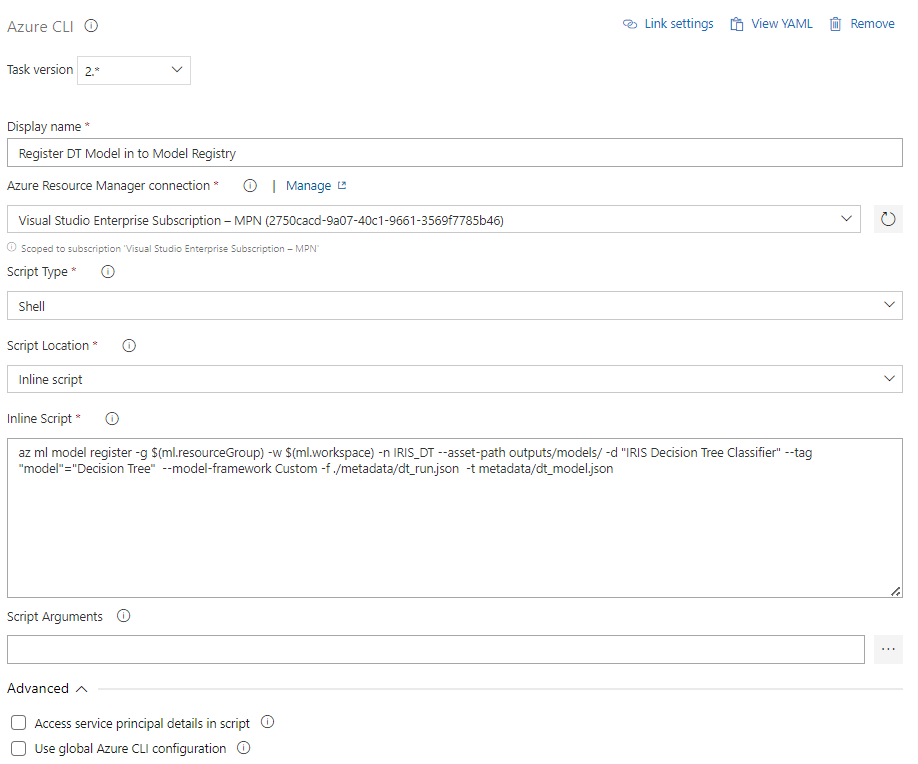
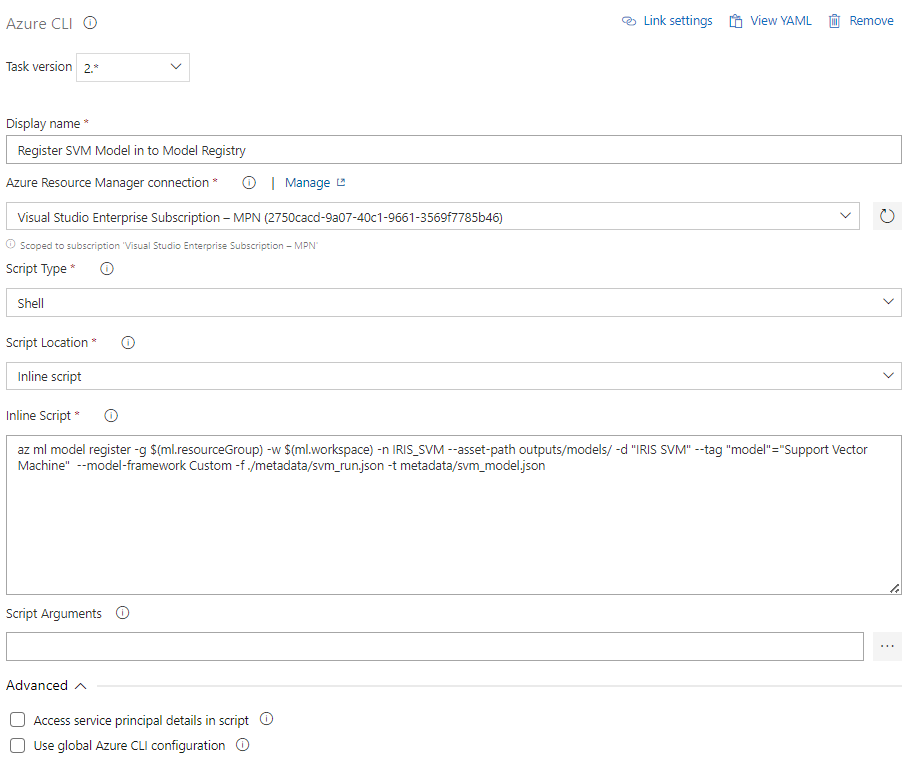
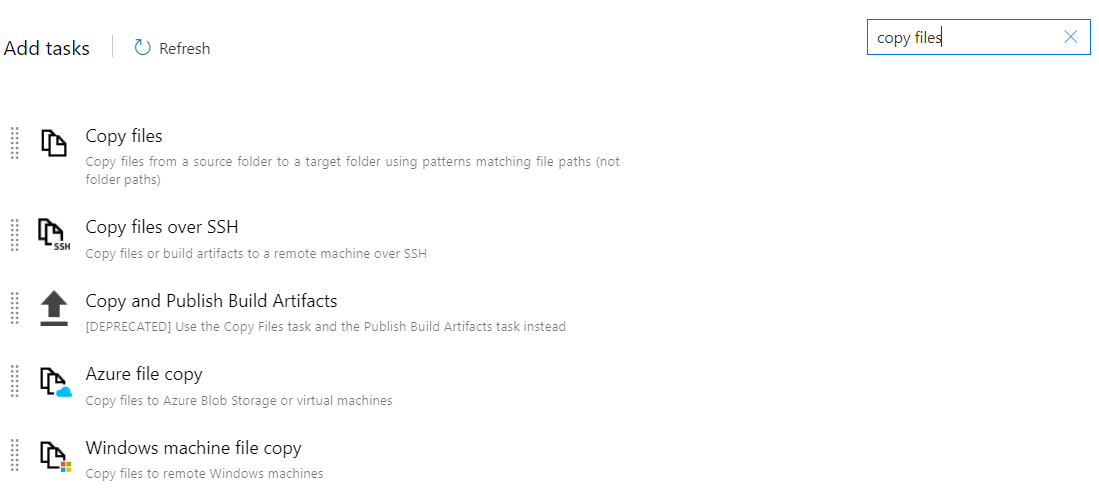
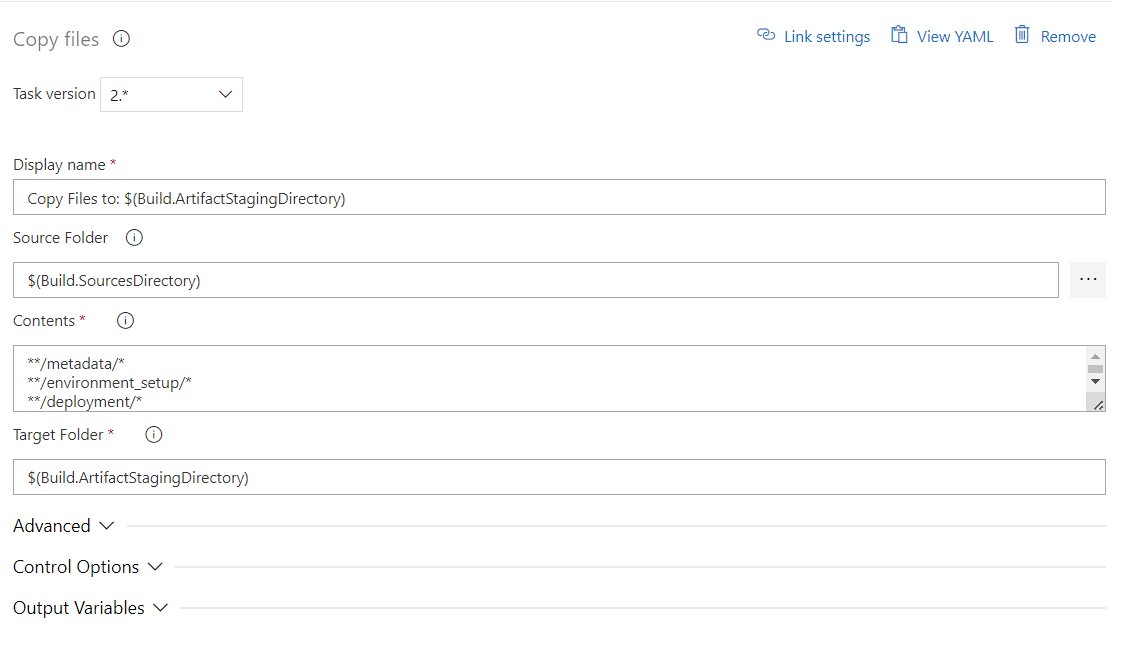
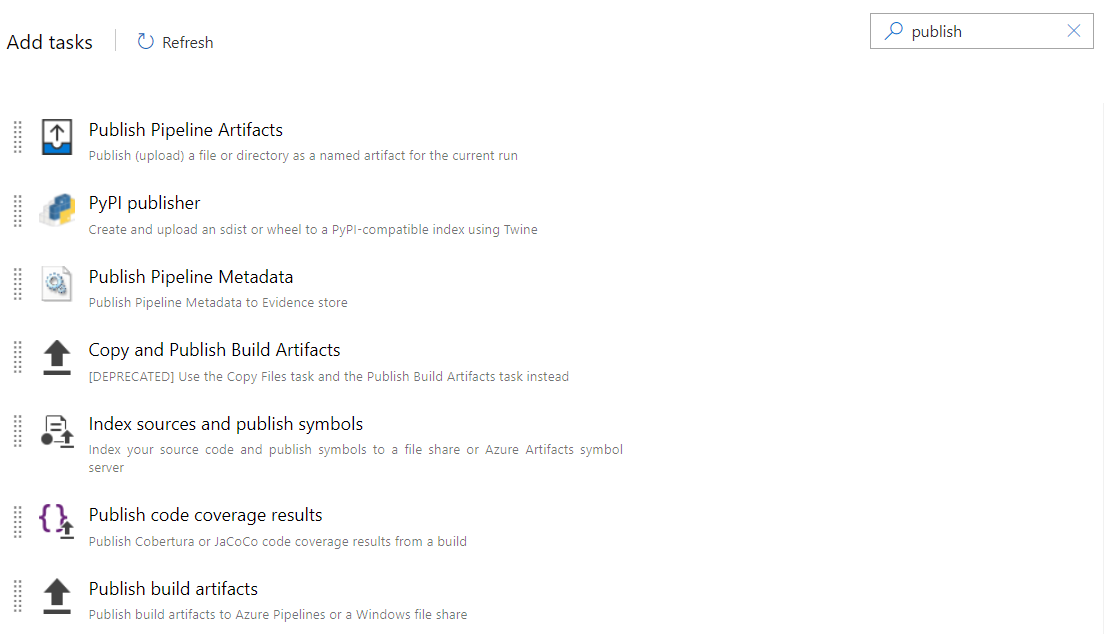
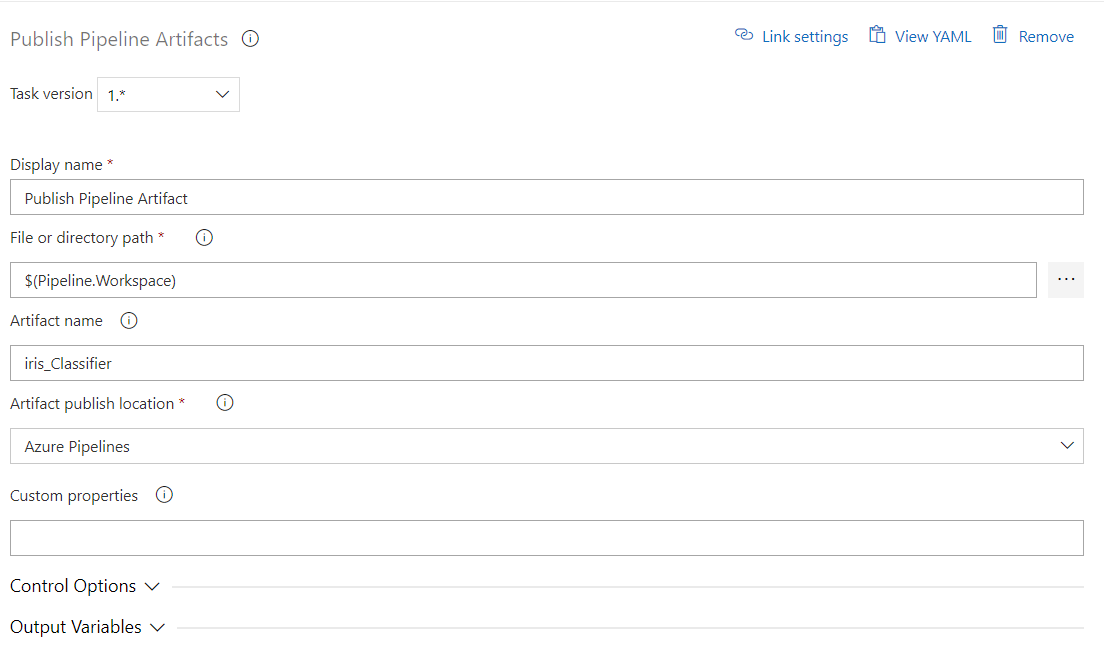
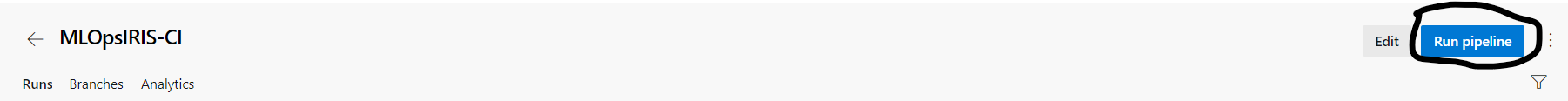
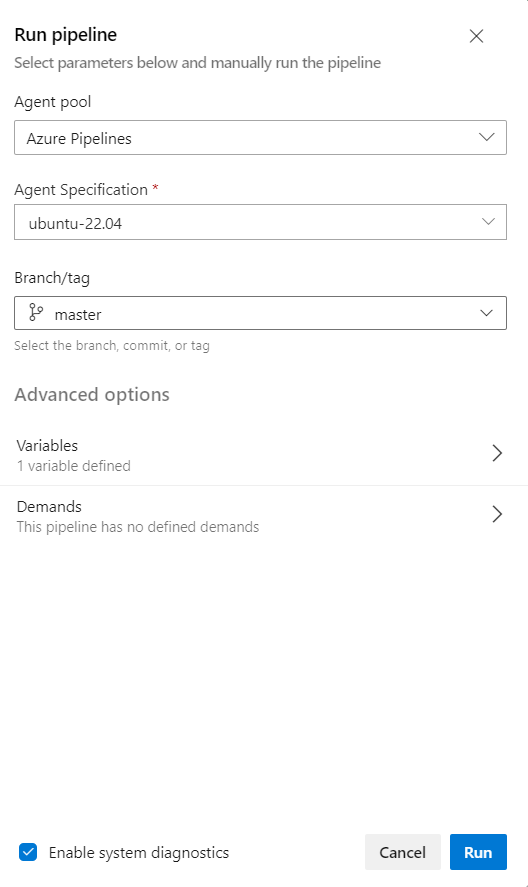
# **Steps to Build the CI Pipeline**

1. Setup Azure Devops

* Create a new Devops Organization and A New Project to start with.
* Enable the Below Policies in the Organization Settings Section. For instance, there might errors launching training jobs if the “Third party application access via OAuth” is Disabled.  
  

