1. Do the model iterations and finalize all the steps for data preprocessing and model building.
2. Divide your script into three functions:
   1. Data preprocessing and train/test split
   2. Model training with hyper-parameters
   3. Get Model metrics

Project\_folder\_structure:

1. First set up azure environment:

‘Environment\_setup’ folder has 2 files:

1. cloud-environment.json: No input given. It has all the items to be created in azure ml workspace along with the name
2. iac-create-environment-pipeline-arm.yml: No changes made. It takes following inputs to define names in azure ml workspace. This should be given as pipeline variables with same group name as ‘mlops-wsh-vg’:
   1. AZURE\_RM\_SVC\_CONNECTION
   2. LOCATION
   3. RESOURCE\_GROUP
   4. BASE\_NAME
   5. WORKSPACE\_NAME
   6. AZURE\_ML\_WORKSPACE\_CONNECTION
3. package\_requirement: Modify the requirements.txt accordingly.
4. Data: keep the data for training the model
5. [project\_param.yaml](http://localhost:8888/edit/Documents/my_deplyment_learning/titanic_model_MLOPS/project_param.yaml): change parameters here according to project. These will be used in train\_aml.py file. [For now, have to keep this file in both training and deployment folder. Have to find a way to read this file from source path]
6. Training the model:
   1. conda\_dependencies.yml: The dependencies defined in this file will be automatically provisioned for managed runs. These include runs against the localdocker, remotedocker, and cluster compute targets. Just check once if all required packages are mentioned in it.
   2. Train.py: define Data preprocessing and train/test split, Model training with hyper-parameters and Get Model metrics functions here according to your business problem.
   3. train\_test.py: modify this file according to the function defined in tain.py file to run the unit testing of these functions.
   4. parameters.json: hyper tuning parameters of the model. Make sure that parameters and their string values are enclosed in double quotes
   5. train\_config.runconfig : it is run configuration file. It has all details about env, history/run logs and data reference. Replace 'DATA\_FOLDER\_NAME' with the name of folder created in datastore in Azure while uploading the data. E.g.: titatnic
   6. train\_aml.py: It registers the dataset from the uploaded one in azureblobstorage and create version. Run the model and save everything in output folder along with logging things at run. No need to change anything here.
7. Deployment:
   1. [aciDeploymentConfigStaging.yml](http://localhost:8888/edit/Documents/my_deplyment_learning/titanic_model_MLOPS/deployment/aciDeploymentConfigStaging.yml): req for staging VM machine. No change
   2. [aksDeploymentConfigProd.yml](http://localhost:8888/edit/Documents/my_deplyment_learning/titanic_model_MLOPS/deployment/aksDeploymentConfigProd.yml): req. for prod incase using kubernet. No change needed.
   3. inferenceConfig.yml: used for deployment. Havepath to scoreing file and its configuration
   4. scoringConfig.yml: similar to conda\_dependecy file. Defining the env
   5. score.py: it acts like fastAPI file which will be used for model testing/scoring. No changes needed.
8. Test staging/ production:
   1. conftest.py: have scoreurl and scorekey to pass on to stag\_test and prod\_test.py files. No changes needed.
   2. Stage\_test.py: this will test if score.py function in stage env. Same thing for prod\_test in prod env. Given the relevant dataset input.

**Create CI-CD pipielines automatically using yaml files:**

1. Create azure environment using ‘iac.yaml’ file mentioned in environment\_setup folder. Just create the pipeline using yaml file, define variables and create azure service connection key. Detailed steps are mentioned in manual steps [1] below.
2. Create CI pipeline using Mlops-CI.yml file. Please make sure all the variables listed in this file at top are predefined in the pipeline and azure environment variables are same as defined in step 1.
3. Create CD pipeline using: Mlops-CD.json. Import this file in Azure devops release then modify:
   1. Update the artifacts to be connected with the output of CI pipeline
   2. Define Agent parameters like pool and machine to run the pipeline
   3. Update pipeline variables
   4. Make sure working directory selected in each task is pointed to correct artefact.

