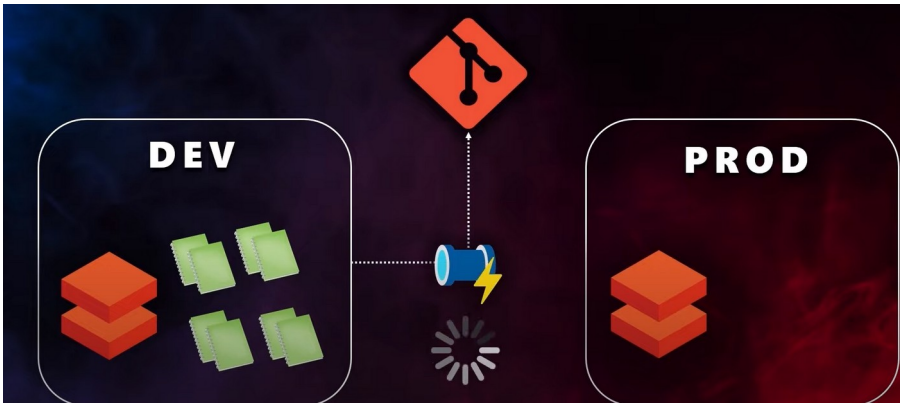
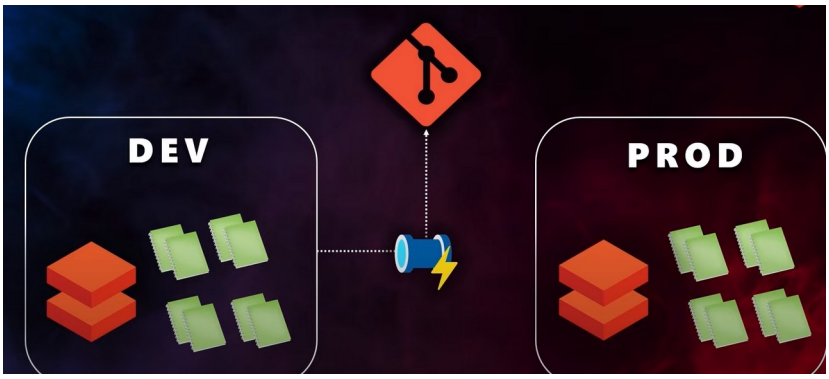
Consider we have a dev resource group , in that resource group consider we have a databricks workspace created. In that workspace, consider we have created different notebooks for doing some data engineering tasks. We have all these in DEV environment which also means that it is in dev resource group. **So main idea of CI/CD is to deploy code from one environment to another environment(Lets say PROD).** Different companies use different numbers and types of environment. For example, a company can have 3 environments like DEV, QA and PROD or DEV, UAT and PROD. Other company might have only 2 environments DEV and PROD which is the standard thing followed by most of the companies. Similar to DEV environment, PROD will have all the resources created. The only difference here is we will not be using prod resource for doing any development work what does this means is we will be creating notebooks and write any code only in Dev environment and we need to deploy all the code changes from DEV to PROD via CI/CD pipeline. The most important thing in CI/CD process is github repository. There are different kinds of repository available such as Github , Azure Devops etc.

In this tutorial , we are using Azure Devops repository for implementing CI/CD pipeline. Repository is mainly used to save all our code . we will be integrating this repository to only 1 environment that is DEV environment i.e; databricks workspace which is in DEV resource group will be integrated with that repository. There will be no connection between PROD databricks workspace and the repository.

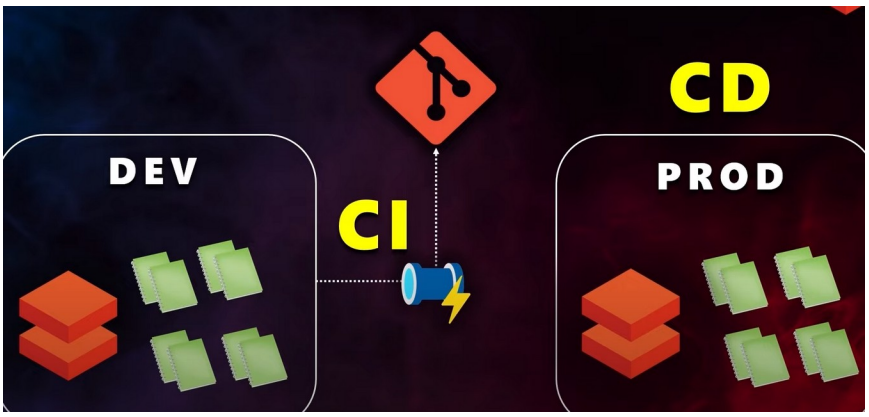
We will be doing all our code changes only in the dev databricks workspace and once we have completed all our work, we can commit all our changes to git repository which means that we are saving all our code to the repository. As soon as we commit the changes to git repository, what will happen is CI/CD pipeline will get triggered as shown in below screenshot