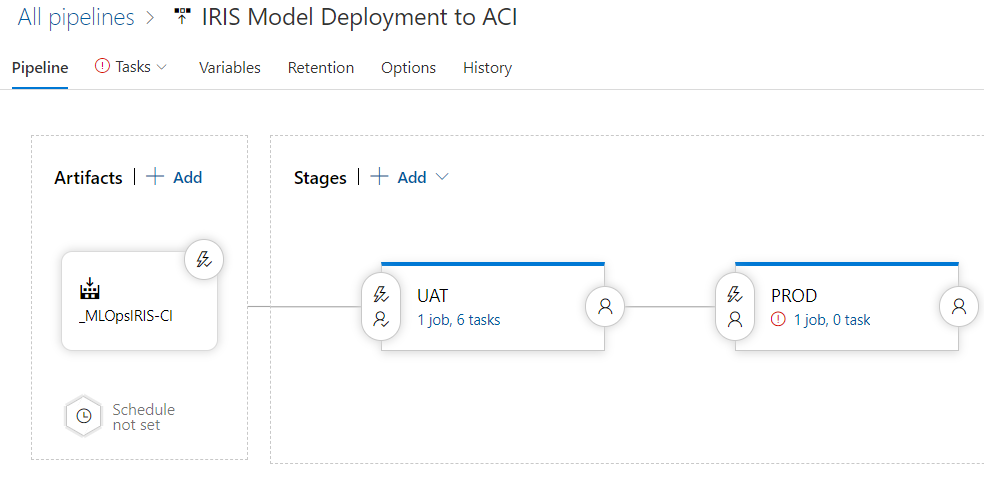
1. Configuring Project Settings

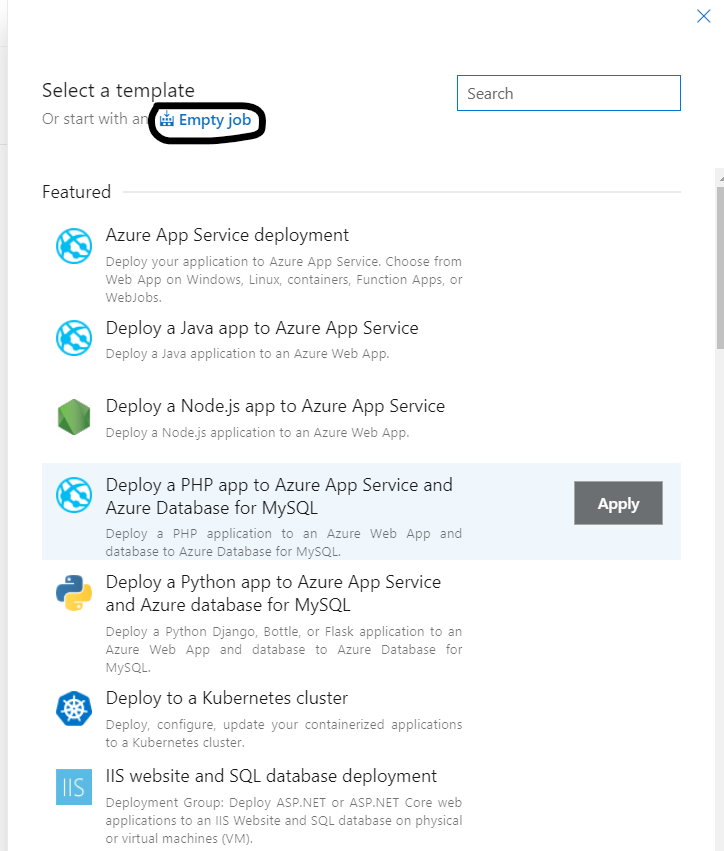
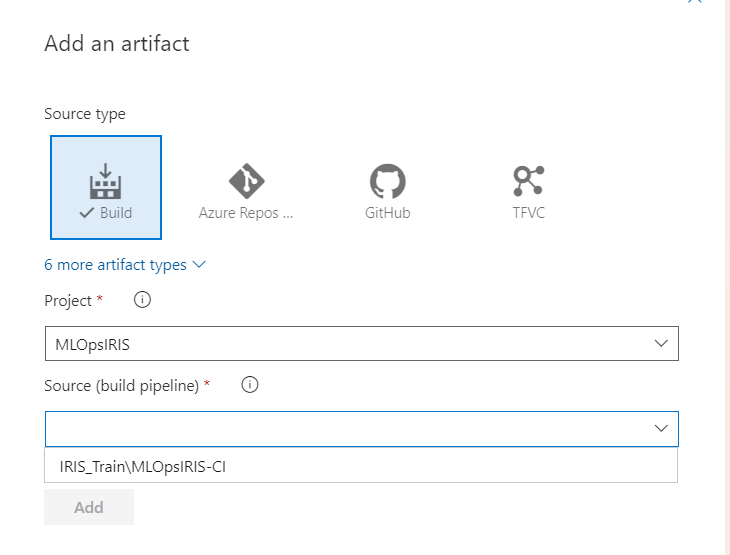
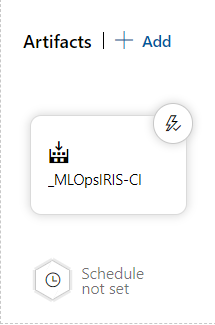
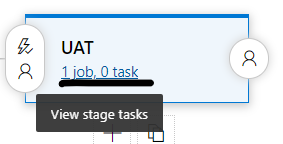
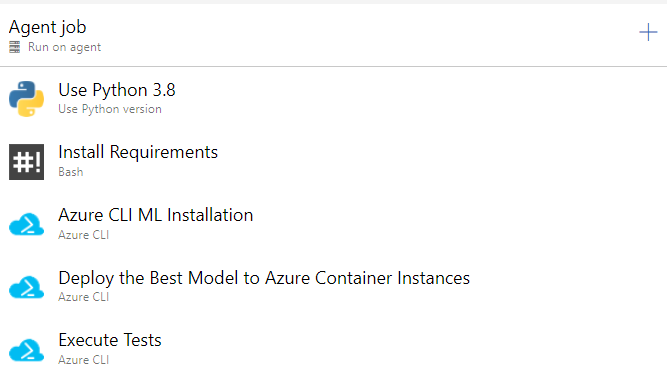
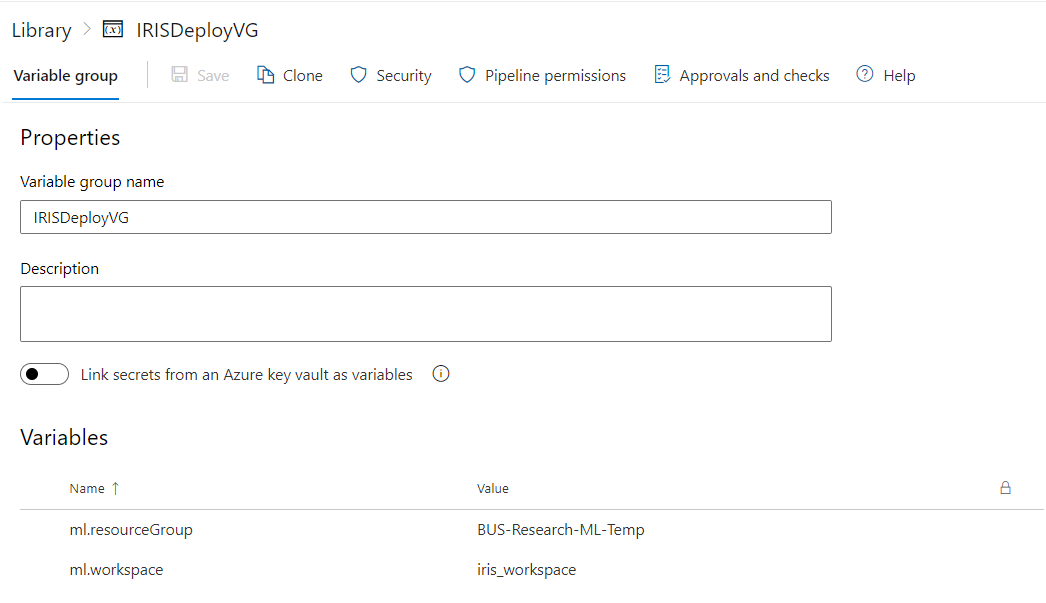
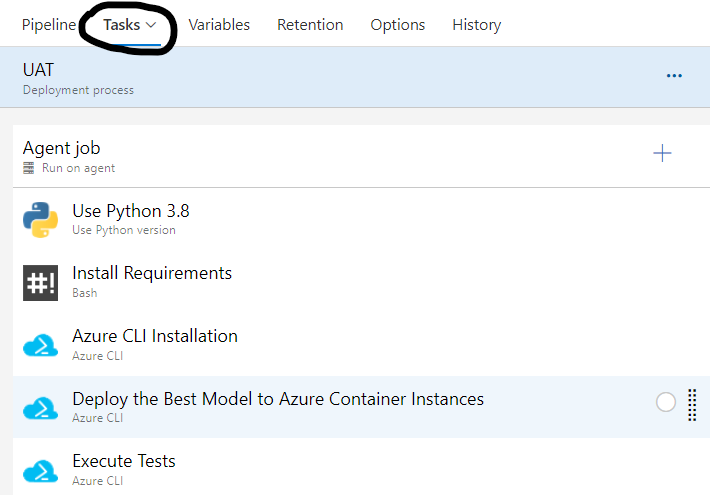
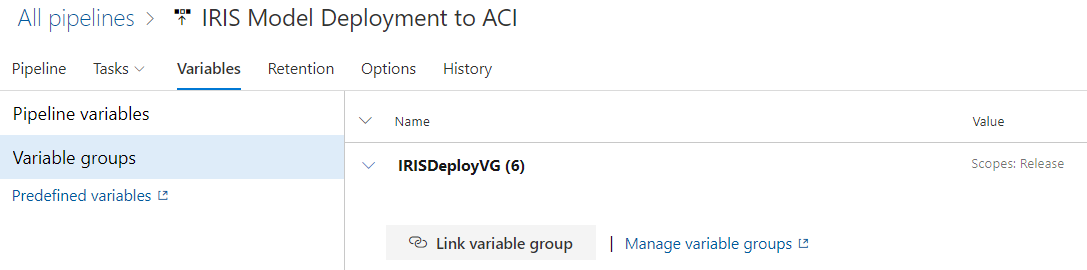
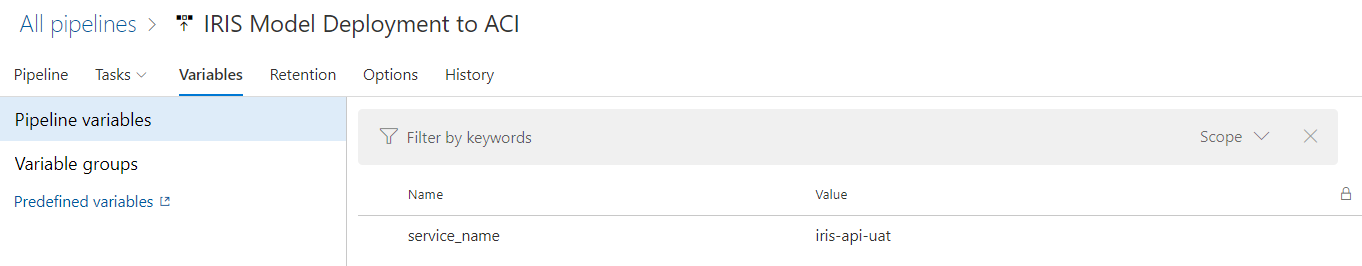
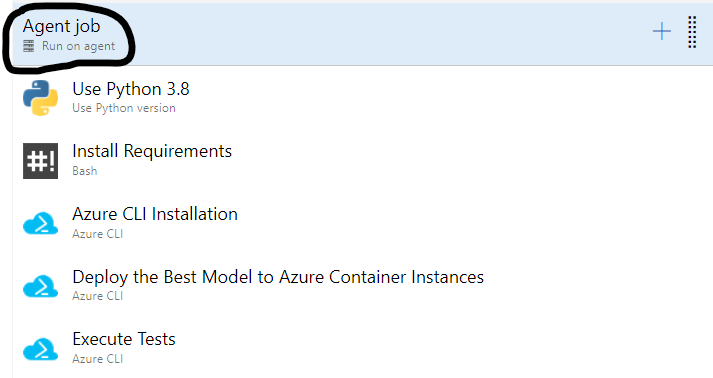
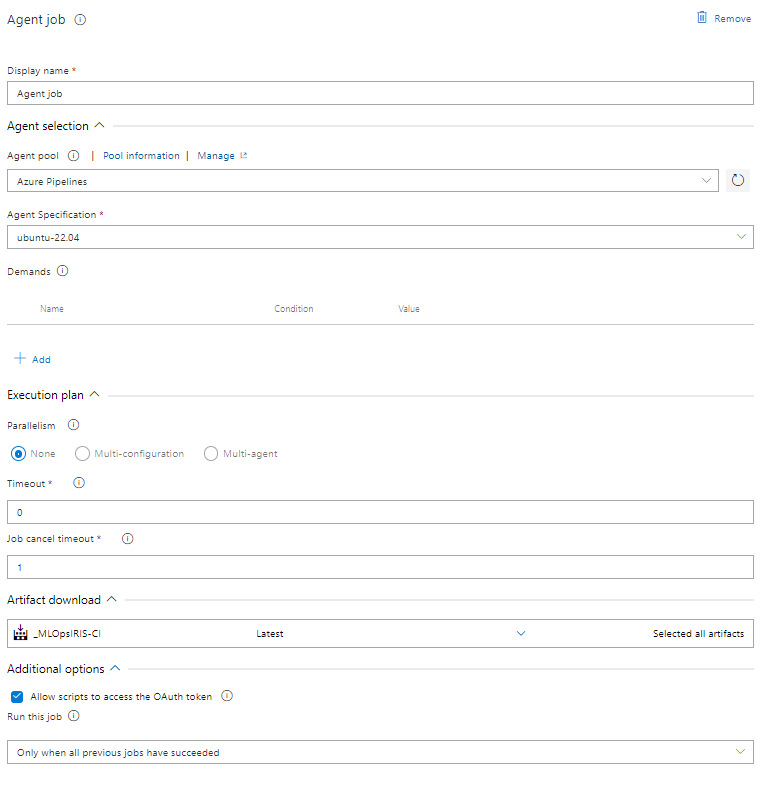
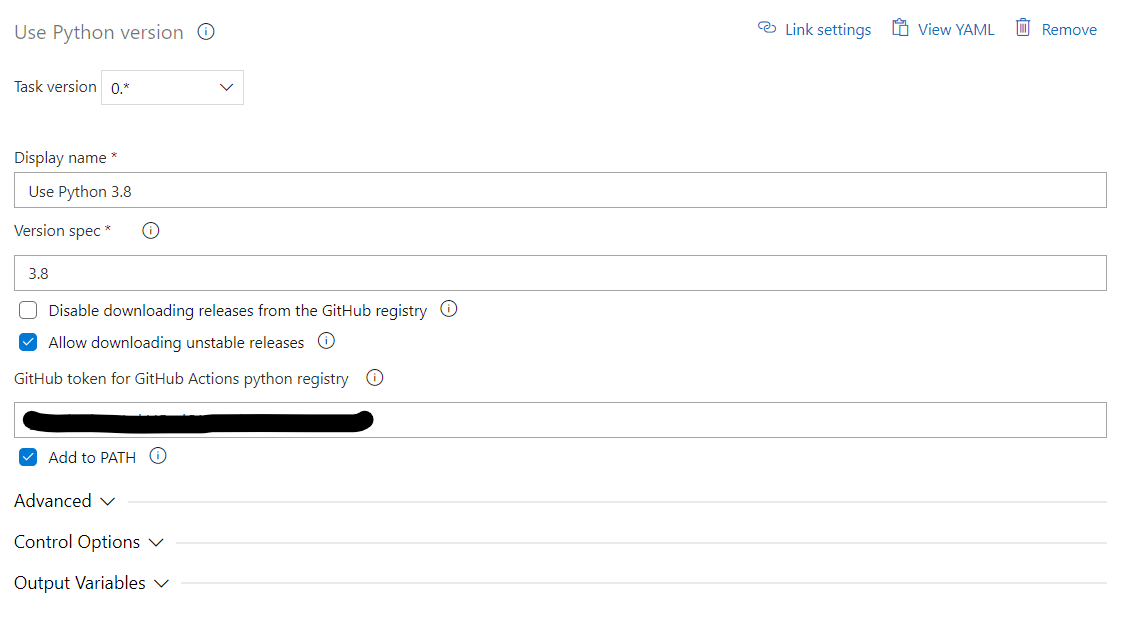
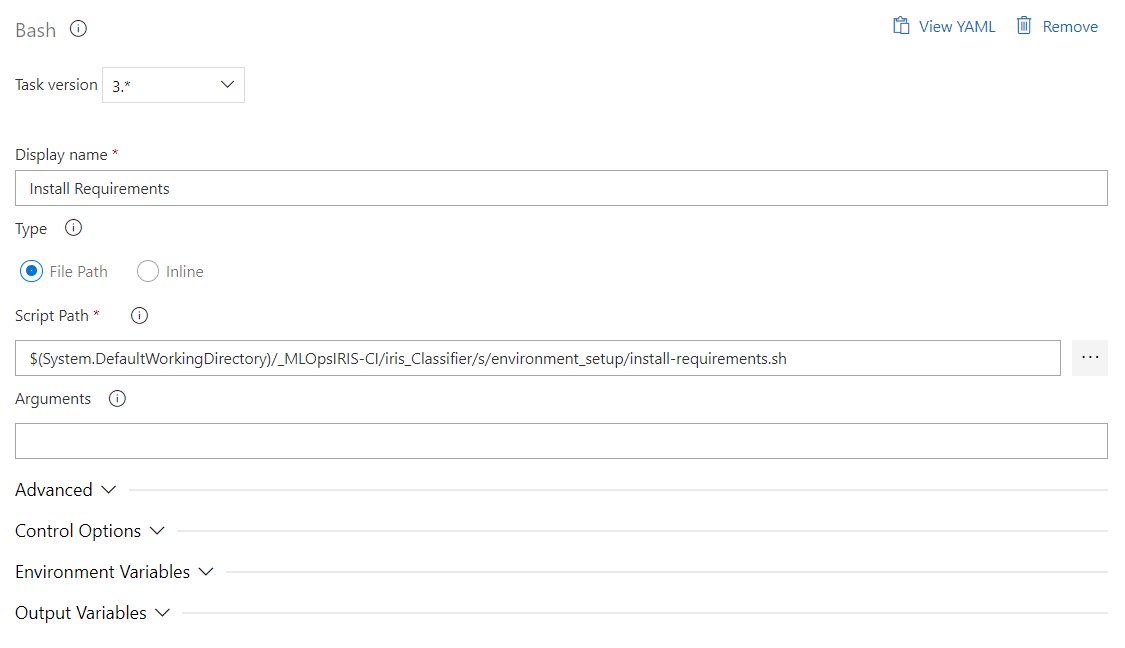
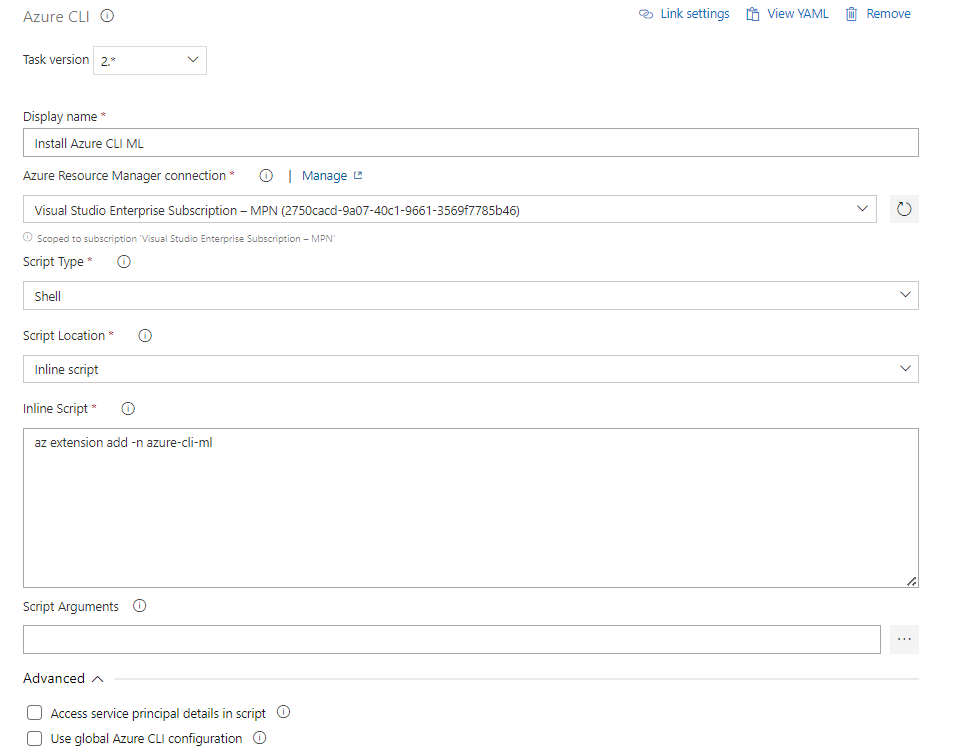
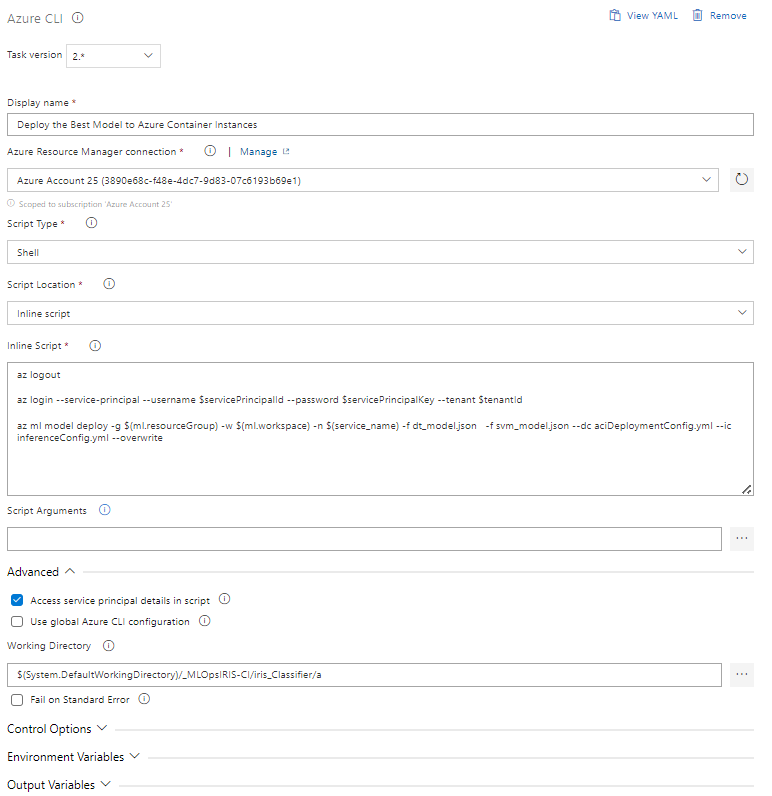
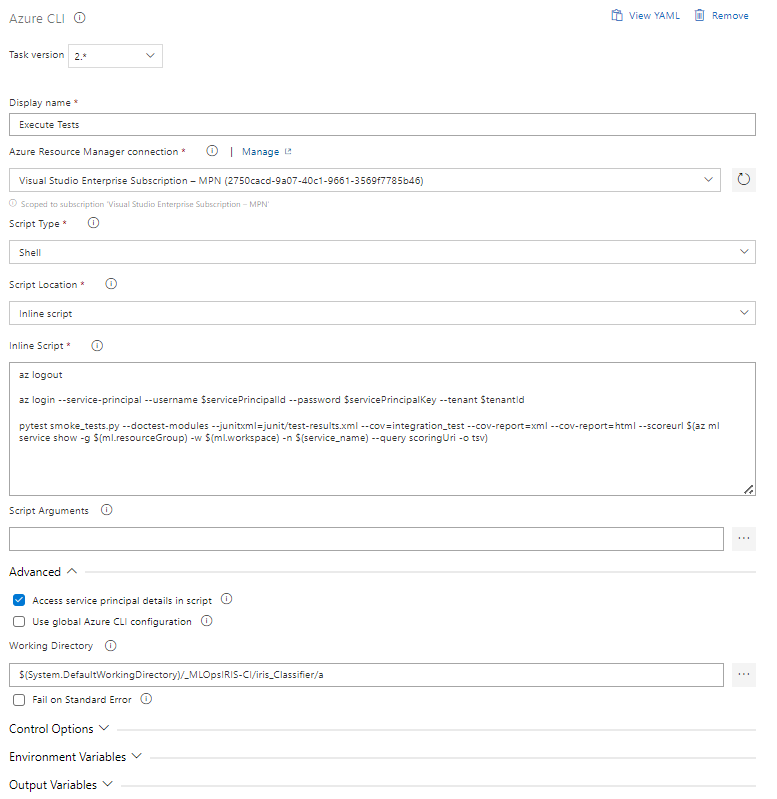
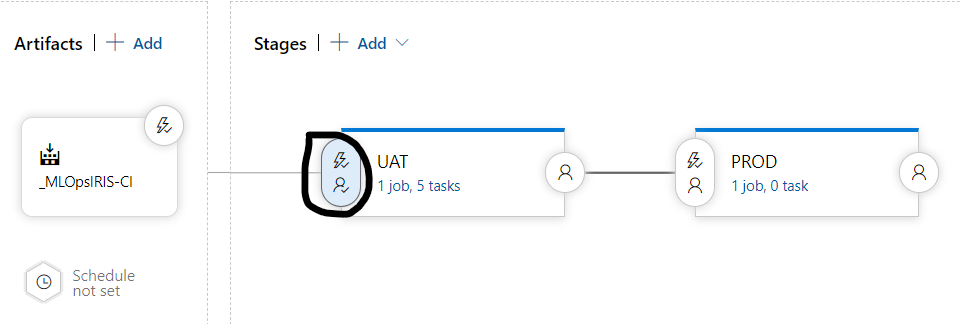
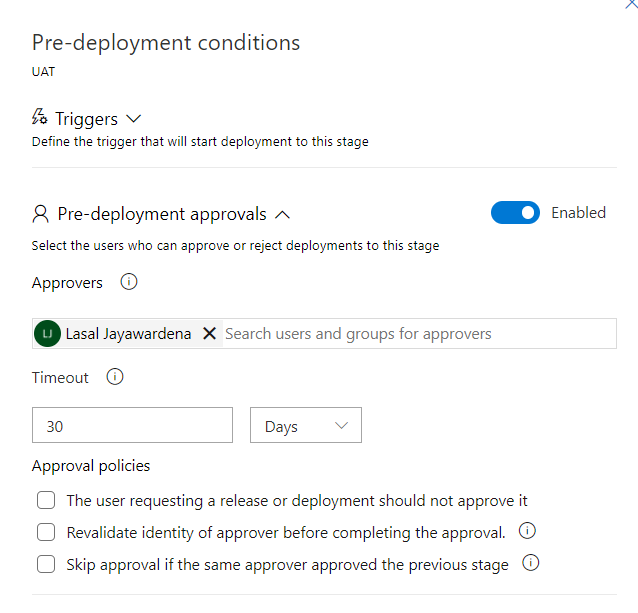
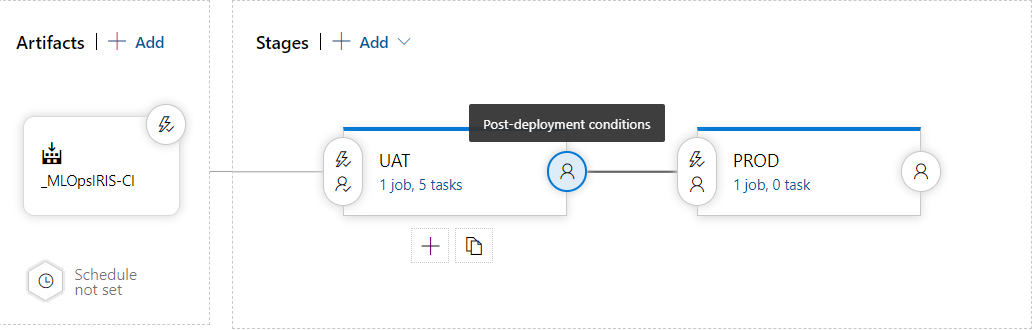
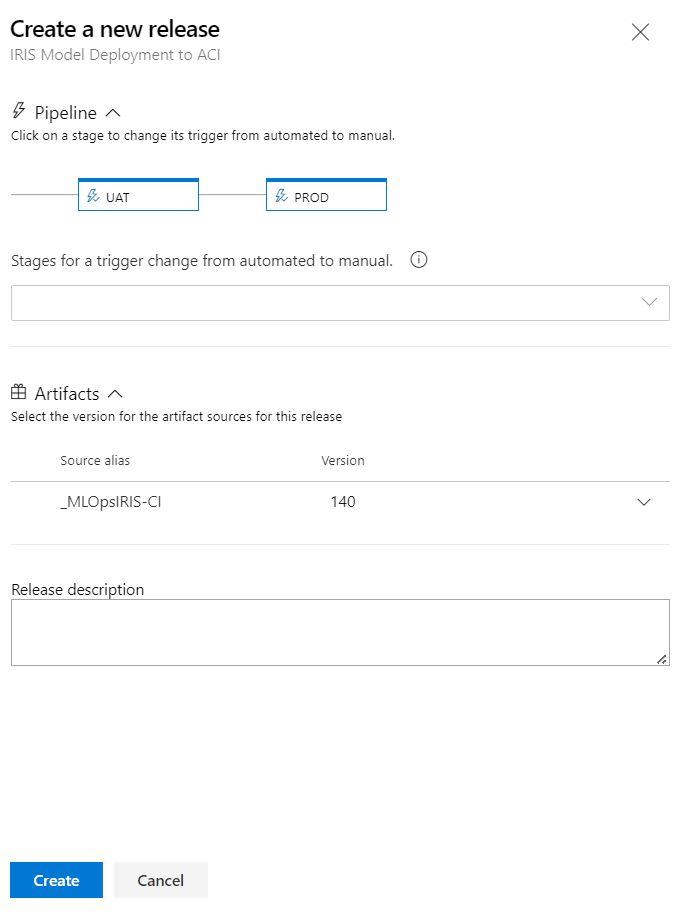
* Creating a new Service Connection  
    