**Model Drift/ Data-drift checking:**

Create a notebook in same ML studio env where model is deployed. Consume this model using the rest api end-points code mentioned in endpoint tab in env. Pass the test data and evaluate the model performance on timely basis. This notebook can be scheduled automatically and results can be emailed as well. See model-drift file. Do the prediction for test data and track the model performance over time with new test data.

You can also create profile in the dataset in ML studio env and then compare different version of registered dataset to track the changes.

Resource: <https://www.youtube.com/watch?v=Fhzad9HkPws>

Some tips:

Cusomize your pipelines:

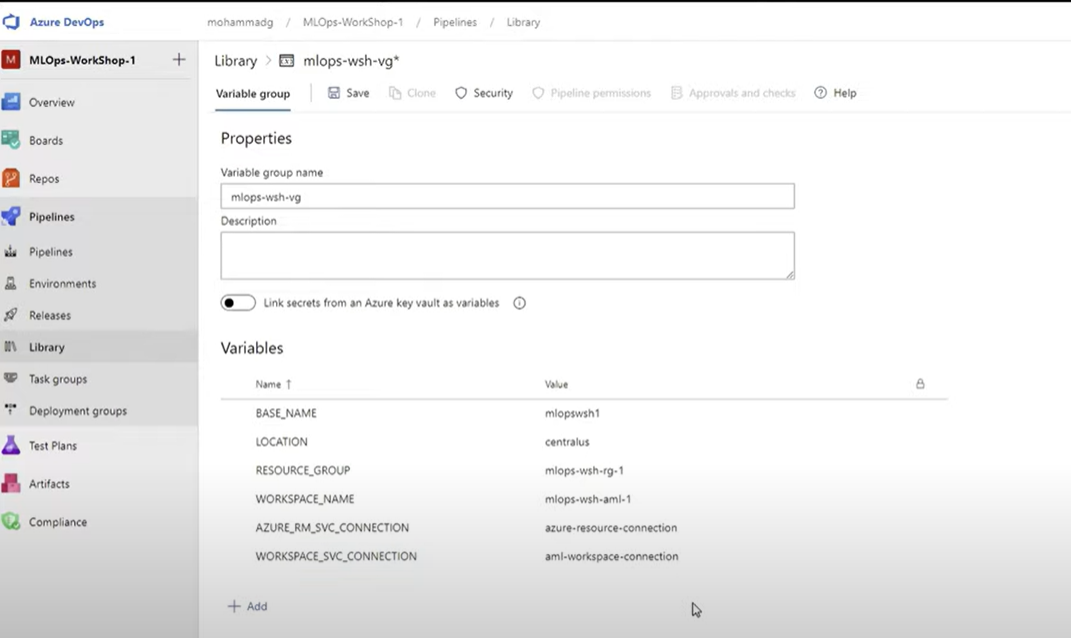
<https://learn.microsoft.com/en-us/azure/devops/pipelines/customize-pipeline?view=azure-devops>