and pipeline will get latest changes from dev environment and deploy all the code to prod environment as shown below.



This is the complete CI/CD process. So here the part **where we integrate dev changes to the repository** is called as continuous integration. The part **where we are deploying all the latest code from Dev to prod** is called as continuous deployment.



MERGING/BRANCHING TECHNIQUES USED IN CI/CD PROCESS

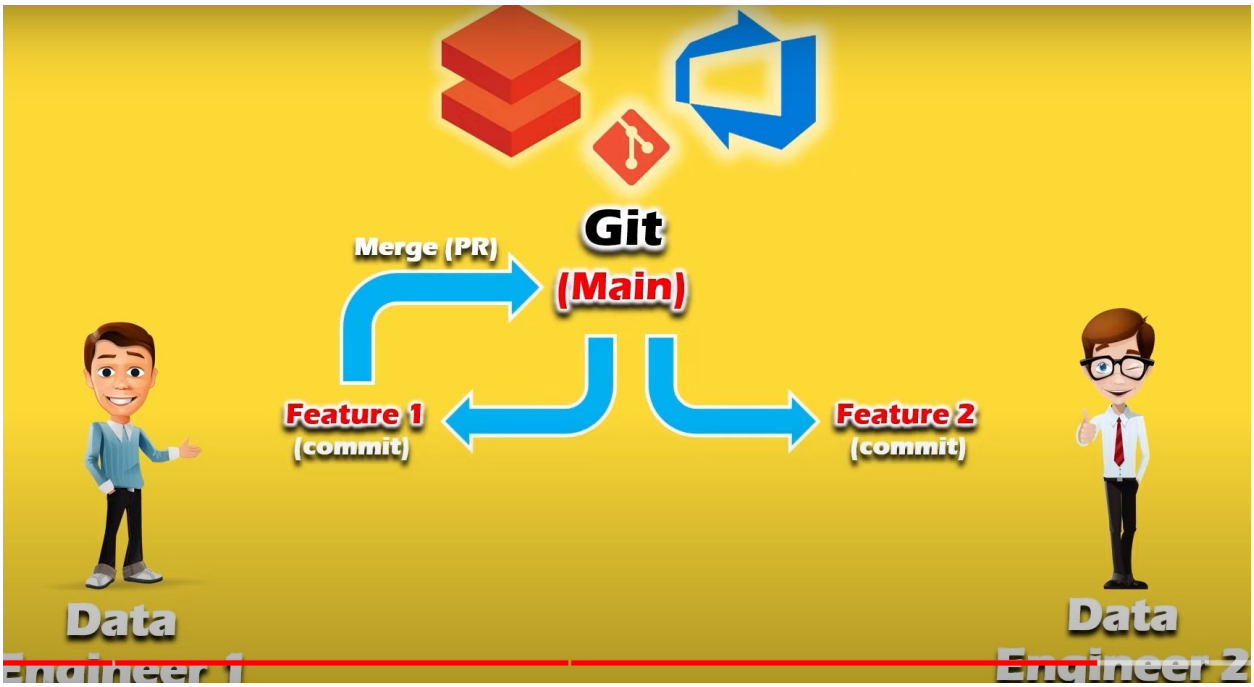
There are different ways of merging techniques followed by different companies . Let's see this with 1 example. For example, let's consider there are 2 data engineers. These 2 data engineers are collaboratively working with 1 databricks workspace. This databricks workspace is in DEV environment. All development work will be done in DEV environment. For CI/CD process, the first and foremost important thing is integrating databricks workspace with git repository, so we are going to use Azure devops repository for this integration. So, let's consider we have integrated azure devops repository to the databricks workspace.So while doing this integration we need to create main branch in this repository. This main branch is very important because CI/CD pipeline that we are going to create will

trigger only when there is a change happening in the main branch which means for example, if data engineer makes some changes to the main branch , then this will act as a triggering point for our CI CD pipeline and pipeline will get latest changes from DEV and will deploy to the PROD . so that's the reason main branch is important.