  Step 1:  
  Go to Project Settings within your DevOps Projects. (It is in the Bottom left-hand Corner).  
    
  Step 2:   
  Go to Service Connections  
    
  Step 3:   
  Click on “New Service Connection”, and create a New Service Connection.  
    
  Step 4:  
  Choose the Azure Classic Connection Type and click next.  
    
    
  Step 5:  
    
  Choose ”Credentials” as the Authentication Method and fill in the required details such as Subscription ID, Subscription Name, etc. Make sure you have granted access to all pipelines by clicking on the last checkbox.   
    
    
    
  The Service Connection will be used within the pipeline For CI and CD.
* Import The Git Repo of the Pipeline  
    
  Go to “Repos” of the Azure DevOps Project. Import the Template Code of  
  MLOpsIRIS repo using the URL.  
    
    
  

1. Building the CI Pipeline  
     
   **Step 1:**  
   Go to Pipelines in Azure Devops and Select “New Pipeline”.  
     
     
   **Step 2:**  
   Select the Classic Editor down below.  
     
     
     
   **Step 3:**  
   Select Azure Repos Git as the source. Choose the Branch, which by default is master. Then Continue.   
     
     
     
   **Step 4:**  
   For the Template, select “Empty job” and continue with the steps below.  
     
     
   **Step 6:**  
   Click on ”Agent Job 1”. Change the Display name and other attributes as per your project requirement.   
     
   For the Steps below click on the ”+” icon to create new tasks for this Agent Job.  
     
   Before Step 7, This the overview of the Entire Pipeline.  
     
     
     
     
   **Step 7:**  
   Creating Task 1.  
     
     
     
   Add a new task. This will be a ”Use Python Version Task” like below.  
     
     
   Fill in the Following Attributes in the task.  
     
   You need to create a GitHub Personal Token. Follow this [guide](https://docs.github.com/en/enterprise-server@3.4/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token) to get the token. This Token will be used to download the Python Interpreter for the Pipeline. You can move forward without it, but you might face an error since there is a limit on the number of times you could go with the default token used by Azure.   
     
   Task Explanation -   
   This task is required to install the Python Interpreter that will be used in the Azure CLI and subsequently all the steps needed to launch Azure ML Jobs.  
     
   **Step 8:**  
   Creating Task 2.  
     
     
   Add a new ”Bash” task like below.  
     
     
     
   Fill in the Following Attributes in the task.  
     
     
   Task Explanation -   
   This task is needed to install all the necessary libraries such as azure CLI and azure ml SDK. Using these libraries, we will launch the training jobs.  
     
   **Step 9:**  
   Creating Variable Groups for CI Pipeline.   
   Variables give you a convenient way to get key bits of data into various parts of the pipeline. So that you extrapolate the key attributes for each task.  
   Check this [doc](https://learn.microsoft.com/en-us/azure/devops/pipelines/process/variables?view=azure-devops&tabs=yaml%2Cbatch) for more info.  
     
   Go to “Library“, which is under Pipelines in the Navbar.   
     
     
   Then Create Variable groups like down below for the CI Pipeline.  
     
     
   Variable Review:  
     
   ml.computeIdleSecs - Number of Seconds after which the Compute Cluster will turn off.   
     
   ml.computeMaxNodes - Maximum number of nodes for the Compute Cluster   
     
   ml.computeMinNodes - Minimum number of nodes for the Compute Cluster  
     
   ml.computeName - Name of the Compute Cluster  
     
   ml.computeVMSize - VM (Virtual Machine) size used for the pipeline. Standard\_DS2\_V2 will be sufficient for the current pipeline. Check out this [doc](https://learn.microsoft.com/en-us/azure/machine-learning/concept-compute-target) for more options. The Options are dependent on the Region where the Resource Group is located.  
     
   ml.DtExperimentName - Name of the Decision Tree Training Job. This is reflected in the Azure ML Studio under the ”Jobs” Section.  
      
   ml.region - Region where the Resource Group is Located.  
     
   ml.resourceGroup - Name of the Resource Group.  
     
   ml.SVmExperimentName - Name of the Support Vector Training Job. This is reflected in the Azure ML Studio under the ”Jobs” Section.  
     
   ml.workspace - Name of the Azure Machine Learning Studio Workspace.   
     
     
     
   Variable Review:  
     
   azml.ExperimentName - Name of the Dataset Versioning Task. This is reflected in the Azure ML Studio under the ”Jobs” Section.  
     
   azml.sasURL - SAS URL for the CSV file in the Blob Storage. Use your own generated SAS URL for this. Check this [doc](https://learn.microsoft.com/en-us/azure/cognitive-services/translator/document-translation/create-sas-tokens?tabs=Containers) on how to obtain the SAS URL.   
     
     
   After Creating the Variable Group make sure to save them.   
     
     
   **Step 10:**  
   Linking the Variables to the CI Pipeline.   
     
   Go to Variables in the CI Pipeline Editor  
     
   Then Go to ”Variable Groups” and click ”Link Variables Groups”. Link the Two Variable Groups that You created above.  
     
     
   After this, Continue with the Steps below.  
     
     
   For the Steps below, whenever you need to specify a ” Azure Resource Manager connection”, use the Service Connection created in Step 1).  
      
   **Step 11:**  
   Creating Task 3.  
     
     
     
   Add a new ”Azure CLI” task like below.  
     
     
   Fill in the Following Attributes in the task.  
     
     
   Inline Script:  
   az extension add -n azure-cli-ml  
     
   Task Explanation -   
   In this task we install Azure Machine Learning Extension using the [az extension](https://learn.microsoft.com/en-us/cli/azure/extension?view=azure-cli-latest) command. You can check this [guide](https://learn.microsoft.com/en-us/cli/azure/extension?view=azure-cli-latest#az-extension-add) to learn more about how to add any other additional extensions and dive deeper into the parameters used.  
     
   **Step 12:**  
   Creating Task 4.  
     
     
     
   Add a new ”Azure CLI” task and fill in the following attributes.  
     
     
   Inline Script:  
   az ml workspace create -g $(ml.resourceGroup) -w $(ml.workspace) -l $(ml.region) --exist-ok –yes  
     
   Task Explanation -   
   In this task we create an Azure Machine Learning Workspace if needed using the [az ml workspace](https://learn.microsoft.com/en-us/cli/azure/ml/workspace?view=azure-cli-latest) command. You can check this [guide](https://learn.microsoft.com/en-us/cli/azure/ml/workspace?view=azure-cli-latest#az-ml-workspace-create) to dive deeper into the parameters used.  
     
   **Step 13:**  
   Creating Task 5.  
     
     
     
   Add a new ”Azure CLI” task and fill in the following attributes.  
     
     
     
   Inline Script:  
     
   az ml computetarget create amlcompute -g $(ml.resourceGroup) -w $(ml.workspace) -n $(ml.computeName) -s $(ml.computeVMSize) --min-nodes $(ml.computeMinNodes) --max-nodes $(ml.computeMaxNodes) --idle-seconds-before-scaledown $(ml.computeIdleSecs)  
     
   Task Explanation -   
   In this task we create the Compute that will be used by Azure Machine Learning to run the Jobs in the Workspace using the [az ml computetarget](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/computetarget?view=azure-cli-latest) command. You can check this [guide](https://learn.microsoft.com/en-us/cli/azure/ml/workspace?view=azure-cli-latest#az-ml-workspace-create) to dive deeper into the parameters used. Make sure the Compute target is not used by any notebook or other pipeline else the task will stall.  
     
   **Step 14:**  
   Creating Task 6.  
     
     
     
   Add a new ”Bash” task and fill in the following attributes.  
     
     
   Inline Script:  
   mkdir metadata && mkdir models  
     
   Task Explanation -   
   In this task we create the folders where the binary files and the metadata files of the models will be stored. These folders will then be passed as artifacts to the CD pipeline.  
     
     
   **Step 15:**  
   Creating Task 7.  
     
     
     
   Add a new ”Azure CLI” task and fill in the following attributes.  
     
     
     
   Inline Script:  
   az ml run submit-script -g $(ml.resourceGroup) -w $(ml.workspace) -e $(azml.ExperimentName) --ct $(ml.computeName) -c version\_data --source-directory . --path environment\_setup -t ./metadata/version\_data.json version\_data.py --dataset\_name iris\_ds --blob\_sas\_url '$(azml.sasURL)' --version  
     
   Task Explanation -   
   In this task will run the Data Versioning Job in the Azure ML Workspace using the [az ml run](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/run?view=azure-cli-latest) command. You can check this [guide](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/run?view=azure-cli-latest#az-ml(v1)-run-submit-script) to dive deeper into the parameters used.   
   If you do not want to create a new data version, make sure to pass ”--no-version” instead of ”--version” or you can simply disable the task.  
     
     
     
     
   **Step 16:**  
   Creating Task 8.  
     
     
     
   Add a new ”Azure CLI” task and fill in the following attributes.  
     
     
     
   Inline Script:  
   az ml run submit-script -g $(ml.resourceGroup) -w $(ml.workspace) -e $(ml.DtExperimentName) --ct $(ml.computeName) -c dt\_iris\_training --source-directory . --path environment\_setup -t ./metadata/dt\_run.json dt\_iris\_training.py --container\_name irisdata --input\_csv Iris.csv --model\_path ./models/dt\_iris\_model.pkl --artifact\_loc ./outputs/models/ --dataset\_name iris\_ds --dataset\_desc "IRIS Data Set"  
     
   Task Explanation -  
   In this task will run the Decision Tree Model training Job in the Azure ML Workspace using the [az ml run](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/run?view=azure-cli-latest) command. You can check this [guide](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/run?view=azure-cli-latest#az-ml(v1)-run-submit-script) to dive deeper into the parameters used.   
     
     
   **Step 17:**  
   Creating Task 9.  
     
     
     
   Add a new ”Azure CLI” task and fill in the following attributes.  
     
     
   Inline Script:  
   az ml run submit-script -g $(ml.resourceGroup) -w $(ml.workspace) -e $(azml.ExperimentName) --ct $(ml.computeName) -c version\_data --source-directory . --path environment\_setup -t ./metadata/version\_data.json version\_data.py --dataset\_name iris\_ds --blob\_sas\_url '$(azml.sasURL)' --version  
     
   Task Explanation -  
   In this task will run the Support Vector Machine Model training Job in the Azure ML Workspace using the [az ml run](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/run?view=azure-cli-latest) command. You can check this [guide](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/run?view=azure-cli-latest#az-ml(v1)-run-submit-script) to dive deeper into the parameters used.  
     
     
   **Step 18:**  
   Creating Task 10.  
     
     
     
   Add a new ”Azure CLI” task and fill in the following attributes.  
     
     
     
   Inline Script:  
   az ml model register -g $(ml.resourceGroup) -w $(ml.workspace) -n IRIS\_DT --asset-path outputs/models/ -d "IRIS Decision Tree Classifier" --tag "model"="Decision Tree" --model-framework Custom -f ./metadata/dt\_run.json -t metadata/dt\_model.json  
     
   Task Explanation -  
   In this task will run the Decision Tree Model Registering Job in the Azure ML Workspace using the [az ml model](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/model?view=azure-cli-latest) command. This job will use the metadata generated from the Training Job to register the model in the Azure ML Workspace. You can check this [guide](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/model?view=azure-cli-latest#az-ml(v1)-model-register) to dive deeper into the parameters used.  
     
   **Step 19:**  
   Creating Task 11.  
     
     
     
   Add a new ”Azure CLI” task and fill in the following attributes.  
     
     
     
   Inline Script:  
   az ml model register -g $(ml.resourceGroup) -w $(ml.workspace) -n IRIS\_SVM --asset-path outputs/models/ -d "IRIS SVM" --tag "model"="Support Vector Machine" --model-framework Custom -f ./metadata/svm\_run.json -t metadata/svm\_model.json  
     
   Task Explanation -  
   In this task will run the Support Vector Machine Model Registering Job in the Azure ML Workspace using the [az ml model](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/model?view=azure-cli-latest) command. This job will use the metadata generated from the Training Job to register the model in the Azure ML Workspace. You can check this [guide](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/model?view=azure-cli-latest#az-ml(v1)-model-register) to dive deeper into the parameters used.  
     
   **Step 20:**  
   Creating Task 12.  
     
     
     
   Add a new ”Copy Files” task.  
     
     
     
   Fill in the following attributes in the Task.  
     
     
   Contents:  
     
   \*\*/metadata/\*  
   \*\*/environment\_setup/\*  
   \*\*/deployment/\*  
   \*\*/inference/\*  
   \*\*/tests/smoke/\*  
   \*\*/outputs/prediction.csv  
     
   Task Explanation -  
   In this task will copy all the files needed for the CD pipeline and publish them as artifacts of the CI pipeline.  
     
     
   **Step 21:**  
   Creating Task 13.  
     
     
     
   Add a new ” Publish Pipeline Artifacts” task.  
     
     
   Fill in the following attributes in the task.  
     
     
     
   These are all the steps in building the CI Pipeline. Next step is to run the pipeline and see.  
     
     
   **Running the Pipeline**  
     
   Select the ”Run pipeline”.   
     
     
   Make sure to select the latest ubuntu version.   
   Also, enable System Diagnostics.  
     
     
   This is the end of building the CI pipeline.

# **Steps to Build the CD Pipeline**

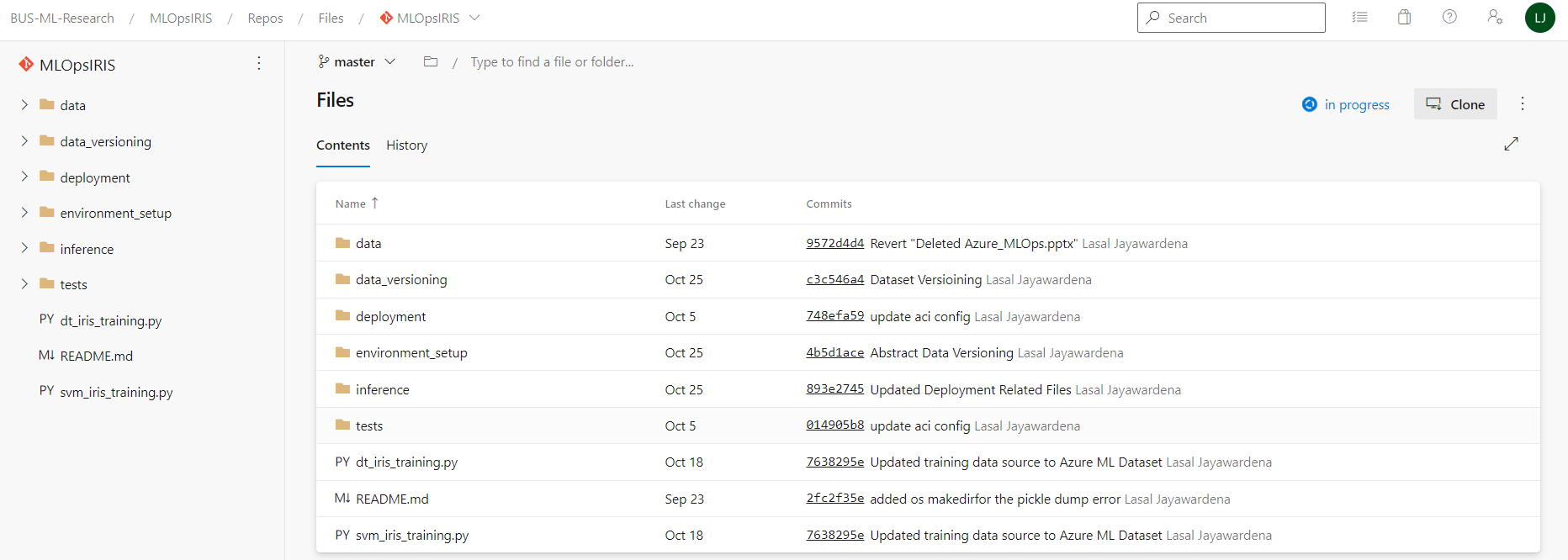
Structure of the CD Pipeline



1. Initial Release Setup  
     
   **Step 1:**  
   Go to “Releases” in Pipelines.  
     
     
     
   **Step 2:**  
   Select ”New” and choose ”New Release Pipeline”.   
   Then Choose ”Empty Job” as the template.  
     
     
     
     
   **Step 3:**  
   Setup Artificats.   
     
   Click on “Add an artifact” and select ”Build” as source type.  
   Choose the CI build in the dropdown and then add.  
     