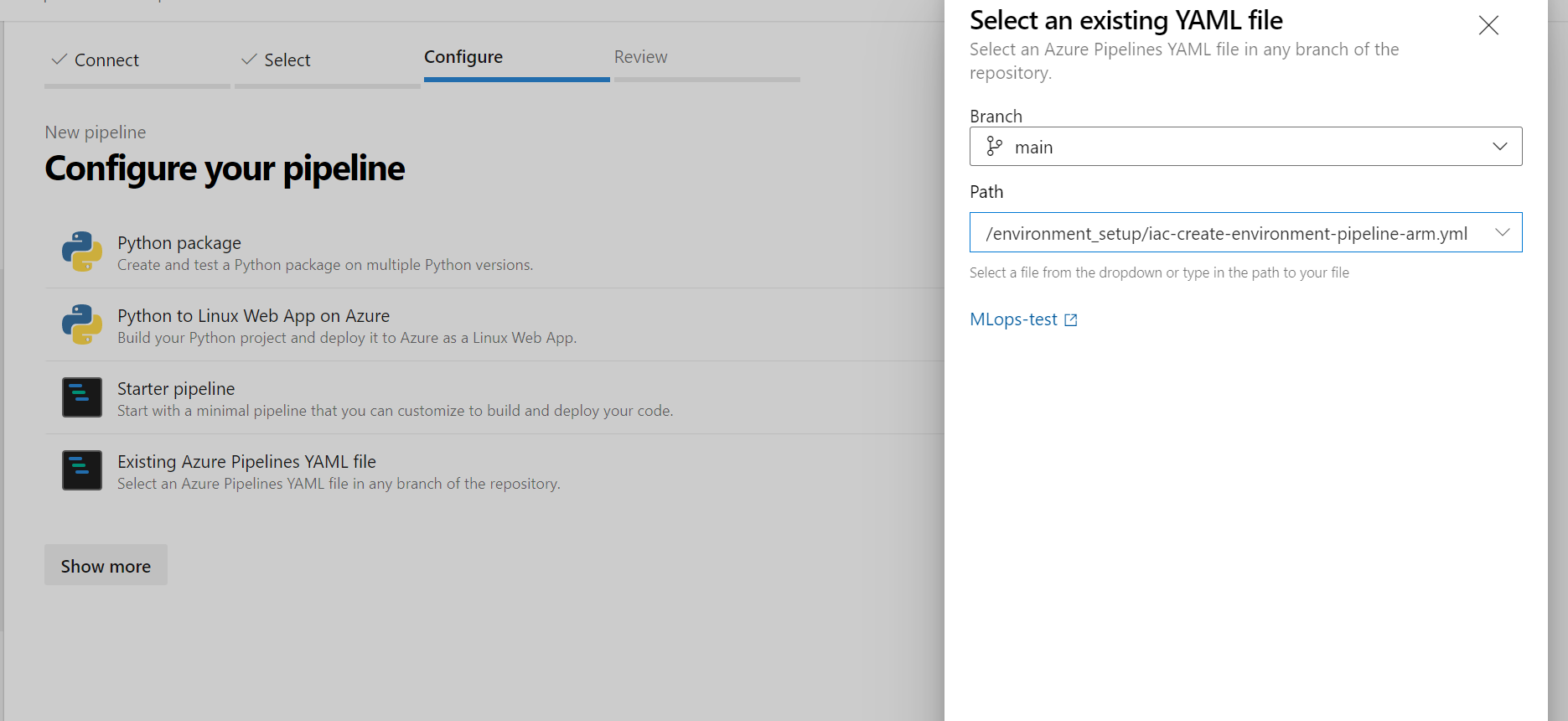
debug your logs during deployment while logging into azure portal. Run azure cli and then try:

az ml online-deployment get-logs --resource-group titanic-rg-v1 --workspace-name titanic-aml-v1 -e titatince-aci -n titatince-aci -l 100