In terms of branching technique, we are going to protect this main branch so what is mean by protection is no data engineers can directly make a change to this main branch. They cannot update any code directly in the main branch say for example **if data engineer 1 needs to make a change then he must create a new branch based on the main branch. This branch can be a feature branch which has exact copy of the main branch so he needs to make all the changes only on this feature branch that he has created and cannot make any changes in the main branch.**Similarly if data engineer 2 needs to make any changes, then he has to create his own feature branch and use that branch to do any work. So, in case if some one tries to make changes directly to the main branch, then they should get an error. So, this is called branch protection and we are protecting main branch based on the merging techniques that we follow in the CI/CD process.

If data engineer 1 has finished his changes, then what he can do is he can commit all his changes to his feature branch.Similarly if data engineer 2 has completed all his changes and have committed all the changes to his feature branch. Both of the data engineers have completed their respective work . Whose changes should be merged to main branch first? Depends on priority between these 2 changes. What we mean by priority is whose changes should be first deployed to PROD should be merged to the main branch first. So for this data engineers and other operation team should discuss and talk about priorities .

The reason for discussing the priority is as soon as we merge changes to the main branch , our ci/cd pipeline will trigger and deploy the changes to the PROD environment so that's the reason, discussing about priorities is very important in the ci/cd process. So consider they have decided for the data engineer 1 to merge changes to the main branch first**. So now what data engineer 1 should do is he has to create pull request to merge his changes from feature1 branch to main branch.** So once this PR gets approved, all the changes done by data engineer1 will be moved to main branch.

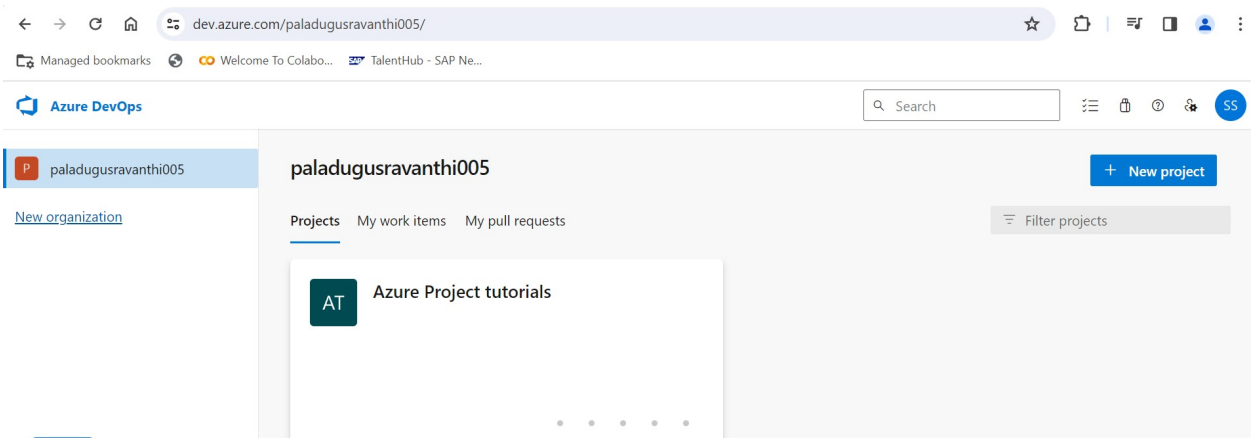


Now main branch will get updated with the latest changes . so now what happen is CI/CD pipeline will gets triggered and it will get all the latest changes from DEV environment and will deploy all the code to the PROD environment. This is the complete process of CI/CD. So once pipeline has finished deploying all the changes , data engineering 2 can follow the same step by creating another pull request to merge his changes from feature branch to main branch. So, after merging the changes CI/CD pipeline will again gets triggered and it will deploy latest changes to the PROD environment. So, this is the merging technique we are going to use in CI/CD process.