     
     
   Expected Result.  
   
2. Building the UAT Stage  
     
   **Step 1:**  
   Create a new Stage and name it UAT.  
     
   **Step 2:**  
   Click on view stage tasks to add Tasks  
     
     
     
   Continue following the steps below.  
     
   UAT Task Overview  
     
     
     
   **Step 3:**  
   Creating Variable Groups for CD Pipeline.   
   Check this [doc](https://learn.microsoft.com/en-us/azure/devops/pipelines/process/variables?view=azure-devops&tabs=yaml%2Cbatch) for more info on Azure Devops Variables.  
     
   Go to “Library“, which is under Pipelines in the Navbar.   
     
     
   Then Create the Variable group like down below for the CD Pipeline.  
     
     
     
   Variable Review:  
     
   ml.resourceGroup - Name of the Resource Group.  
     
   ml.workspace - Name of the Azure Machine Learning Studio Workspace.   
     
     
   After Creating the Variable Group make sure to save them.   
     
     
   **Step 4:**  
   Linking the Variables to the CD Pipeline.   
     
   Go to Variables in the UAT Task Editor  
     
     
   Then Go to ”Variable Groups” and click ”Link Variables Groups”. Link the Variable Group that you created above.  
     
     
   Add the below pipeline variable as well.  
     
     
     
   After this, Continue with the Steps below.  
     
     
   **Step 5:**  
   Configuring UAT’s Agent Job.  
     
   Click on Agent Job  
      
     
   Make sure the Agent has the following configurations. Check whether the ”Agent Configuration” is set to the latest ubuntu version.  
     
     
     
   **Step 6:**  
   UAT Task 1.  
     
     
     
   Add a new ”Use Python Version Task” task and fill the Following Attributes in the task.  
     
   You need to create a GitHub Personal Token. Follow this [guide](https://docs.github.com/en/enterprise-server@3.4/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token) to get the token.  
     
   Task Explanation -   
   This task is required to install the Python Interpreter that will be used in the Azure CLI and subsequently all the steps needed to launch Azure ML Jobs.  
     
   **Step 7:**  
   UAT Task 2.  
     
     
   Fill in the Following Attributes in the task.  
     
     
   Task Explanation -   
   This task is needed to install all the necessary libraries such as azure CLI and azure ml SDK. Using these libraries, we will launch the training jobs.  
     
   **Step 8:**  
   UAT Task 3.  
     
     
     
   Add a new ”Azure CLI” task and fill in the Following Attribute.  
     
     
   Inline Script:  
   az extension add -n azure-cli-ml  
     
   Task Explanation -   
   In this task we install Azure Machine Learning Extension using the [az extension](https://learn.microsoft.com/en-us/cli/azure/extension?view=azure-cli-latest) command. You can check this [guide](https://learn.microsoft.com/en-us/cli/azure/extension?view=azure-cli-latest#az-extension-add) to learn more about how to add any other additional extensions and dive deeper into the parameters used.  
     
     
   **Step 9:**  
   UAT Task 4.  
     
     
     
   Add a new ”Azure CLI” task and fill in the Following Attribute.  
     
     
     
   Inline Script:  
   az logout  
     
   az login --service-principal --username $servicePrincipalId --password $servicePrincipalKey --tenant $tenantId   
     
   az ml model deploy -g $(ml.resourceGroup) -w $(ml.workspace) -n $(service\_name) -f dt\_model.json -f svm\_model.json --dc aciDeploymentConfig.yml --ic inferenceConfig.yml --overwrite  
     
   Task Explanation -  
   This task will deploy the best performing ML Model to Azure Container Instances using the [az ml model](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/model?view=azure-cli-latest) command.   
   This job will use the metadata generated from the Training Job in the CI pipeline and then access the models. The default evaluation criterion is precision which can be customized in the “score.py“ script. You can pass any number of models by passing the .json file generated from the training job. Monitor the space required by all the models as a potential cause of task failure.   
   You can check this [guide](https://learn.microsoft.com/en-us/cli/azure/ml(v1)/model?view=azure-cli-latest#az-ml(v1)-model-deploy) to dive deeper into the parameters used.  
     
     
   **Step 10:**  
   UAT Task 5.  
     
     
     
   Add a new ”Azure CLI” task and fill in the Following Attribute.  
     
     
     
   Inline Script:  
   az logout  
     
   az login --service-principal --username $servicePrincipalId --password $servicePrincipalKey --tenant $tenantId   
     
   pytest smoke\_tests.py --doctest-modules --junitxml=junit/test-results.xml --cov=integration\_test --cov-report=xml --cov-report=html --scoreurl $(az ml service show -g $(ml.resourceGroup) -w $(ml.workspace) -n $(service\_name) --query scoringUri -o tsv)  
     
   Task Explanation -  
   The task will run a simple smoke test to see whether the best Model Deployment is successful.  
     
     
   **Step 11 (Optional):**  
   Adding Pre-Deployment / Post Deployment Approvals.  
     
   Click on the Thunder Icon  
     
     
   Enable Pre-deployment Approvals and add users who need to give approval before the CD pipeline begins  
     
   Same procedure if you want Post-deployment Approvals. Select the User Icon at the end of each stage as per requirement.  
     
   
3. Building the Production Stage  
     
   **Step 1:**  
   Create a new Stage and name it PROD.  
     
   For this demo, this will be just a dummy stage. So that is it. Possible Architectures/Tasks will be discussed under the “Tailoring To your Requirements” section.  
     
   These are all the steps in building the CI Pipeline. Next step is to run the pipeline and see.
4. Running the CD pipeline  
     
   Select the CD Pipeline above and select “Create release”.  
     
   After Creating a Release Give the necessary Approval for the release to start if needed.  
     
     
     
   

# **Project File Structure**

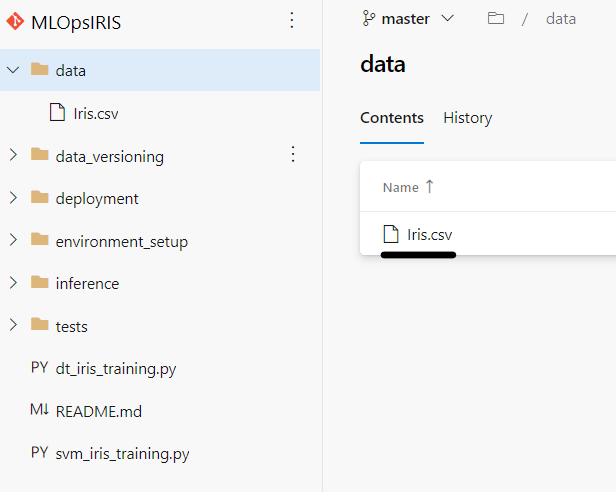
# **Repository Overview**



Below will below will be an overview of what each folder and file contain.

# **The “data” Folder**

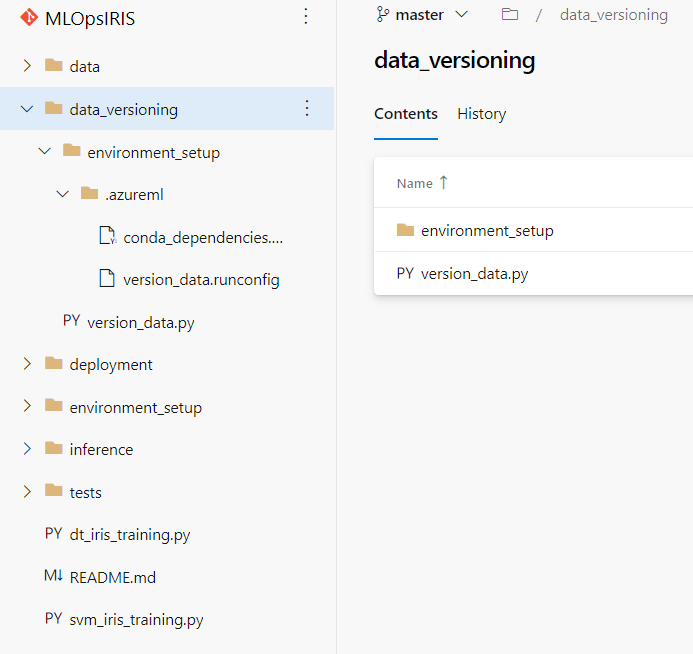
The data folder contains the original Iris Dataset as a CSV file. The dataset documentation can be found [here.](https://archive.ics.uci.edu/ml/datasets/iris)



# **The “data\_versioning” Folder**

The data\_versioning folder contains all the code and environment setup needed for Versioning the Dataset used for Model Training.

Inside “version\_data.py” is the script where the Python script is written for versioning the Dataset. Remember that you can modify the file to take any service as a data source instead of the default Blob Storage.

The “.azureml” is the folder where the job configurations and required packages are stored. “conda\_dependencies.yml” is where you have all the needed python packages for the versioning job. “version\_data.runconfig” is the Config file for the Job.  


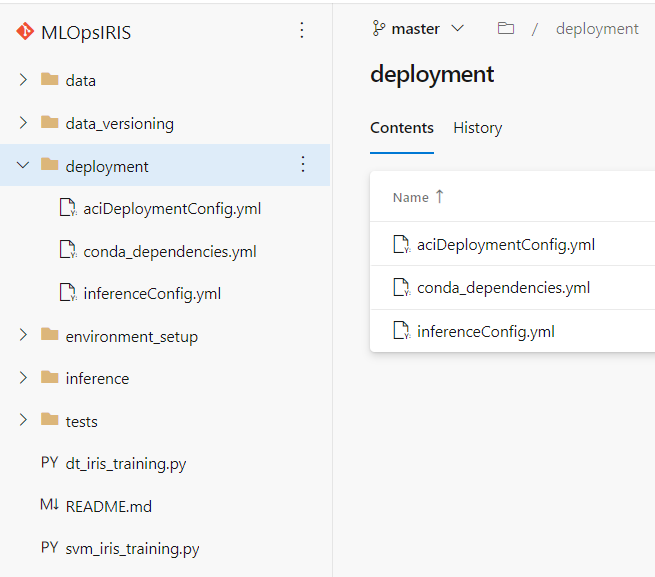
# **The “deployment” Folder**

The deployment folder contains the files needed to build the infrastructure for Model Deployment after Model Training.

“aciDeploymentConfig.yml” is a file where we specify the required specification for the Azure Container Instance to be created.

“conda\_dependencies.yml” is where you have all the needed python packages for running inference on the deployed model.

“inferenceConfig.yml” is the file where you have attributes regarding about Model Inference Environment and references the Scoring script.

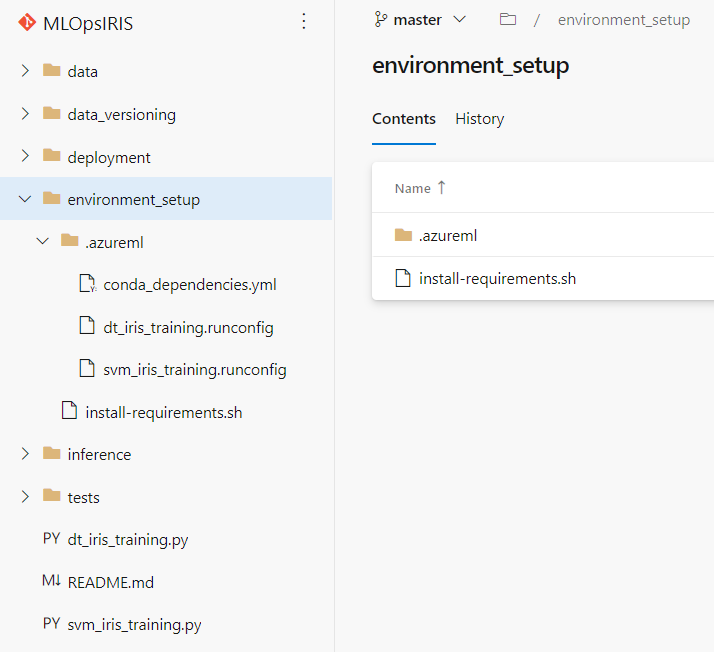


# **The “environment\_setup” Folder**

The environment\_setup folder contains all the files needed for environment setup for each Model Training Job.

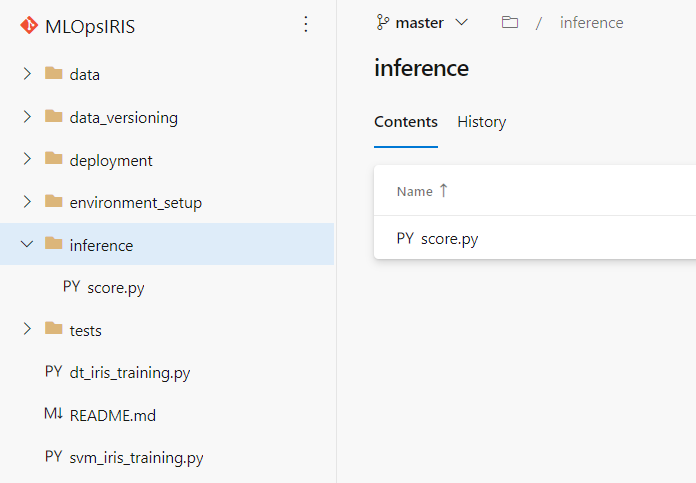
“install-requirements.sh” is a bash file that installs the packages in the Azure CLI.   
Reason being that the Azure CLI is where the training Jobs will be launched from in the CI pipeline.

“conda\_dependencies.yml” is where you have all the needed python packages for both the SVM and DT training jobs. You can have additional dependency files if other training jobs require different packages or versions of a certain package.  
  
“dt\_iris\_training.runconfig” and “svm\_iris\_training.runconfig” are the Config files for the SVM and DT training jobs.



# **The “inference” Folder**

The inference folder contains the script for deploying the best model and the scoring logic. This is where the best is chosen on a pre-defined metric/metrics and then deployed to Azure Container Instances. The Model is evaluated on a data stored in a Blob Container. Modify this data source as per project requirement.



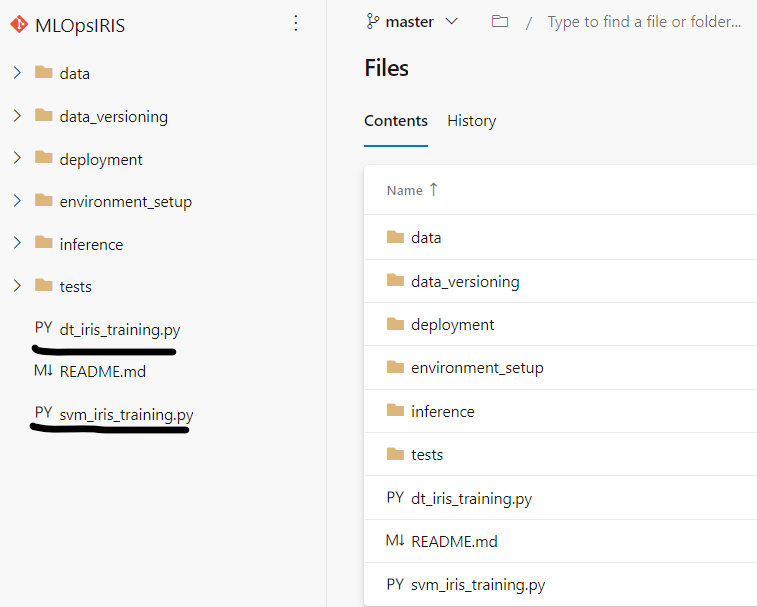
# **The “tests” Folder**

The tests folder contains the logic for a basic smoke test used in the CD pipeline to validate whether the model deployment is successful.



# **The Main Training Scripts**

“dt\_iris\_training.py” and “svm\_iris\_training.py” are the scripts used for training the SVM and DT models. You abstract the training script and configurations to separate folders. Make sure to be careful about script and configuration locations since there might be bit of a hassle.



# **Data Storage**

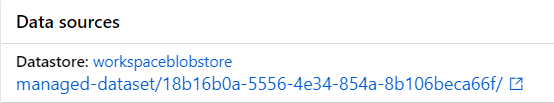
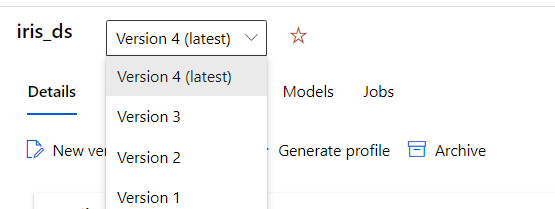
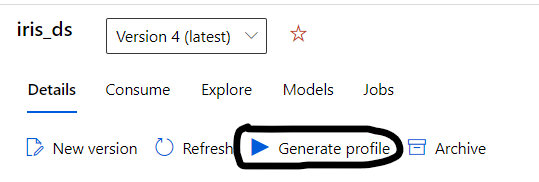
# **Dataset used for Model Training**

The Dataset used for Model Training needs to have two unique characteristics apart from the project related dataset requirements; they are Dataset versioning and Easy Accessibility.

Data versioning enables one to easily understand if a newer version of a dataset is available. Explicit versioning allows for repeatability in research, enables comparisons, and prevents confusion. Easy Accessibility. Allows the consumer to easily ingest the data within the shortest period of time.

For these reasons, an **Azure ML Dataset** is used as the main data source for model training in the pipeline. You could refer to this [doc](https://learn.microsoft.com/en-us/azure/machine-learning/v1/how-to-create-register-datasets) to help you get started with building your own Azure ML Dataset and this [doc](https://learn.microsoft.com/en-us/python/api/azureml-core/azureml.core.dataset.dataset?view=azure-ml-py) which has all methods and related documentation.

Here are a few details about an **Azure ML Dataset:**

* You can use any data source as an **Azure ML Dataset**. You have different Dataset types that you could register your Dataset as. You have the [FileDataset](https://learn.microsoft.com/en-us/python/api/azureml-core/azureml.data.file_dataset.filedataset) which is suitable for Computer Vision Datasets or datasets where data points are ingested iteratively through files. Then there is [TabularDataset](https://learn.microsoft.com/en-us/python/api/azureml-core/azureml.data.tabulardataset) which is suitable for structured datasets like CSV, TSV, Parquet, JSONL files, and SQL query results.  
  
* Data Wrangling capabilities with Metadata using methods like [filter().](https://learn.microsoft.com/en-us/python/api/azureml-core/azureml.data.tabulardataset#filter-expression-)
* Fine Data Administration. Explore this [doc](https://learn.microsoft.com/en-us/azure/machine-learning/how-to-administrate-data-authentication) to dive in depth.
* Easy to switch between Dataset versions and can import multiple dataset versions at once. This capability is incredibly useful to detect data drift, concept drift and during experimentations.   
    
  
* Ability to generate Dataset profiles with a click of a button. This will create a profile with summary statistics, distribution of the data, etc.  
  

In the pipeline, the original iris dataset is stored as a CSV file in a Blob container. This blob asset is then registered as an Azure ML Dataset and used in the pipeline and in the experimentation notebooks.

In the much large-scale projects, Azure ML Datastores may need to be created. Datastores securely connect to a storage service on Azure by storing connection information. With datastores, you no longer need to provide credential information in your scripts to access your data. Refer this [doc](https://learn.microsoft.com/en-us/azure/machine-learning/how-to-datastore?tabs=cli-identity-based-access%2Ccli-adls-identity-based-access%2Ccli-azfiles-account-key%2Ccli-adlsgen1-identity-based-access) for more insight on that.

# **Dataset used for Model Testing**

In the pipeline, the test dataset used in the training job for model evaluation is a subset of the Model Training Dataset. Therefore, there was no separate source used.

The data split was done using the [sklearn.model\_selection.train\_test\_split](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html) method. An important parameter passed is the “random\_state”. Setting the random state to a specific value controls the shuffling applied to the data before applying the split. This ensures reproducibility which will be helpful for testing purposes.

Ideally, there needs to be another **Azure ML Dataset** registered for testing purposes. This dataset needs to follow a similar distribution to the training Dataset. Managing this dataset must be done outside of the pipeline, using a separate azure function.