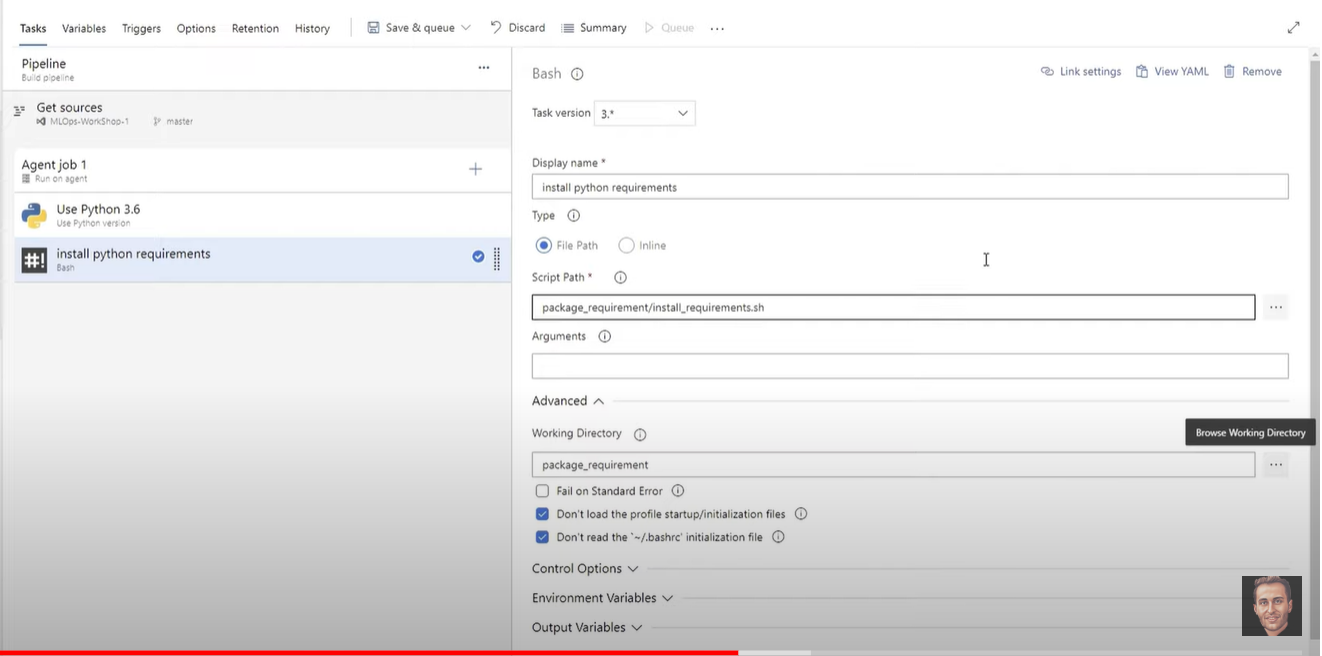
**Create CI-CD pipielines manually:**

Now push this repo to Azure repo in a new project. and create pipeline for Azure ML environment:

1. Create the infrastructure/ environment in azure ML workspace using the code. To create certain variables like env name, passwords etc go to pipelines🡪 library🡪 create variable groups and all variables. You can also add passwords and connect to azure key vault. You have to create the connection using service connection in project setting b/w project in devops to ML resource group. 
2. Get the code directly from git or local system. You can push it to azure repo. Now create infrastructure using yaml file iac[ infrastructure as code] using a pipeline. Here you can use a file **iac-create-environment-pipeline-arm.yml [this files uses variables we defined in step2]** or UI also. Make sure you have same variable group name as mentioned in yaml file. After running this successfully, ML workspace resourgroup will be created automatically in ML workspace. Best way to work is develop code locally then clone in to azure repo directly from local machine and create a different branch and merge with main once finalized.