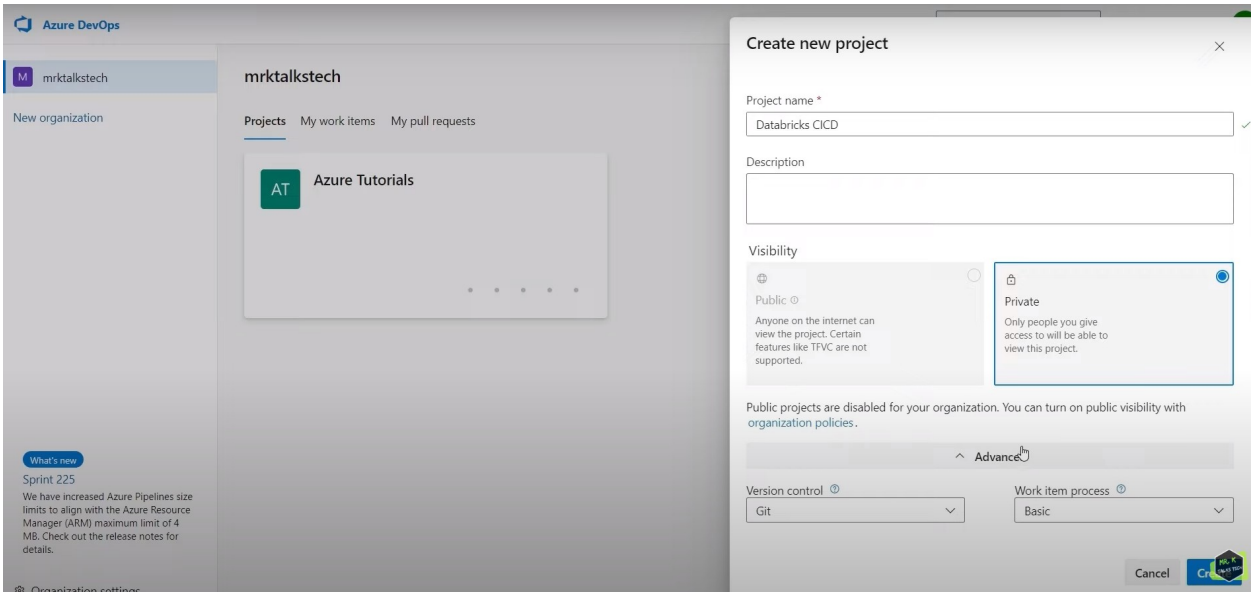
Environment Setup - Integrating Azure DevOps GIT Repository to Azure Databricks for CI/CD

We will integrate repository only to DEV environment . we will not integrate any prod resource to their repository. So now we are going to integrate Azure devops repository with this Dev databricks workspace.

Now we go to dev.azure.com (Azure devops portal) sign up for free and will enter project name and organization name will be gmail name taken by default and we will region and continue , below screen will be populated.