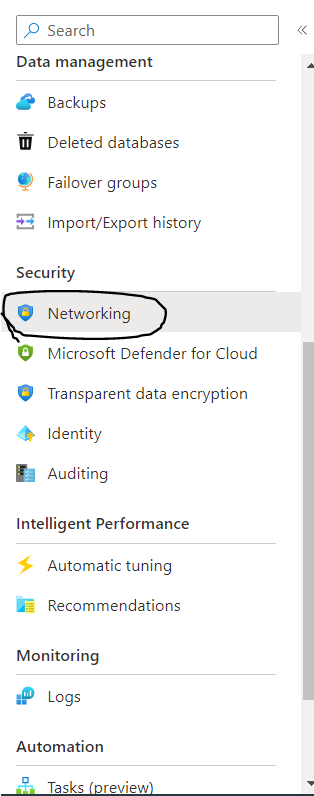
# **Storage used to Simulate Data Change/Drift**

# **To Upload data into an SQL Database**

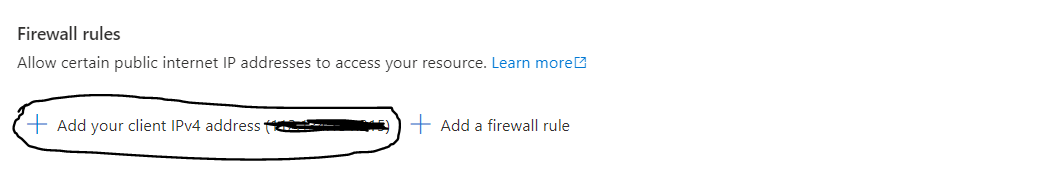
Pre-requisites- Create an SQL database and an SQL server in Azure.

Updating the firewall rule

First go to the networking section in the created SQL server



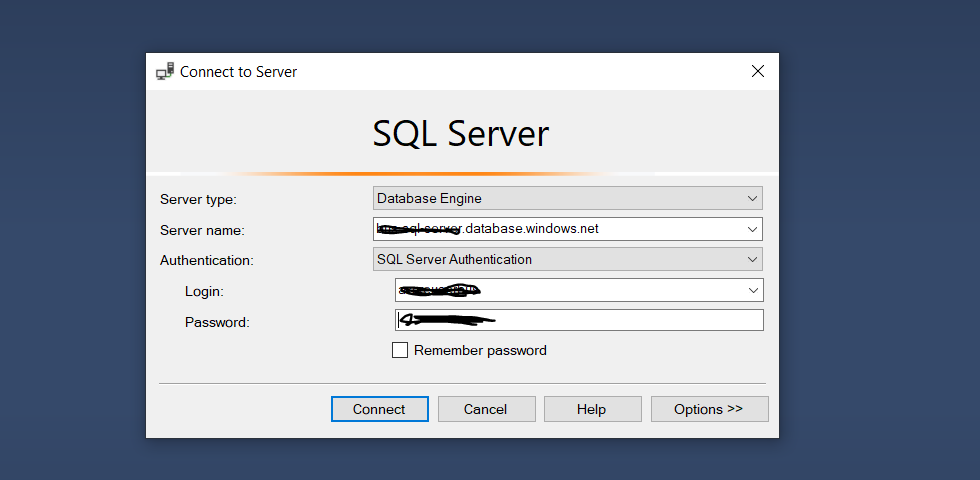
Next click on the add your client IPv4 address and add your IPv4 address to help connect the sql database to the Microsoft SQL Management Studio.



Updating the dastaset using Microdoft SQL Server Management Studio

First download Microsoft SQL Server Management Studio([SQL Server Downloads | Microsoft](https://www.microsoft.com/en-us/sql-server/sql-server-downloads?rtc=1))

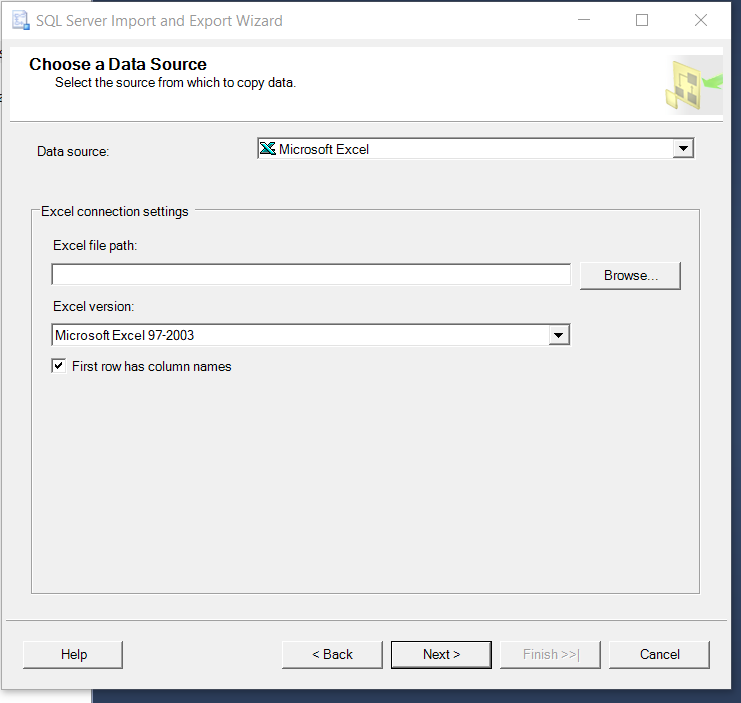
Next open Microsoft Server Management Studio and enter the required credentials.



After connection is established navigate to Databases and right click on the previously created SQL Database and select tasks and then import data.

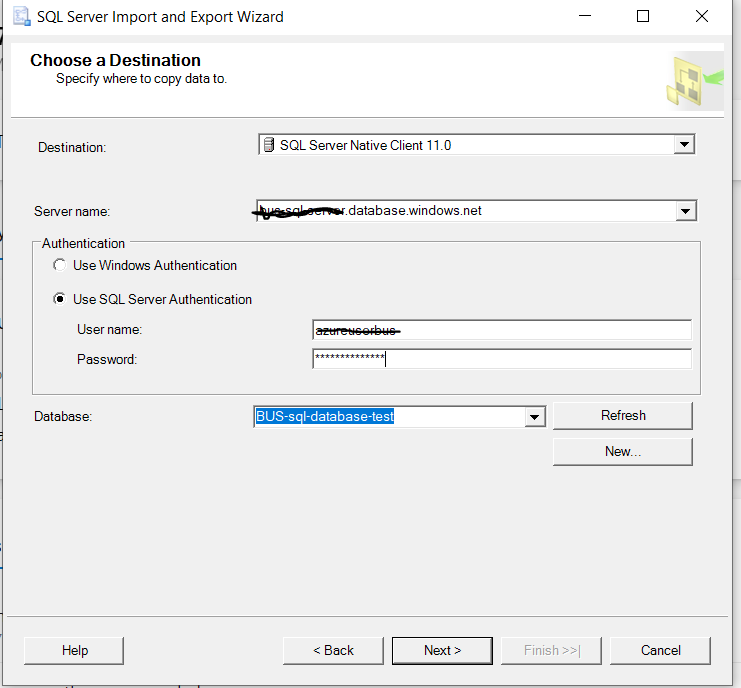


Click on next and the below window will show.



In the above window select the data source as Microsoft Excel and in the Excel file path browse to where the dataset has been saved(the dataset should be saved in the .xls format) and click on next.

Next we have to specify where to send the data.



In the above window we have to select SQL Server Native Client 11.0 as the destination. Next we have to use SQL Server Authentication and fill in the credentials and then once its done click next till we have completed the entire upload.

# **Create a BLOB Storage**

The following documentation can be used to create a container, upload a blob and download a blob ([Quickstart: Azure Blob Storage client library for Python | Microsoft Learn](https://learn.microsoft.com/en-us/azure/storage/blobs/storage-quickstart-blobs-python?tabs=connection-string%2Croles-azure-cli%2Csign-in-azure-cli)).

Special note-Within the container in Data protection under the data management section of the container we need to enable versioning for blob to help overwrite blobs and update the blobs.

# **To Extract Data from an SQL Database**

[Step 3: Connecting to SQL using pyodbc - Python driver for SQL Server | Microsoft Learn](https://learn.microsoft.com/en-us/sql/connect/python/pyodbc/step-3-proof-of-concept-connecting-to-sql-using-pyodbc?source=recommendations&view=sql-server-ver16)

A python script which connects to the sql database and queries the database was created to extract the data from the database as a csv file to upload to a blob storage.

# **Reverting Data Back to its original form in the BLOB Storage**

The existing blob is downloaded as a csv and then using the pandas package we save the first 150 rows into a dataframe and upload the new csv as a blob to overwrite the existing blob.

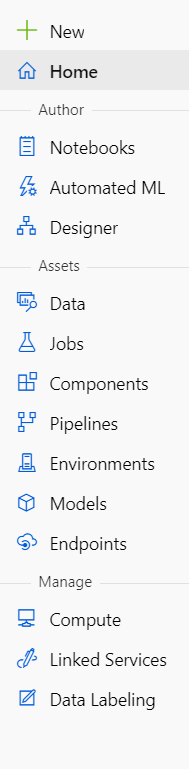
Special note- blob\_client.upload\_blob(data, overwrite=True

(Set the overwrite parameter to true to help overwrite a blob)

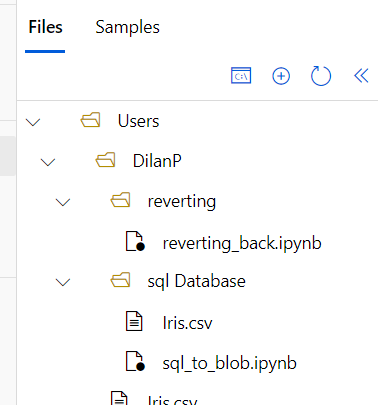
# **Running the script on Azure Machine learning Studio**

First go to Azure Machine Learning Studio.

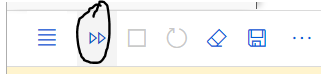
Next- Navigate to the Notebooks section in the navbar



Upload the 2 Scripts created in to extract data from an sql database and reverting data back to its original form in the Blob storages seperately as 2 different .ipynb files .



To run the files, you must create a compute and press the run button.



# **ML Algorithms Used**

# **Support Vector Machine (SVM)**

[Support Vector machine](https://en.wikipedia.org/wiki/Support_vector_machine) is a supervised machine learning model. The current project used [sklearn’s SVC algorithm](http://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html) for data classification purposes.

The Algorithm is used inside the “svm\_iris\_training.py” script.

# **Decision Trees**

[Decision Trees](https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052) are a supervised machine learning model where the data is continuously split according to a certain parameter. The tree can be explained by two entities, namely decision nodes and leaves.

The current project used [sklearn’s Decision Tree Classifier algorithm](http://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html) for classification purposes.

The Algorithm is used inside the “dt\_iris\_training.py” script.

# **Neural Networks (Still Not Incorporated)**

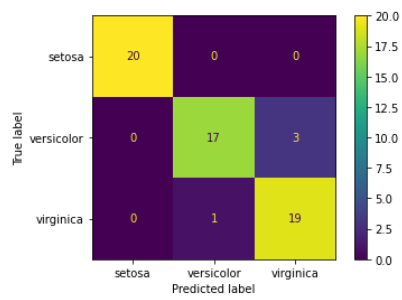
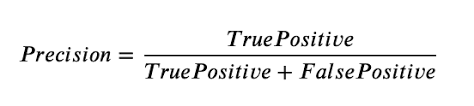
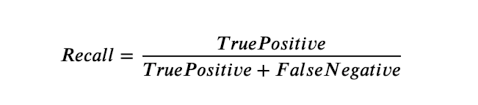
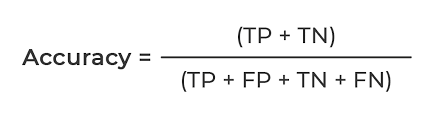
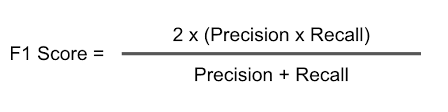
[A Neural Network](https://en.wikipedia.org/wiki/Artificial_neural_network) is a series of algorithms that tries to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.

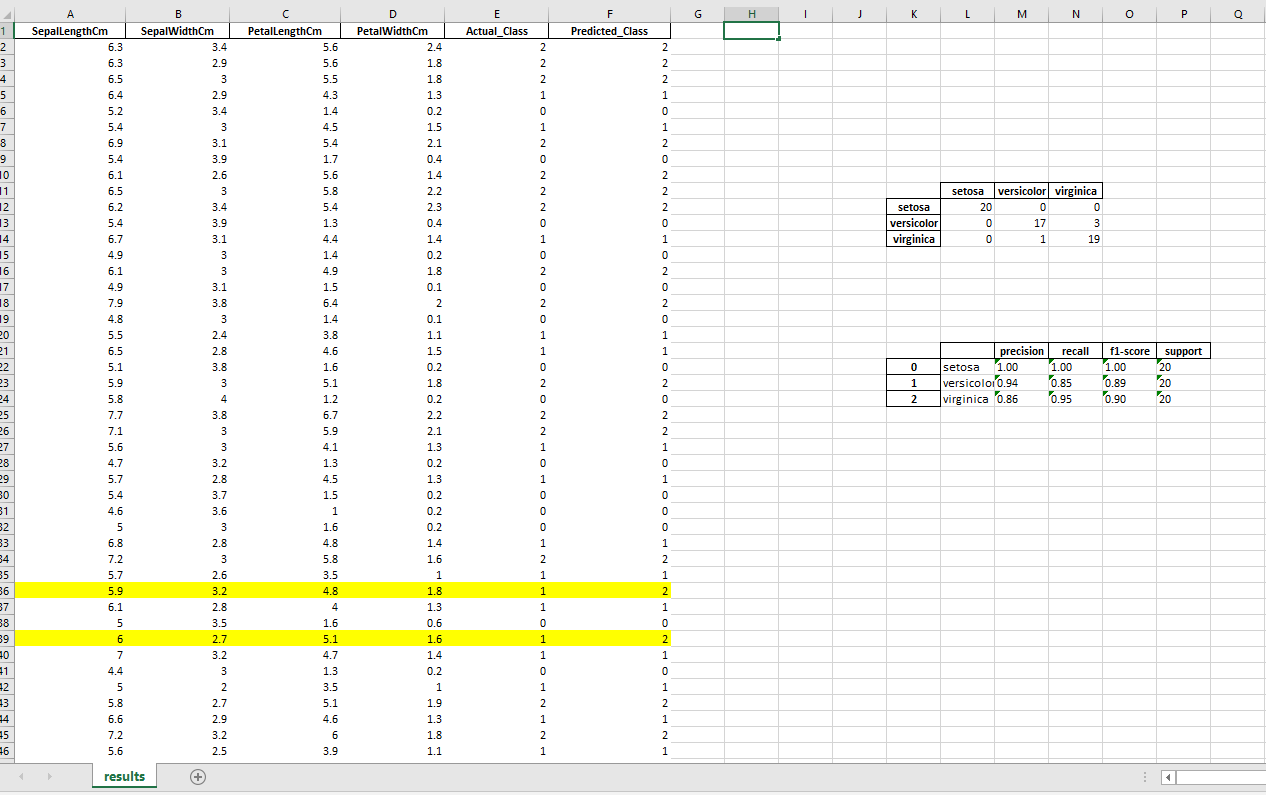
The current project uses a custom Neural Network built using [Tensorflow](https://www.tensorflow.org/) and [Keras](https://www.tensorflow.org/api_docs/python/tf/keras).

# **QA Focus**

# **Metrics Report**

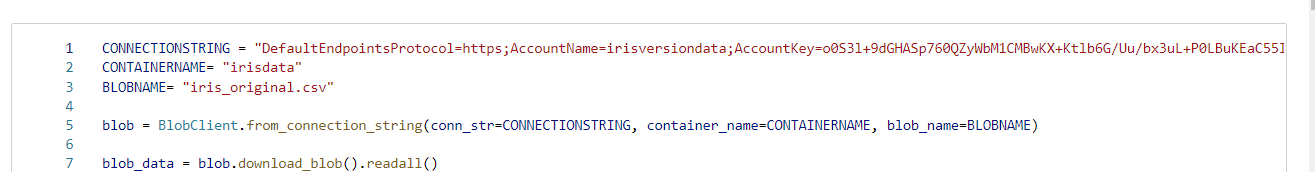
It is a report that outlines the key metrics and highlights the rows which need attention by the QA.  
  
By default:

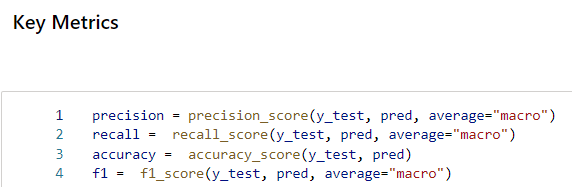
* Classification Matrix is Generated.  
  
* Predefined Metrics are Calculated on the Test Data.  
  Default Key Metrics:  
  1) Precision  
    
  2) Recall  
    
  3) Accuracy  
    
  4) F1 Score   
  
* Rows are highlighted if the predicted class was incorrect.

Sample Report View as an Excel File:  
  


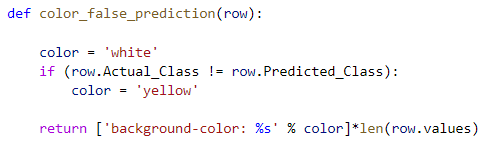
In the notebook, the predictions for the test data are generated by sending a POST request to the best ML model deployed in Azure Container Instances.

The data source of the test data can be changed by the QA. By default, a csv file from the blob is set as the source.



Note that the Metrics Calculated can be customized according to the task or project requirements.   
  


The highlighting criterion can also be customized according to the QA’s requirement.

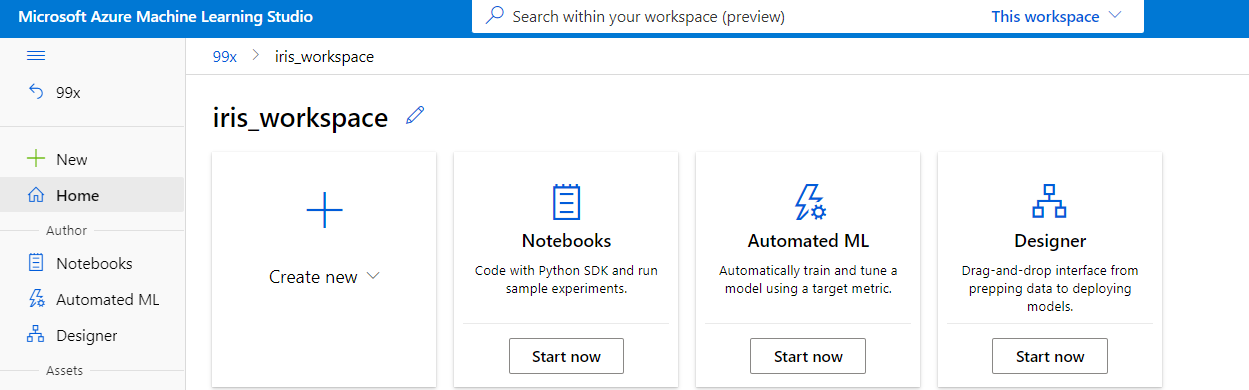
  
  
The Metrics notebook is created so that it is easier for the QA to automate the report building process.

# **Local Model Testing**

Even though the QA has access to the best model chosen by pre-defined metrics, via Azure Container Instances, there may be instances where the other candidate models need to be manually evaluated. The process outlined below will be a guide for this scenario.

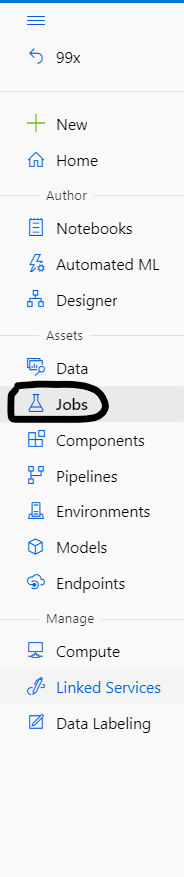
**Step 1**:

Go the Azure Machine Learning Studio Workspace where each model was trained in. The Default workspace is named “iris\_workspace”.