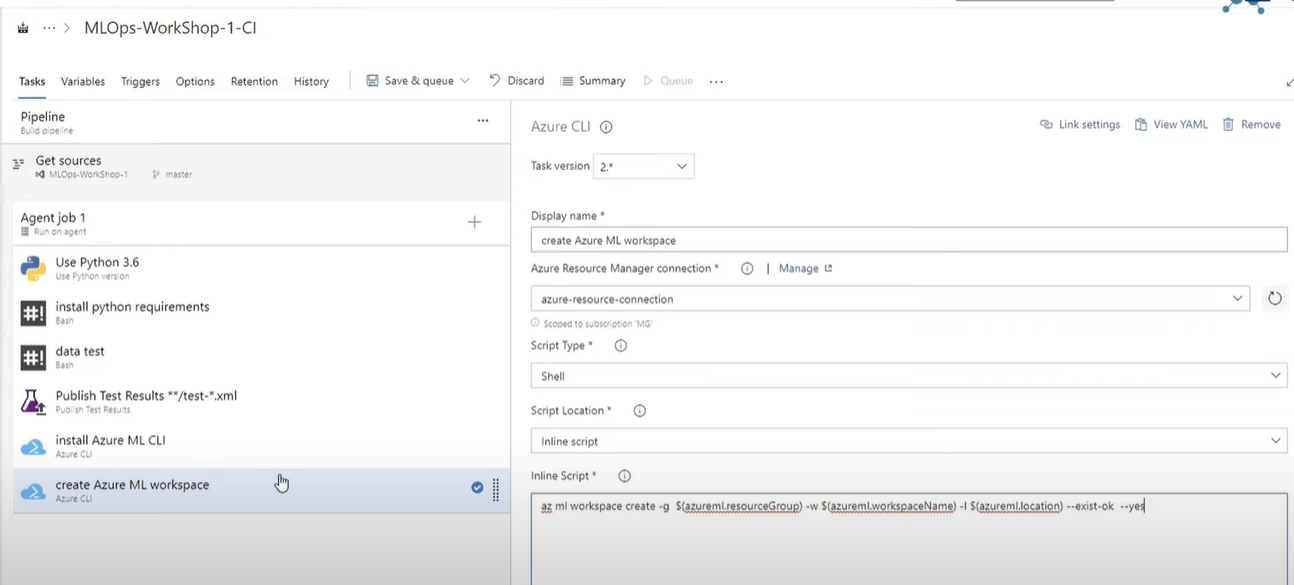


1. Once the environment is set. Now Create dataset using datastore or local m/c. Profile this data which will give statistics abt data.
2. Create a new pipeline to select repo and branch you want to run . first select agent with Ubuntu-latest and create empty job and then add task one by one like python version, install library etc. you can select agent specification and trigger enable CI as anything changes in master this CI will run automatically. this pipeline will execute the whole code and will be triggered if anything changes. Pipeline in step 3 creates only environment as we selected only env creation yml file. This can be created using UI: like running bash cmd task and passing the cmd either using file path or inline.



* For running train\_test.py file ‘pytest training/train\_test.py --doctest-modules --junitxml=junit/test-results.xml --cov=data\_test --cov-report=xml --cov-report=html’
* Publish these test results where file can be mentioned as ‘\*\*/test-\*.xml’. change control option as run this even if previous steps failed to debug

1. Create a new service connection key in devops for MLworkspace so that you can connect to ml workspace separately and install azure-cli so that steps like creating compute and stuff can be done using cli commands as below from devops.
2. Run this in cmd to install azure ml using cmd. ‘az extension add -n azure-cli-ml’



* 1. To create azure ml workspace: ‘az ml workspace create -g $(azureml.resourceGroup) -w $(azureml.workspaceName) -l $(azureml.location) --exist-ok –yes’ and defined these variables in the pipeline.
  2. To create compute : az ml computetarget create amlcompute -g $(azureml.resourceGroup) -w $(azureml.workspaceName) -n $(amlcompute.clusterName) -s $(amlcomput.vmSize) --min-nodes $(amlcomput.minNodes) --max-nodes $(amlcomput.maxNodes) --idle-seconds-before-scaledown $(amlcompute.idleSecondsBeforeScaledown)
  3. Upload the data to datastore/ blob storage so that it can be tracked.:

az ml datastore upload -w $(azureml.workspaceName) -g $(azureml.resourceGroup) -n $(az ml datastore show-default -w $(azureml.workspaceName) -g $(azureml.resourceGroup) --query name -o tsv) -p data -u titanic --overwrite true . This will push your file from data folder to datablob default storage. However, creation of dataset will happen when you register data in train\_aml.py file.

Here titanic is folder name which will be created in blob storage and it is copied from data folder in azure repo

1. Make metadata and model directory using bash cmd: ‘mkdir metadata && mkdir models’
2. Train the model using azure cli: ‘az ml run submit-script -g $(azureml.resourceGroup) -w $(azureml.workspaceName) -e $(experiment.Name) --ct $(amlcompute.clusterName) -d conda\_dependencies.yml -c train\_config -t ../metadata/run.json train\_aml.py’. give working directory in this as this refer to train\_aml.py file and add variable name like experiment name etc. [-c Name (without extension) of a run configuration file. The file should be in a sub-folder of the directory specified by the path parameter, -t: o/p data file, --ct compute target to run and in last user script] <https://learn.microsoft.com/en-us/cli/azure/ml(v1)/run?view=azure-cli-latest#az-ml(v1)-run-submit-script>
3. Register the model using azure cli: ‘az ml model register -g $(azureml.resourceGroup) -w $(azureml.workspaceName) -n $(model.Name) -f metadata/run.json --asset-path outputs/models/titanic\_model.pkl -d "Classification model for Survival prediction" --tag "data"="titanic" --tag "model"="classification" --model-framework ScikitLearn -t metadata/model.json [ --asset-path:The cloud path where the experiement run stores the model file, -d: model descrition, **--model-framework:** Framework of the model to register. Currently supported frameworks: TensorFlow, ScikitLearn, Onnx, Custom, Multi, -t: Path to a JSON file where model registration metadata will be written. Used as input for model deployment, --tag: Key/value tag to add (e.g. key=value )]. These metadata and model folder are currently in buildSourceDirectory and need to be copied to save in artefact folder.
4. Download mode: az ml model download -g $(azureml.resourceGroup) -w $(azureml.workspaceName) -i$(jq -r .modelId metadata/model.json) -t ./models --overwrite [-i: model id, -t: target dir]
5. Copy file from sources to staging directory using copy files. Source folder: $(Build.SourcesDirectory) target fold: $(Build.ArtifactStagingDirectory) Contents:

\*\*/deployment/\*

\*\*/tests/integration/\*

\*\*/package\_requirement/\*

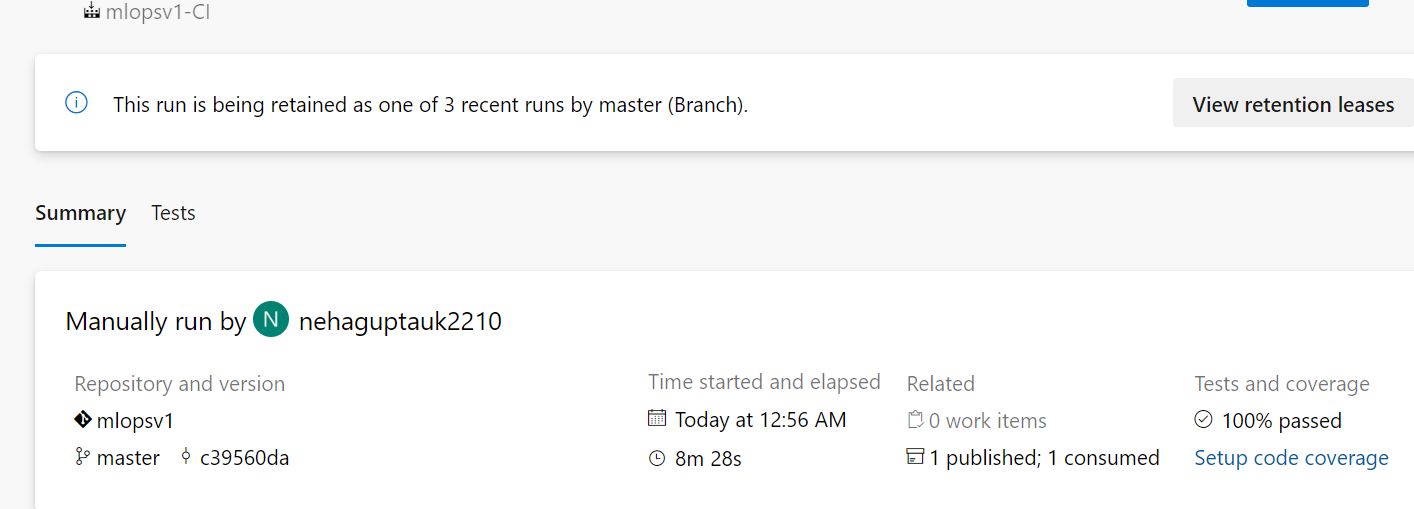
\*\*/models/\*

\*\*/training/\*

\*\*/metadata/\* \*\*/deployment/\*

1. Publish pipeline artifacts to: $(Build.ArtifactStagingDirectory)

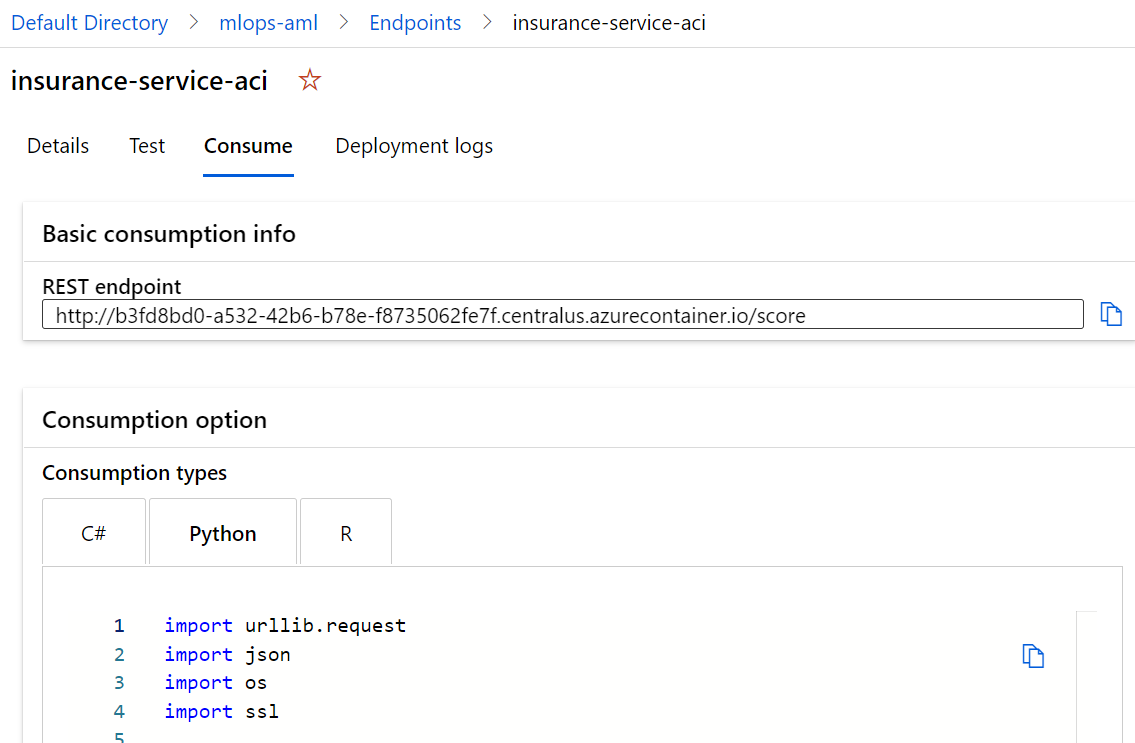
You can see published artifacts here which can be used for deployment:



Search here <https://learn.microsoft.com/en-us/cli/azure/ml(v1)/> for azure cli commands

Now create the CD pipeline which will deploy models.

1. Create a new release pipeline and add artifacts from CI pipeline. Then create empty job in agent and start adding tasks.
2. Use python version same as CI and add azure ml CLI as az extension add -n azure-cli-ml’.
3. Deploy into azure container instance for staging. If it works fine then deploy using kubernets in production. Use azure cli: az ml model deploy -g $(azureml.resourceGroup) -w $(azureml.workspaceName) -n $(service.name.staging) -f ../metadata/model.json --dc aciDeploymentConfigStaging.yml --ic inferenceConfig.yml --overwrite and also add working directory as landing/deplyment folder created in CI which has all yml files.
4. Now release this pipeline. After it is successfully ran, end points are created in azure ml workspaces. Which also has code hohw to consume this services.



1. As soon as you make any changes locally. Push those change in dev branch in azure and merge in Main branch. Once it is merged, it will trigger CI pipeline and then CD as artifacts will be changed. That’s how CI and CD will work.
2. Now create env using install reuirement.sh file from to run test funciotns using bash cmd
3. Run staging test using aml: pytest staging\_test.py --doctest-modules --junitxml=junit/test-results.xml --cov-report=xml --cov-report=html --scoreurl $(az ml service show -g $(azureml.resourceGroup) -w $(azureml.workspaceName) -n $(service.name.staging) --query scoringUri -o tsv)
4. Publish these staging test results where file can be mentioned as ‘\*\*/test-\*.xml’. change control option as run this even if previous steps failed to debug
5. Rerun these release pipelines. You can also manually test by copying the code of how to consume api endpoints in notebook in ml workspace. Pass the data and run it.

Now you can check in azure ml workspace, container instance is created.