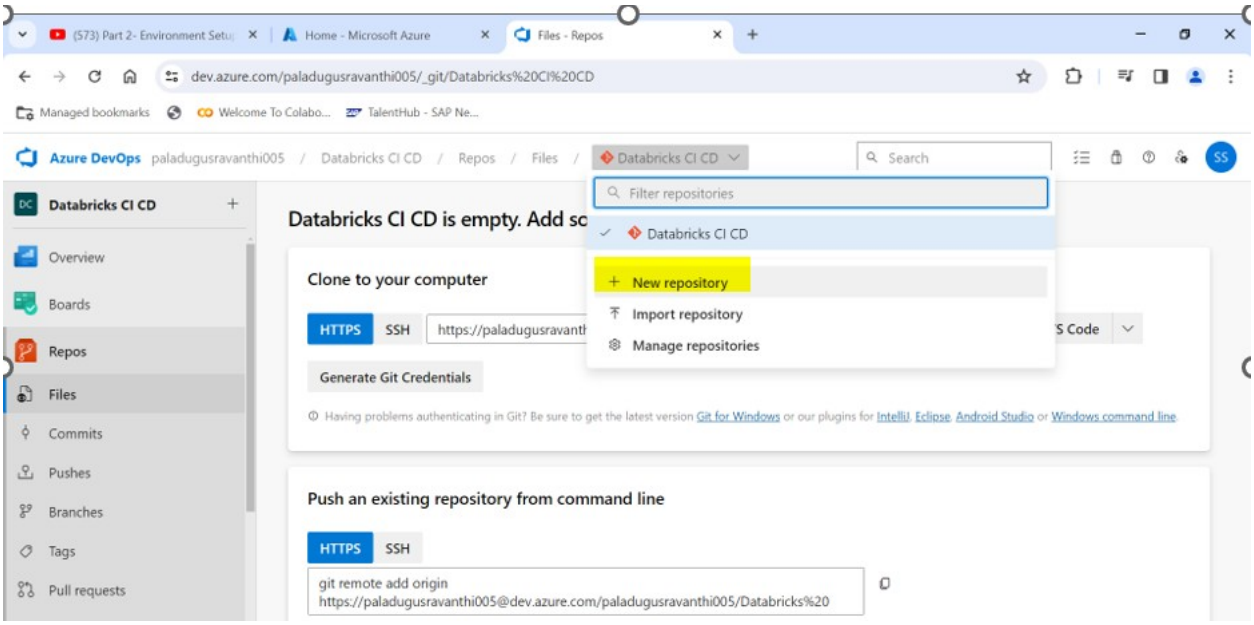
Here paladugusravanthi005 is organization name for our azure devops . Project name is Azure Project tutorials.

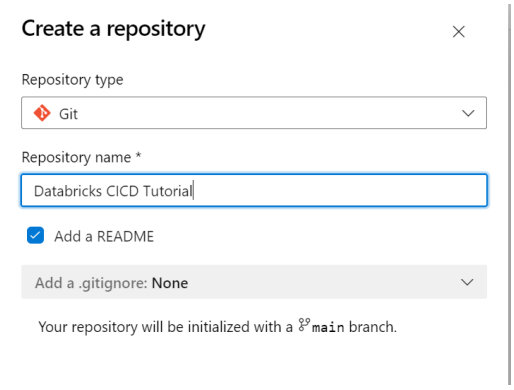


We are creating new project named Databricks CI CD .

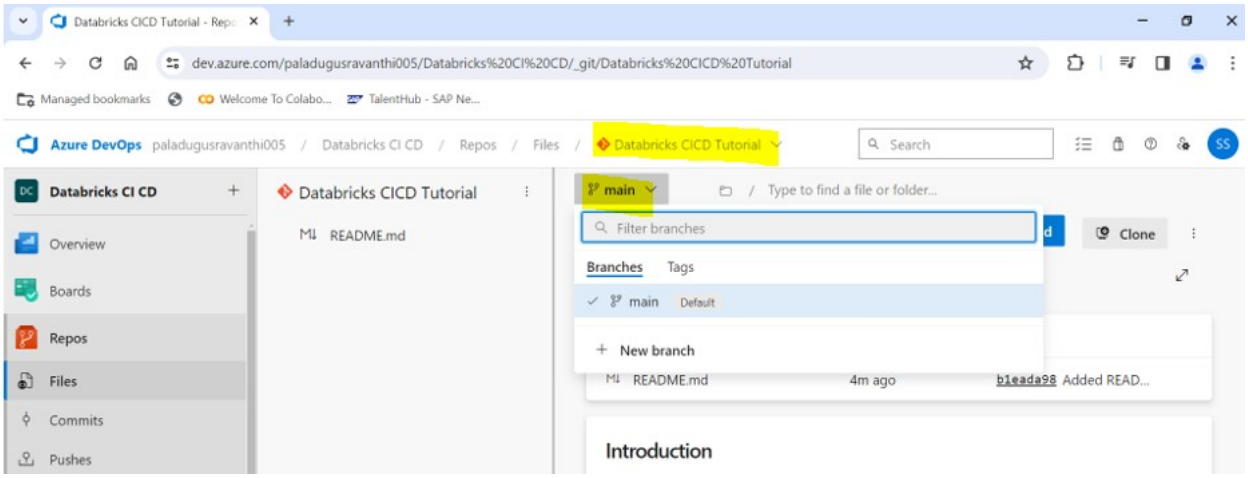
Now we created project, Inside project we now needed to create new repository inside this project. So, for that in LHS go to repos . By default, repository name is created as same name as project name. we can use this Databricks CI CD repository for our integration with databricks workspace or we can create new repository for integration.