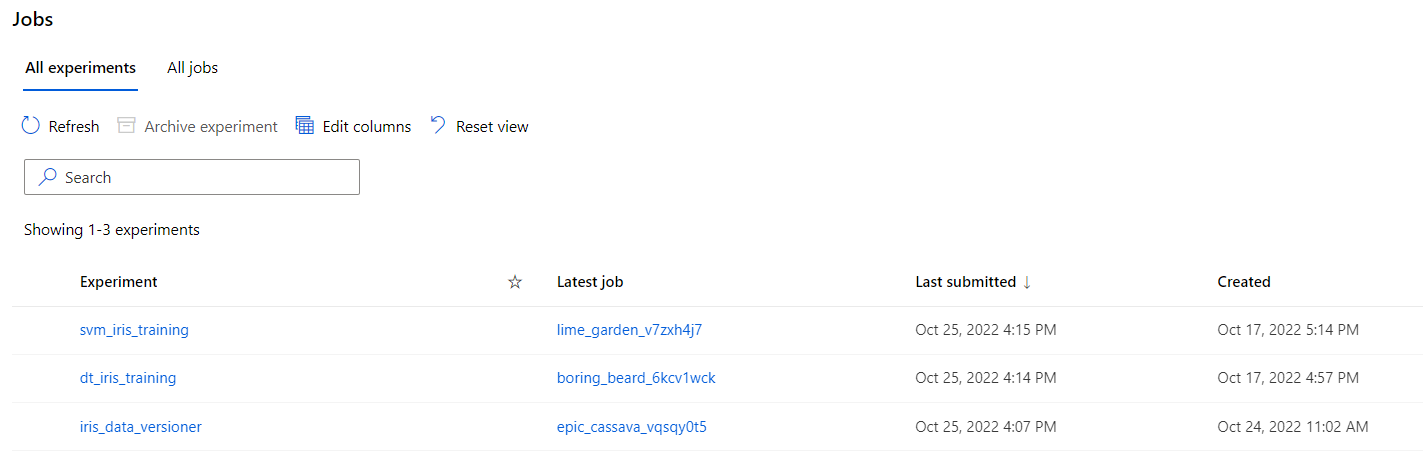


**Step 2**:

Go to the "Jobs” section. This will list all the jobs submitted to Azure Machine Learning. This includes the training jobs and the data versioning jobs.



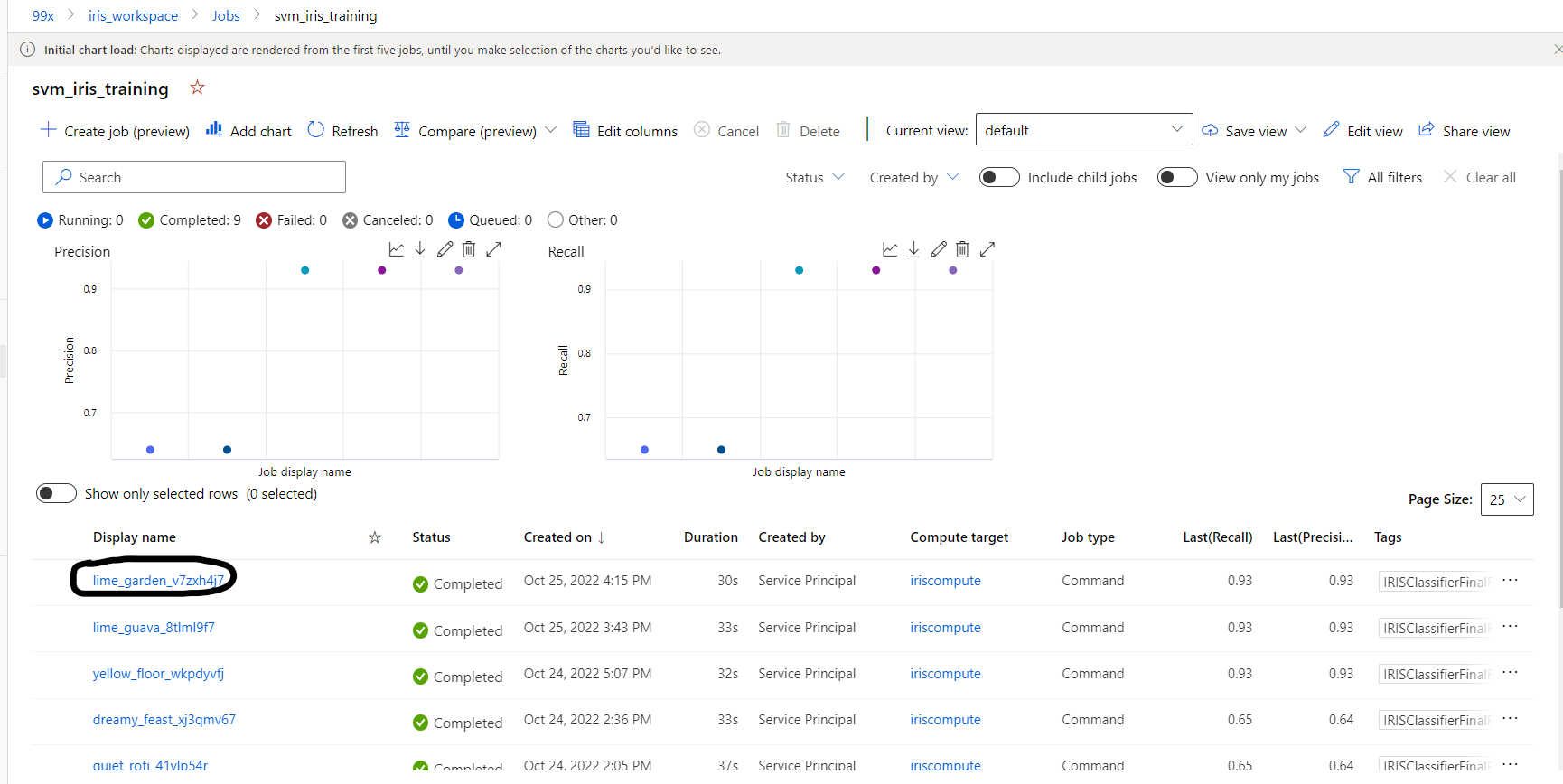
Just to validate, check whether there are “Experiments” like below:



**Step 3**:

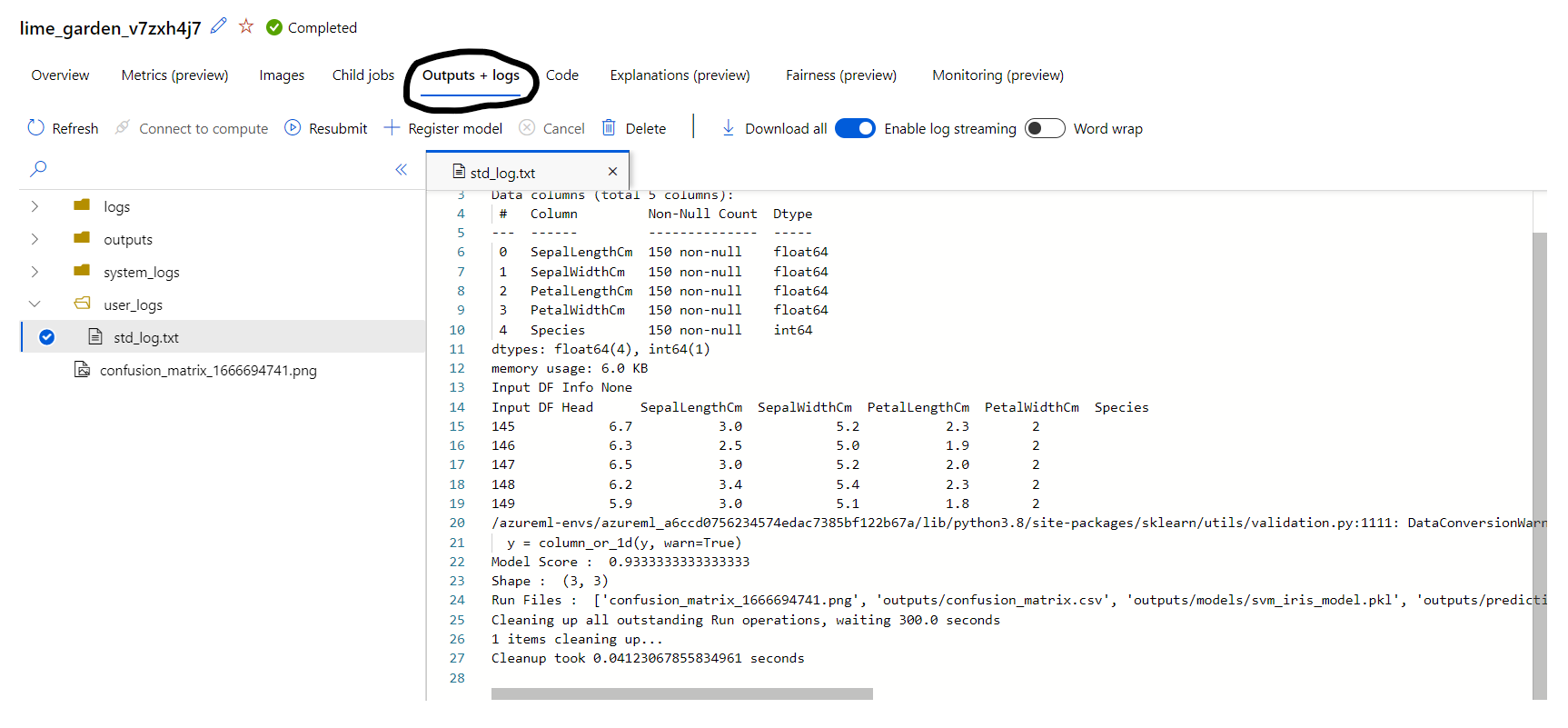
Go to a Model Training Experiment and Click on the latest training job.

The name of the training jobs may be different.



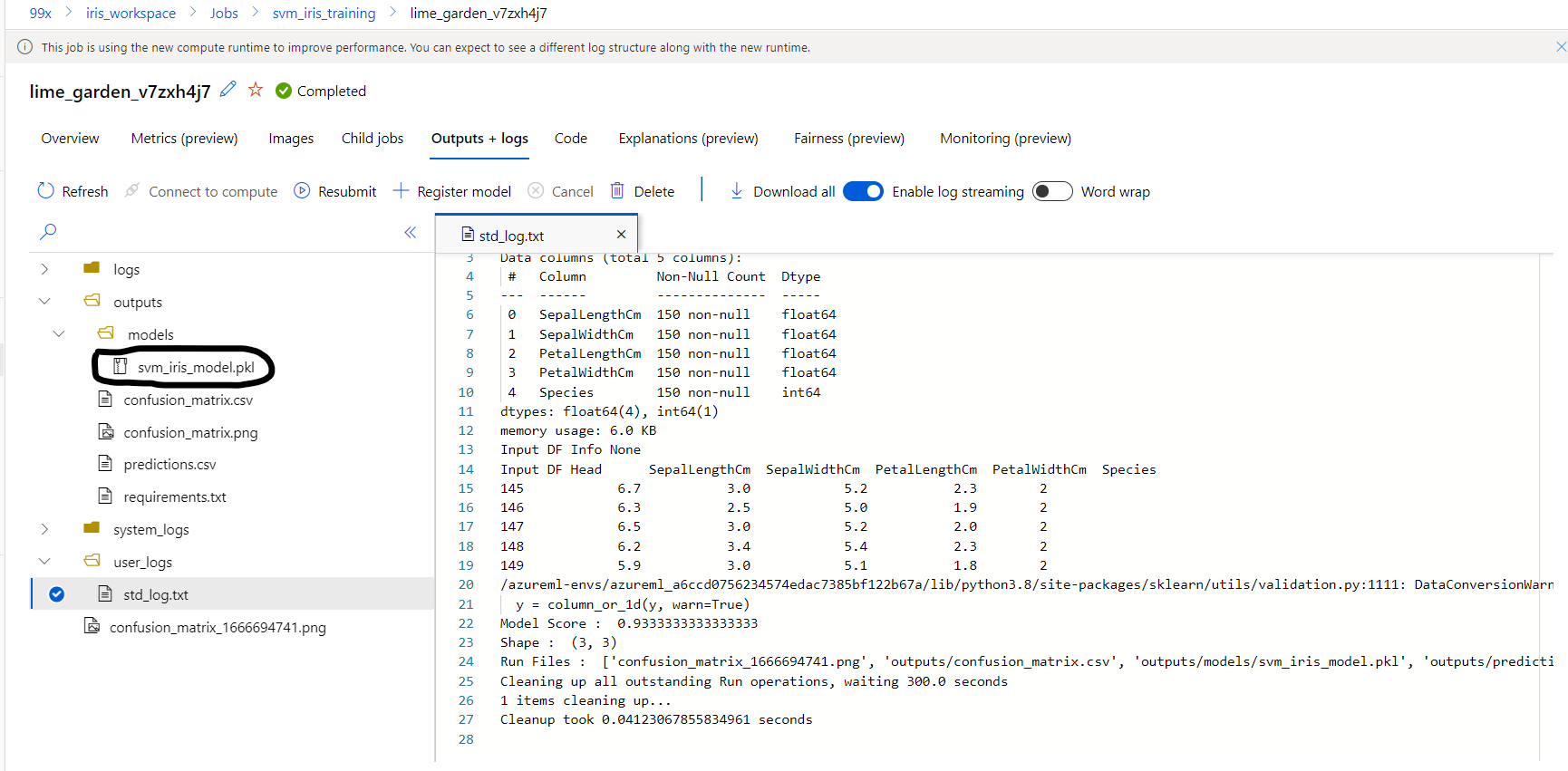
**Step 4**:

After Selecting the latest training Job, navigate to the “Outputs + logs” Section.



**Step 5**:

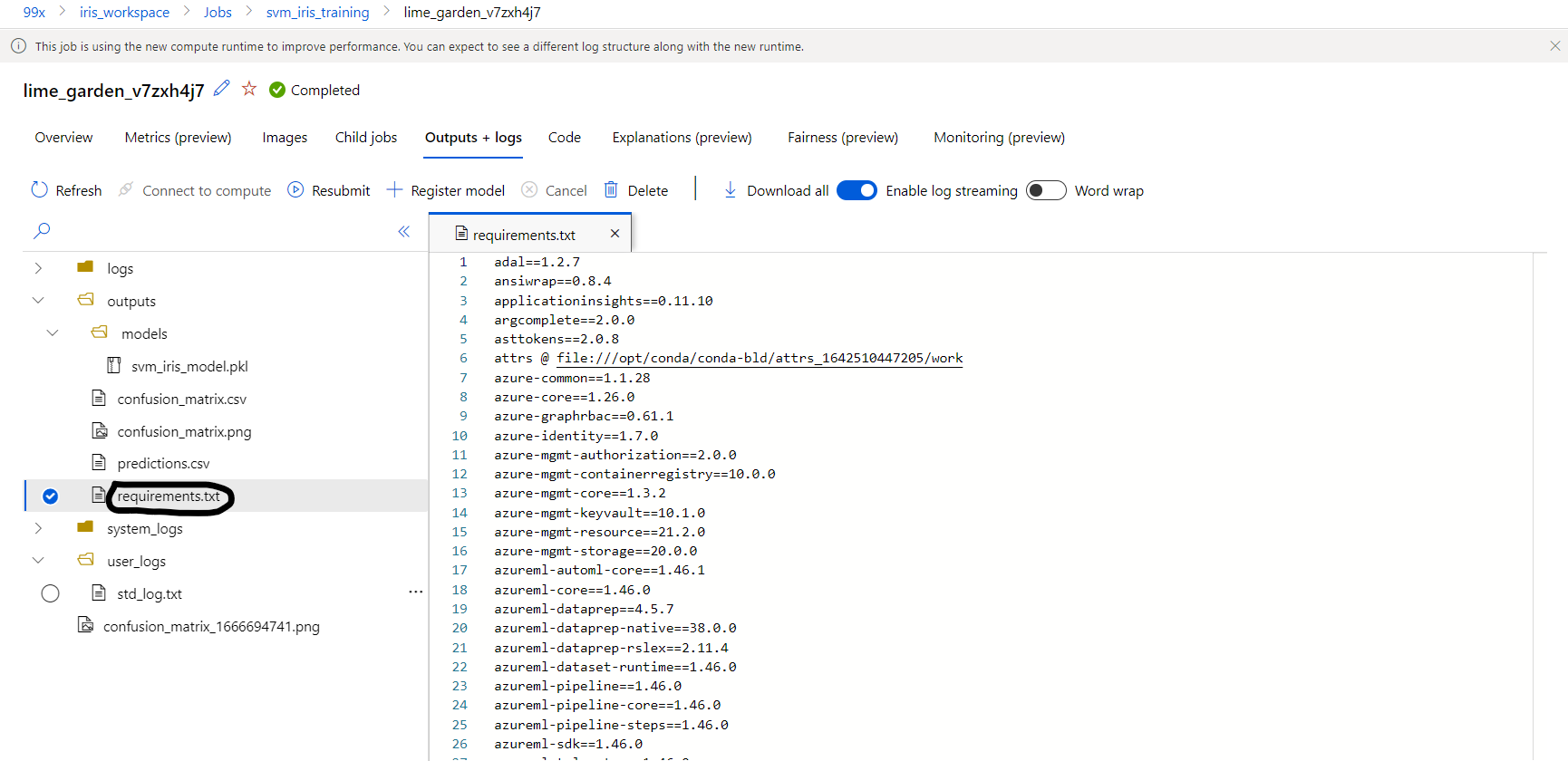
Inside “Outputs + logs”, go to the “outputs” folder and then the “models” folder. Download the **.pkl** or **.h5** file inside the folder.



This is a binary file where the model is stored. Right click and Download this file locally. We will upload this file to the “Notebooks” for testing purposes.

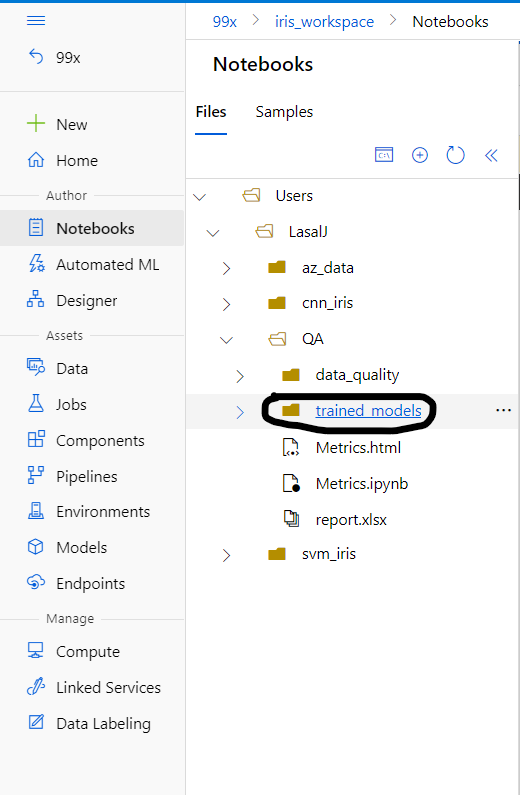
**Step 6**:

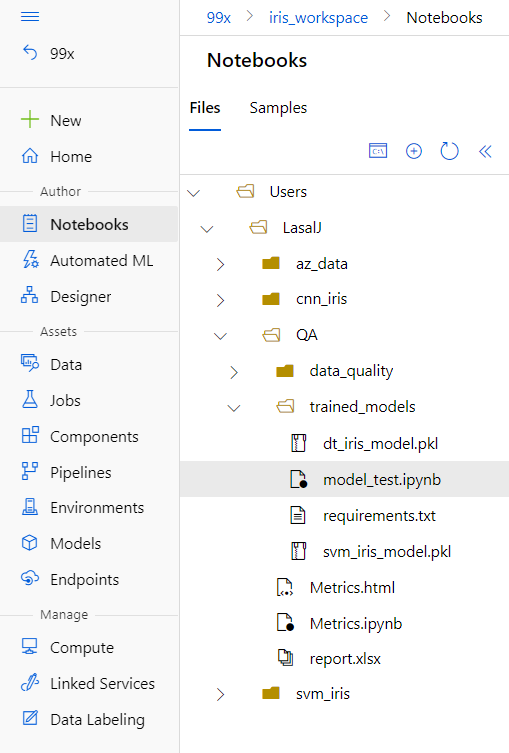
Download the **requirements.txt** file locally. This will be used to create a new kernel for model testing in the “Notebooks”.



**Step 7**:

Go to “Notebooks” in the Azure ML Workspace. Then go to the “QA” folder and inside the “trained\_models” folder, upload the **requirements.txt** file and the **.pkl** or **.h5** file inside the folder.





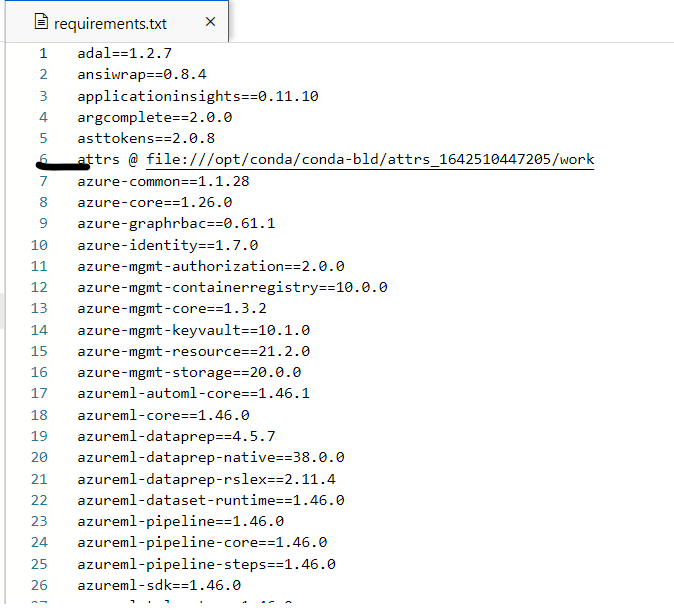
**Step 8**:

Creating a new Kernel for Model testing. It is recommended to create a separate kernel for each model.

Follow this [guide](https://medium.com/analytics-vidhya/how-to-create-virtual-environments-in-azure-ml-workspace-in-azure-portal-39245a34b370) to setup a new kernel.

At the step (step 7) where python is installed, make sure to install the python version used in the training job. By default, python 3.8 is used. Make sure to run **conda install python==3.8** or any other version used.

Before installing the packages in **requirements.txt** make sure to remove imports with “@” like below:



Finally, when it comes to installing other packages, run **conda install --file requirements.txt** to install packages used in the training job with the exact versions. If you do not use the exact versions, there may be errors when reading the binary files of the models.

**Step 9**:

Make sure to select the created kernel and run any manual experimentations on the model. Create separate kernels for each model, and make sure to update each environment before starting experimentations.

# **Data Quality and Drift**

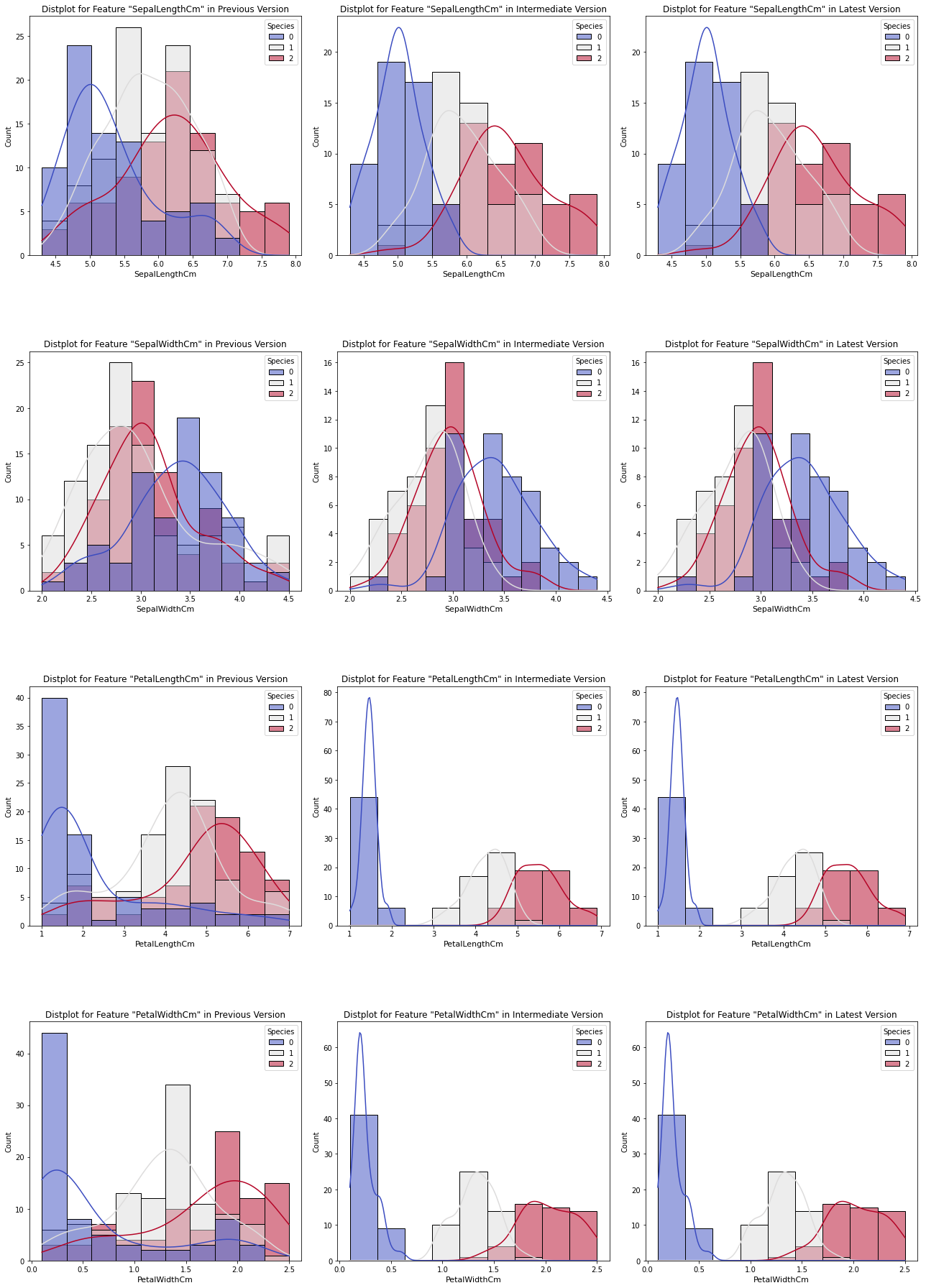
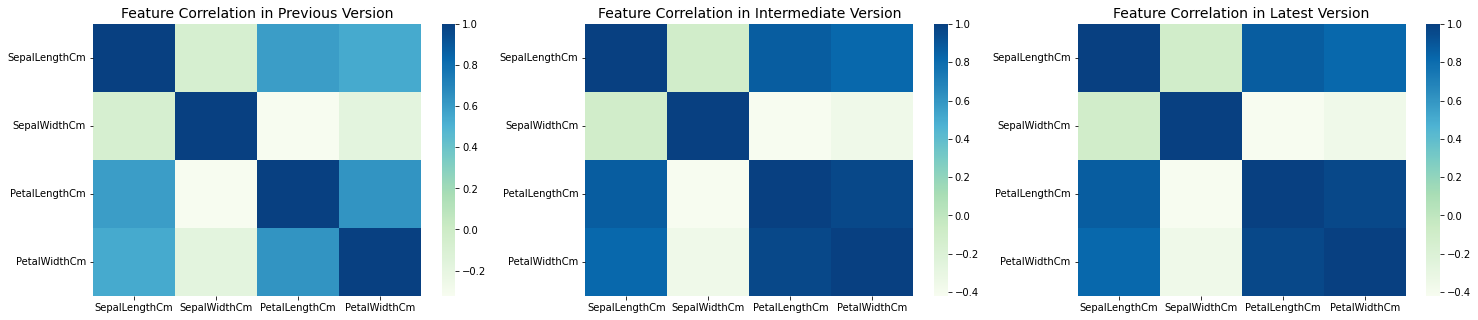
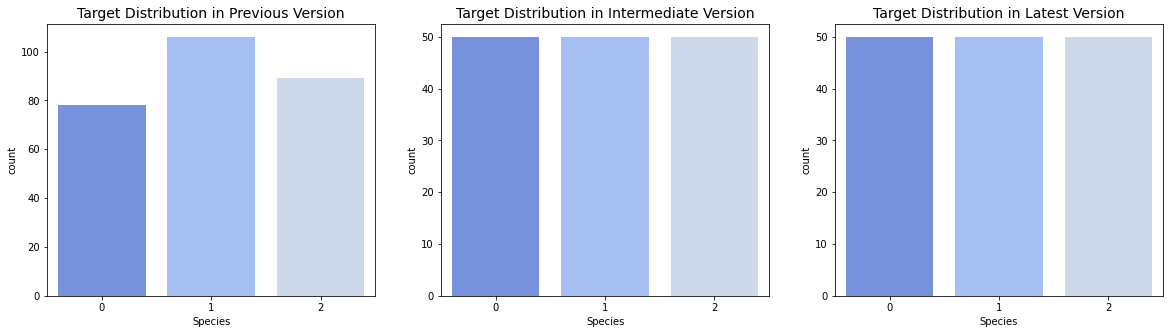
Data is the core of any Machine Learning Task. Therefore, Data Quality is a major concern, especially in production-grade projects. One of the major concerns in this context is **Data drift**.

**Data drift** is one of the top reasons model accuracy degrades over time. For machine learning models, data drift is the change in model input data that leads to model performance degradation. Monitoring data drift helps detect these model performance issues.

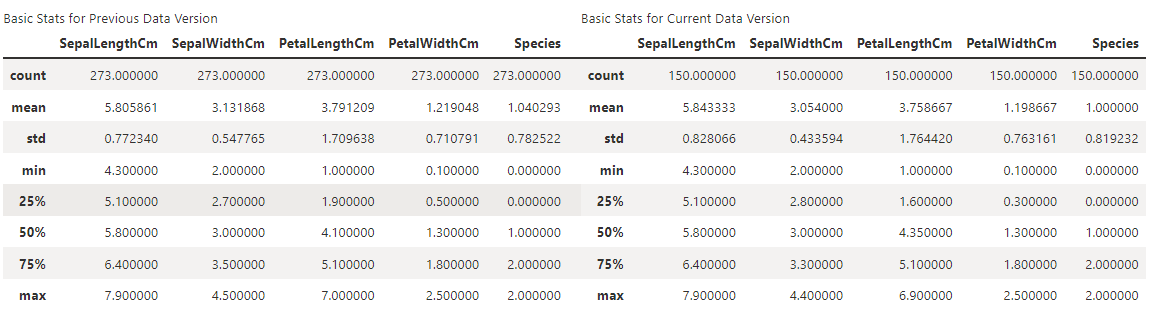
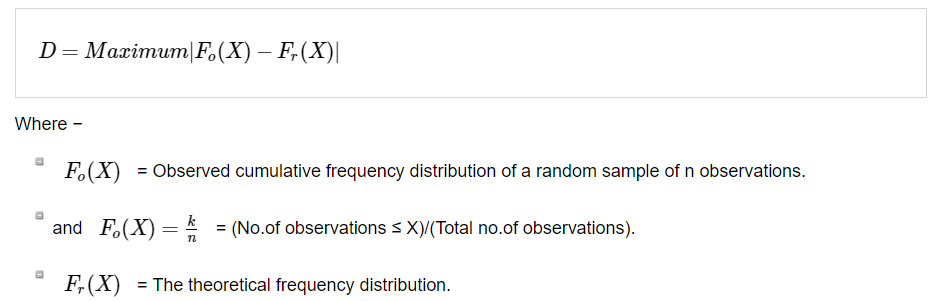
Causes of data drift include:

* Upstream process changes, such as a sensor being replaced that changes the units of measurement from inches to centimeters.
* Data quality issues, such as a broken sensor always reading 0.
* Natural drift in the data, such as mean temperature changing with the seasons.
* Change in relation between features, or covariate shift.

Visual Tools in the notebook:

* Visualizing Individual Feature Distribution:  
  A Distribution plot is constructed that depicts the variation in the feature distribution. Multiple data versions are compared to detect changes in drift for each feature.  
    
  The [Kernel Density estimates](https://en.wikipedia.org/wiki/Kernel_density_estimation) are also plotted for each distribution. This will give additional insight into the data. Follow this [guide](https://www.statology.org/density-curves/) to help understand how to interpret the Kernel Density Estimates.   
    
  The Plotted has been shaded according to the Categories of the Truth Labels, in this case, it is a species of plants. This categorization helps to drill down deeper to gain more insight as to how each feature has changed pertaining to each Truth Label.   
    
  Changes is feature distribution can change model performance and may require changes in model architecture.   
    
  
* Visualizing Relationships of each feature pair:  
    
  A Scatterplot is constructed for each feature pair within the dataset. Multiple data versions are compared to detect changes in each feature pair.  
    
  The plot shows how [Multicollinearity](https://www.investopedia.com/terms/m/multicollinearity.asp#:~:text=Multicollinearity%20is%20a%20statistical%20concept,in%20less%20reliable%20statistical%20inferences.) can change between data versions. Such changes in feature pairs can have a massive impact on the model performance.  
    
  
* Visualizing Correlation of features:  
    
  A Heatmap is plotted on top of the correlation matrix created from the dataset. Heatmaps can be compared to see how strong the relationship between each feature has changed.  
    
  Correlation is a statistical measure that expresses the extent to which two variables are linearly related. Correlation is likely to be more reliable and near to reality, that makes it a good starting point.  
  
* Visualizing Feature Importance:  
    
  Horizontal Barplots are generated which depict the importance of each feature in the dataset. The feature importance can be compared to see each feature’s distribution over time.  
    
  Feature Importance can help identify key contributing features. This can then be leveraged by further improving the feature or engineering additional features that enhance this variable.  
    
  The Feature Importance in the project is calculated by aggregating results from several algorithms to give out a robust measure.  
    
  The algorithms used are:  
  ⁍ XGB Classifier   
  ⁍ Extra Trees Classifier   
  ⁍ Decision Tree Classifier   
  ⁍ Gradient Boosting Classifier   
  ⁍ AdaBoost Classifier  
  ⁍ Random Forest Classifier  
    
  The aggregate value of these algorithms’ “feature\_importances\_” is used in the calculation.  
    
  
* Visualizing Target Distribution:  
    
  A Countplot is constructed for the target variable. A Countplot can be thought of as a histogram across a categorical, instead of quantitative, variable.  
    
  With plot we can see whether there are any class imbalances, changes in class distribution may require new metrics for training. This will also give a good idea on which class to prioritize data collection.  
    
  

Algorithms used to Detect Drift in Numerical Variables:

1. Basic Statistics (Mean, Max, Min, Std) -   
     
   The Basic Statistic gives a high-level overview of each feature, which will be useful for comparison.  
     
   The Mean is considered the center of the data. Many statistical analyses use the mean as a standard measure of the center of the distribution of the data.  
     
   The Maximum and Minimum helps identify the range of the feature.  
     
   Standard Deviation illustrates the spread of the data in the analysis.  
     
   The Basic Statistics can be compared to detect changes in the numerical features.  
     
   
2. Population Stability Index (PSI) -   
     
   The [Population Stability Index](https://www.listendata.com/2015/05/population-stability-index.html) is a drift metric for numerical and categorical features. It is often used in domains like finance.  
     
     
     
   The PSI is a number that can take any value from 0 and above. It also reflects the relative "size" of the drift: the larger the PSI value, the more different the distributions are.   
     
   As a rule of thumb,  
   ⁍ PSI < 0.1 is interpreted as no change   
   ⁍ 0.1 ≤ PSI < 0.2 is interpreted as a moderate change   
   ⁍ PSI ≥ 0.2 is interpreted as a significant change  
     
   PSI only starts detecting significant change for a drift size larger than 10%. When it comes to segment drift, PSI has the chance of detecting only major changes, such as the 100% shift in the data segment.  
     
   PSI has low sensitivity relative to other drift metrics. PSI is regarded as "predictable" as it returns the same result regardless of the sample size. The benefit of PSI is its interpretability.   
     
   PSI is recommended to be used in industries where the key scientist have significant domain knowledge. It might be helpful to rely on existing rules of thumb to define "drift size," especially for business stakeholders.   
     
   The Population Stability Index can be used when the project deals with a large volume of data and a stable metric is needed to detect "major changes."  
     
   The Population Stability Index can also be used in **Proactive Feature Selection**. When choosing features to go into a model, certain features may have a lot of predictive power at the time of training, but if a feature is prone to rapid changes in distribution, it may not be a wise decision to include it in the model or it may prompt more frequent monitoring once deployed. PSI is a straightforward way to check the volatility of population changes for features by comparing populations for several previous time periods.
3. Wasserstein Distance (Earth-Mover Distance) -   
     
   [Wasserstein distance (WSD)](https://en.wikipedia.org/wiki/Earth_mover%27s_distance) is applied only for numerical features.   
     
   **WSD Formula:**  
     
     
   Wasserstein distance shows the absolute value of data drift. WSD can be seen as the minimum amount of “work” required to transform *u* into *v*, where “work” is measured as the amount of distribution weight that must be moved, multiplied by the distance it must be moved. Roughly, it measures how much effort it takes to turn one distribution into another.   
     
   To give some intuition behind WSD: if drift happens in one direction (e.g., all values increase), the absolute value of the WSD metric often equals the difference of means. Even if the changes happen in both directions (some values increase, some values decrease), the WSD metric will sum them up to reflect the change. If the difference of means were used, then these changes would "cancel" each other. This makes WSD a more informative metric.  
     
   The WSD metric returns a value from 0 to infinity, making the degree of drift comparable between features. This also means that the Drift Threshold must be set with help of domain knowledge.  
     
   When the sample size is "small," WSD tends to overestimate the drift. The WSD metric tends to be more sensitive than PSI. Overall, it can be a good compromise between "way-too-sensitive" and "notice-only-big-changes".
4. Kolmogorov–Smirnov Test (K–S Test or KS Test)  
     
   The [Kolmogorov–Smirnov Test (KS Test)](https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test#:~:text=In%20statistics%2C%20the%20Kolmogorov%E2%80%93Smirnov,test)%2C%20or%20to%20compare%20two) is a nonparametric statistical test. This means it does not make any assumptions about the underlying distribution.  
     
   **KS Test Formula:**  
     
     
     
     
   The null hypothesis is that the two samples/features come from the same distribution. The KS Test is applied to reject or accept it. This allows us to detect whether the samples/features have changed i.e., a data drift has occurred.  
     
   The KS Test returns the p-value. If the p-value is less than 0.05, it can be declared that there is convincing evidence to reject the null hypothesis and consider the two samples/features different. A different significance level (p-value) can be set, for example, react only to p-values less than 0.01. It is good to remember that the p-value is not a "measurement" of drift size but a declaration of the statistical test significance.  
     
   Kolmogorov-Smirnov test is often used to detect drift in numerical features by default. One thing to consider is the sensitivity in the case of large datasets. The larger the dataset, the bigger the test's statistical power (sensitivity).   
     
   The KS Test helps to detect changes with pinpoint accuracy. The KS test is recommended to be conducted on fewer observations (e.g., under 1000). It is also a good fit when the data is expected to be stable and want to react even to a slight deviation inside a particular data segment. In the case where you need to apply the KS test on a larger dataset, it is important to take a data sample without sampling bias.

Algorithms used to Detect Drift in Categorical Variables:

There are no standard algorithms available that are used to detect data drift in categorical features.   
  
There are two ways that are currently used for ball parking the data drift effect:

1. Cardinality -   
   The number of unique values of the feature.
2. Euclidian distance -  
   Euclidean distance is computed on two vectors, generated from empirical distribution of the same categorical column from two datasets. 0 indicates there is no difference in the empirical distributions. The more it deviates from 0, the more this column has drifted. Trends can be observed from a time series plot of this metric and can be helpful in uncovering a drifting feature.

The above methods were implemented in the “Data Quality” section in the Azure ML Workspace.

You could also checkout Azure’s Beta Data Drift Detection functionality [here](https://learn.microsoft.com/en-us/azure/machine-learning/v1/how-to-monitor-datasets?tabs=python).

The main advantage to developing an In-House Data Drift Detection is the customizability. Since Data Drift in practice is yet a very new concept it best to opt for In-House solution at the moment.

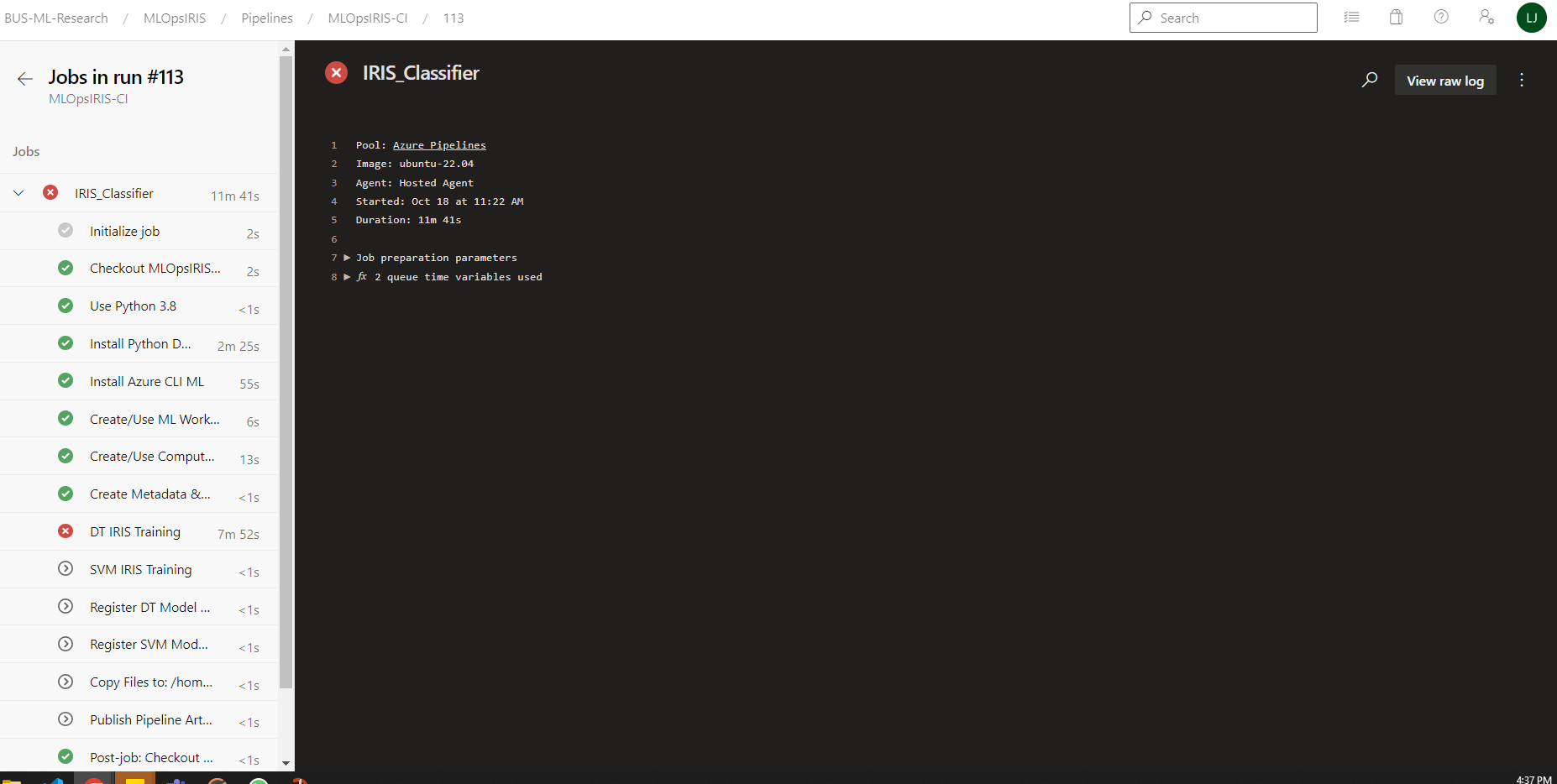
# **Challenges to Anticipate**

# **Integrating a New ML Model**

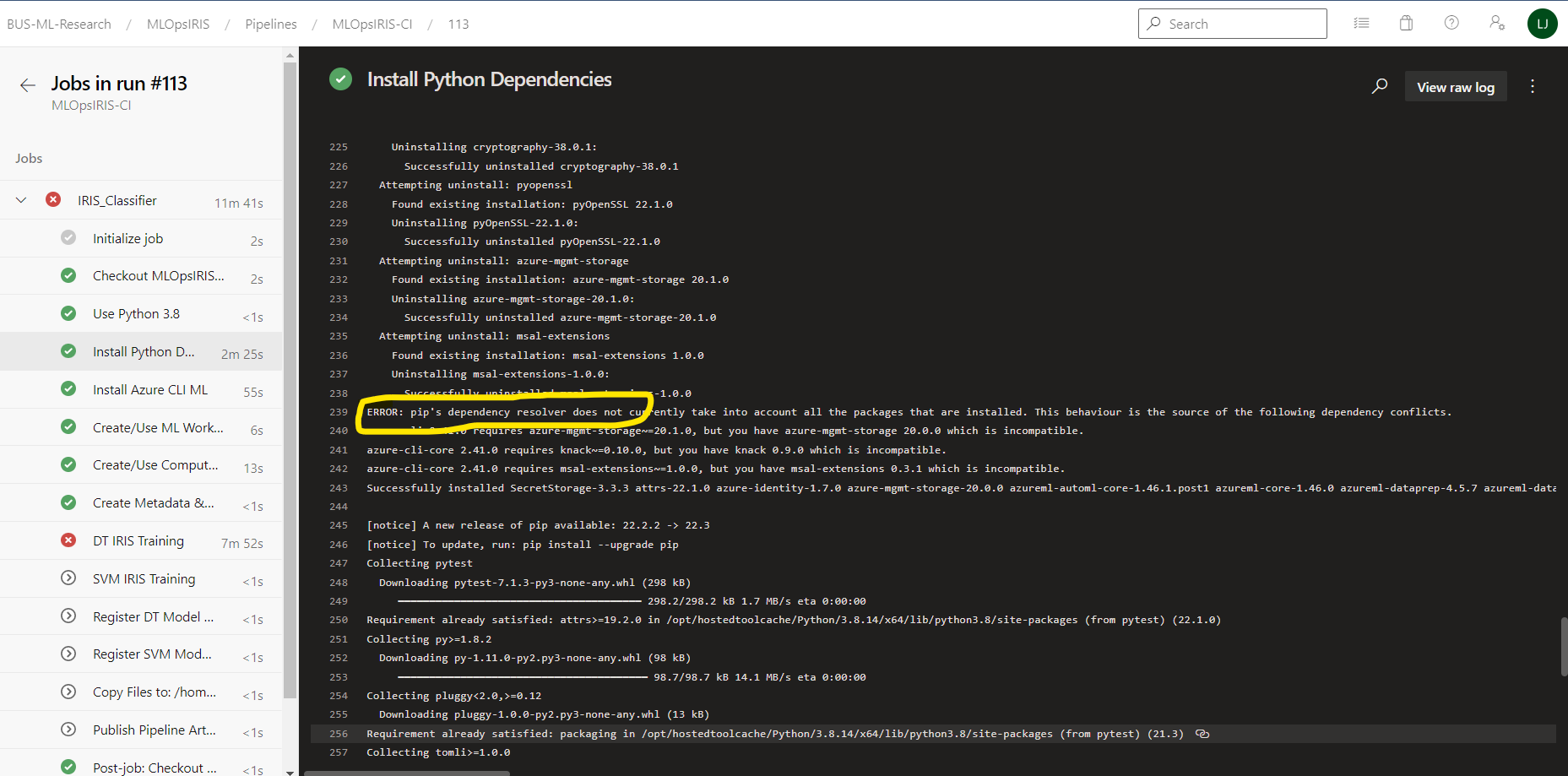
When incorporating a new ML model into the pipeline, the most difficult part is figuring out the package dependency issue.

Most of the time the pipeline will fail due to this issue and it may not be straight forward.

Below the Error is raised at the Model Training Step:



When diving deeper it is evident that the issue is at the package installation.



The best way to mitigate this issue is using a custom docker environment which will be incorporated in time to come.

Sometimes the dependency resolver will take more than 30 minutes which will cause a timeout and the pipeline to fail. This again is due to the versions of the libraries installed.

# **Data Storage in Azure Blob**

{Dilan}

When using azure blob, we have to enable versioning in the data protection section of the storage account the blob is saved in.

For more information follow the steps in the azure blob versioning documentation. ([Enable and manage blob versioning - Azure Storage | Microsoft Learn](https://learn.microsoft.com/en-us/azure/storage/blobs/versioning-enable?tabs=portal))

# **Additional Tools**

# **Triggering the Model Training Pipeline when there is a change in Data**

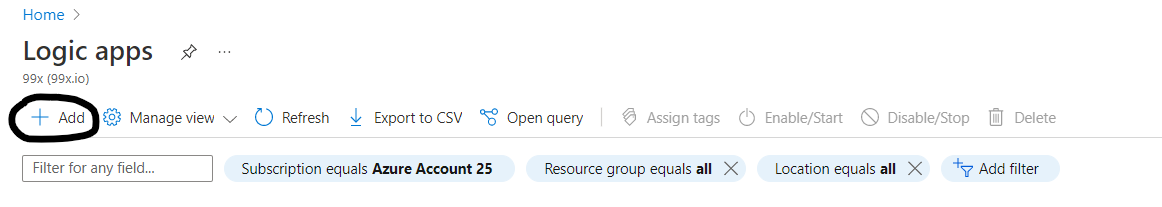
Automatically triggering the pipeline when there is a data change can be extremely useful and further automates the entire end-to-end ML process. The system below is implemented using Azure Logic Apps.

To give a simple overview, the logic app will periodically check whether there has been any change to the data source, in this case a blob container, and then trigger the Model Training Pipeline in Azure Devops. The best part is that, the time duration to check the change in the data can be seconds or days. This replaces the need for a human to manually to the triggering process.

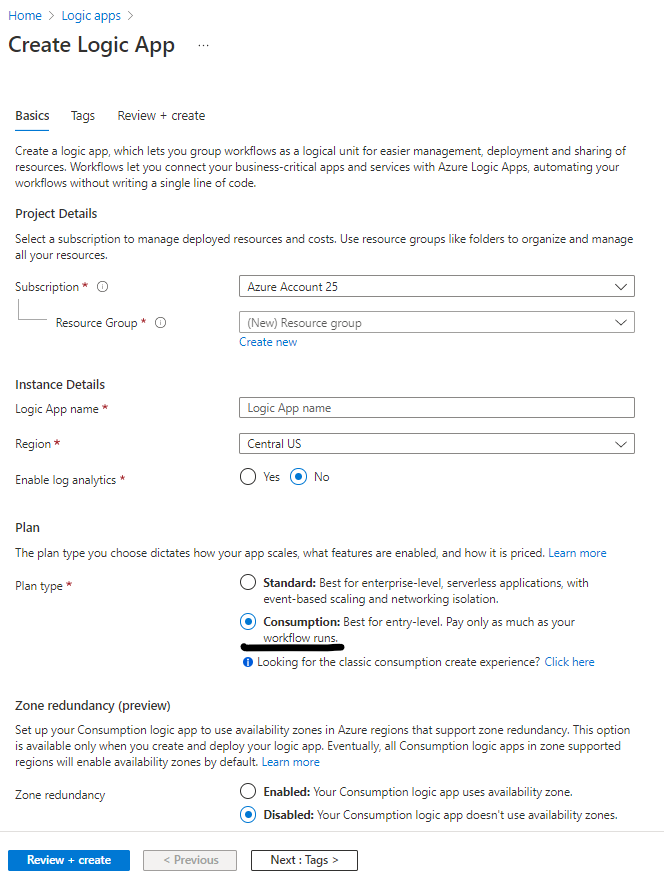
Steps to build the Logic App.

**Step 1:**  
Go to Azure Logic Apps.

**Step 2:**  
Create a new Logic App by selecting the Add button.



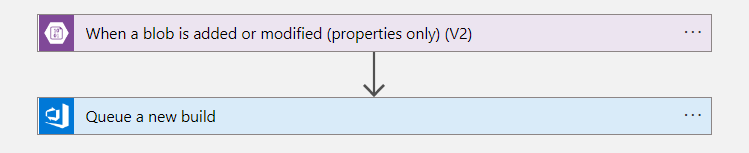
**Step 3:**  
Create a Logic App with a Consumption plan. Fill the other detail as required.



**Step 4:**

Go to Logic App Designer to build the app.

This is the structure of the Logic App

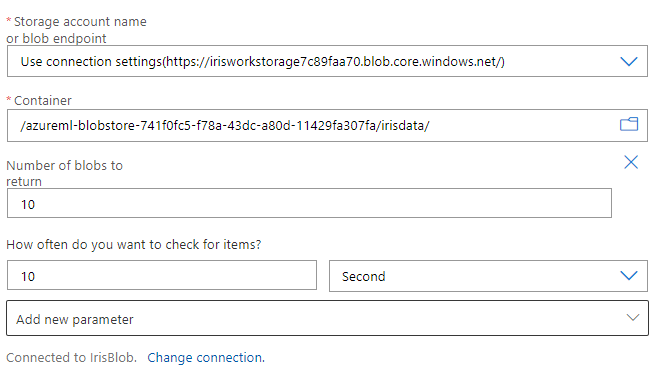


**Step 5:**

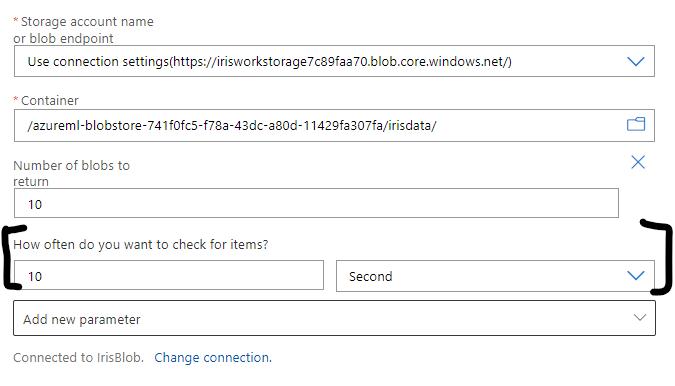
Click on “New Step” inside the Logic App Designer and create a “When a blob is added or modified” operation.



Setup blob is blob endpoint and Container as per the current project destinations.



Change the duration where the delta of the data must be checked. You can configure it to run every few seconds all the way to several months.

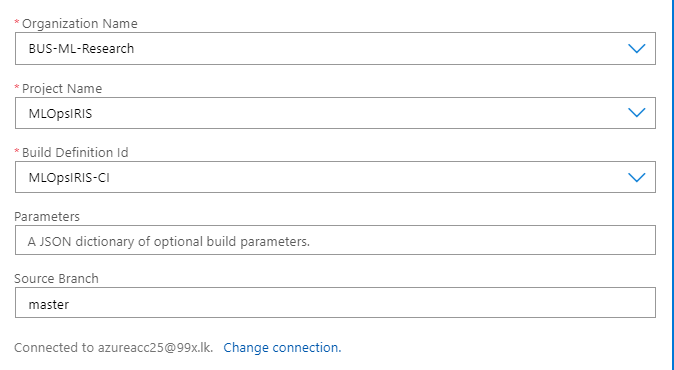


**Step 6:**

Click on “New Step” inside the Logic App Designer and create a “Queue a new build” operation.



Setup the details pertaining to which branch to trigger. Also make sure to use a “connection” which has permission to access the Devops pipeline.



**Step 7:**

Reference of the code view.

{

"definition": {

"$schema": "https://schema.management.azure.com/providers/Microsoft.Logic/schemas/2016-06-01/workflowdefinition.json#",

"actions": {

"Queue\_a\_new\_build": {

"inputs": {

"body": {

"sourceBranch": "master"

},

"host": {

"connection": {

"name": "@parameters('$connections')['visualstudioteamservices']['connectionId']"

}

},

"method": "post",

"path": "/@{encodeURIComponent('MLOpsIRIS')}/\_apis/build/builds",

"queries": {

"account": "BUS-ML-Research",

"buildDefId": "1"

}

},

"runAfter": {},

"type": "ApiConnection"

}

},

"contentVersion": "1.0.0.0",

"outputs": {},

"parameters": {

"$connections": {

"defaultValue": {},

"type": "Object"

}

},

"triggers": {

"When\_a\_blob\_is\_added\_or\_modified\_(properties\_only)\_(V2)": {

"evaluatedRecurrence": {

"frequency": "Second",

"interval": 10

},

"inputs": {

"host": {

"connection": {

"name": "@parameters('$connections')['azureblob\_1']['connectionId']"

}

},

"method": "get",

"path": "/v2/datasets/@{encodeURIComponent(encodeURIComponent('AccountNameFromSettings'))}/triggers/batch/onupdatedfile",

"queries": {

"checkBothCreatedAndModifiedDateTime": false,

"folderId": "JTJmYXp1cmVtbC1ibG9ic3RvcmUtNzQxZjBmYzUtZjc4YS00M2RjLWE4MGQtMTE0MjlmYTMwN2ZhJTJmaXJpc2RhdGElMmY=",

"maxFileCount": 10

}

},

"metadata": {

"JTJmYXp1cmVtbC1ibG9ic3RvcmUtNzQxZjBmYzUtZjc4YS00M2RjLWE4MGQtMTE0MjlmYTMwN2ZhJTJmaXJpc2RhdGElMmY=": "/azureml-blobstore-741f0fc5-f78a-43dc-a80d-11429fa307fa/irisdata/"

},

"recurrence": {

"frequency": "Second",

"interval": 10

},

"splitOn": "@triggerBody()",

"type": "ApiConnection"

}

}

},

"parameters": {

"$connections": {

"value": {

"azureblob\_1": {

"connectionId": "/subscriptions/3890e68c-f48e-4dc7-9d83-07c6193b69e1/resourceGroups/Bus-MlOps\_group/providers/Microsoft.Web/connections/azureblob-1",

"connectionName": "azureblob-1",

"id": "/subscriptions/3890e68c-f48e-4dc7-9d83-07c6193b69e1/providers/Microsoft.Web/locations/eastus/managedApis/azureblob"

},

"visualstudioteamservices": {

"connectionId": "/subscriptions/3890e68c-f48e-4dc7-9d83-07c6193b69e1/resourceGroups/Bus-MlOps\_group/providers/Microsoft.Web/connections/visualstudioteamservices-8",

"connectionName": "visualstudioteamservices-8",

"id": "/subscriptions/3890e68c-f48e-4dc7-9d83-07c6193b69e1/providers/Microsoft.Web/locations/eastus/managedApis/visualstudioteamservices"

}

}

}

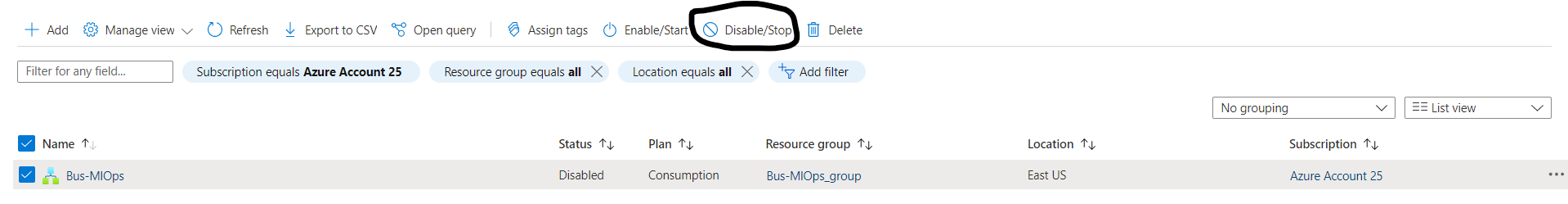
}

}

**Step 8:**

Now press Save and Run the Logic App. You could also test by manually “running the trigger”.

Make sure to Stop the Logic App if not in use to not accumulate any additional cost.



# **Future Improvements and Extensions**

# **Concept Drift**

Adding Concept drift is a practical way to expand the current architecture.

In predictive analytics and machine learning, concept drift means that the statistical properties of the target variable, which the model is trying to predict, change over time in unforeseen ways. This causes problems because the predictions become less accurate as time passes.  
  
The cause of the relationship change is some kind of external event or process. For example, we try to predict life expectancy using geographic regions as input. As the region’s development level increases (or decreases) region loses its predictive power, and our model degrades.

Adding a mechanism to detect concept drift depends on the project requirement is a feature that can save a lot of time and money in case of a Concept drift.

# **Robust Way to Handle Package Dependencies (Coming Soon)**

Currently the Packages needed for Model training are specified in a dependency file. This approach is not stable and some packages would not be resolved in the job’s pip resolver. At the current stage there is an issue installing TensorFlow.

In the next MLOps architecture Release Custom Environments will be used and integrated to the Training Jobs. This way the packages are installed in a docker image which will then be used by the Training Job.

# **Mechanism to Check Whether the Data is Suitable Before Model Training (Coming Soon)**

Currently the pipeline runs all the Model training Job and then in CD pipeline before Model deployment the QA or a particular developer can give the necessary approval to continue with the Model Deployment.

Even though this is a good enough solution, in the next iteration of the MLOps architecture, there will be mechanism where within the CI pipeline certain statistical tests will be run on the Data to see whether its “fit” or suitable for Model training. If not, there will be a manual override facility for a developer to continue with model training process. If the developer does not approve the Data, then the Model training will not take place.

# **Better Model Training Architecture (Coming Soon)**

All the current Model training processes that take place by in the pipeline runs from a python script. In the next iteration of the MLOps architecture, more model training methods will be analyzed and compared. For example, is it beneficial to use pipelines with Azure ML studio or Databricks.

# **Expanding to Other ML Tasks (Coming Soon)**

This project is currently based on simple classification task. The project will be extended to handle regression, Computer Vision, and NLP related tasks.