Will create new repository as shown below.





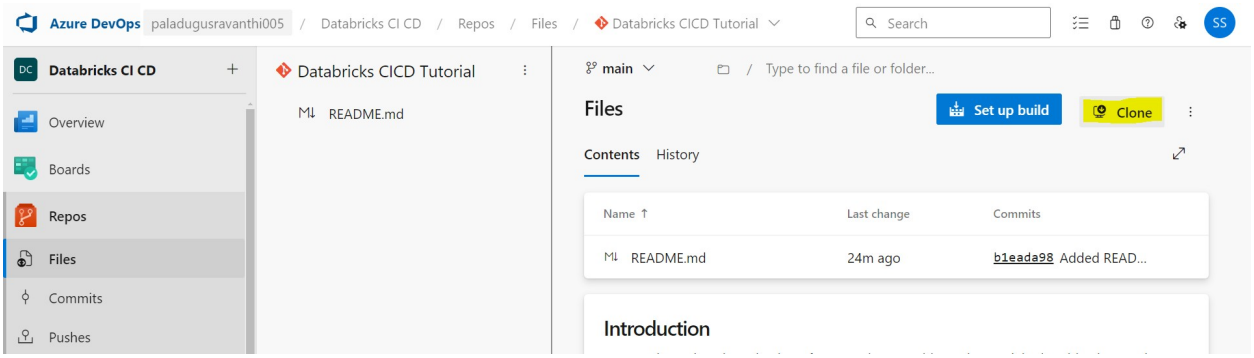
Here we got repository will be initialized with main branch which means when creating this repository (Databricks CICD Tutorial), a new branch with the name main will be created in this repository.



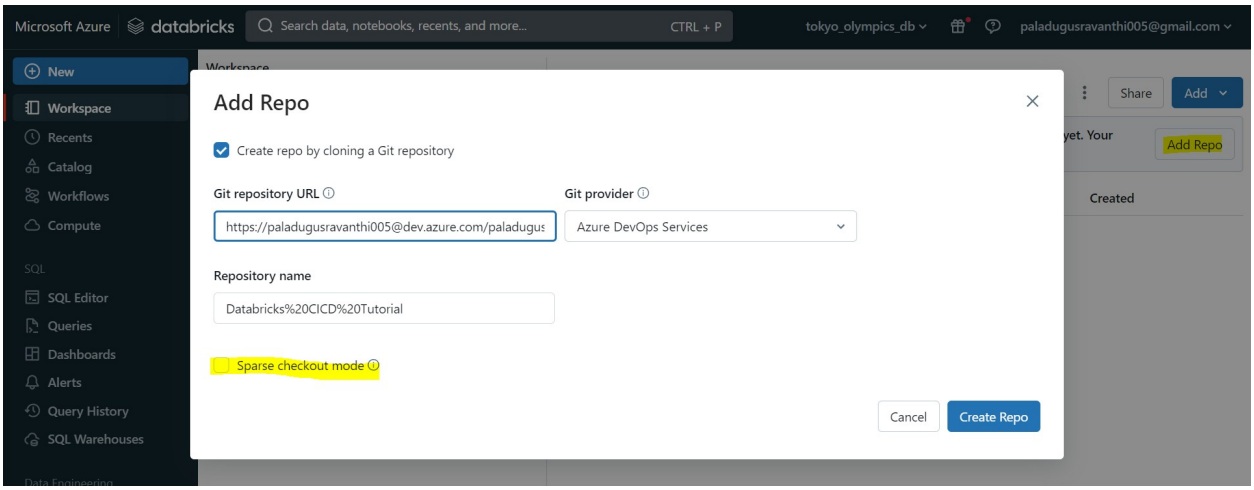
Main branch is created as shown above in Databricks CICD Tutorial repository .

Now go back to ADB workspace cluster , go to repos option → we need to integrate this with azure devops repository. So go to repos → Add Repos → enter git repository URL (we will get this from devops repository)

Click on clone , we will get http URL ,



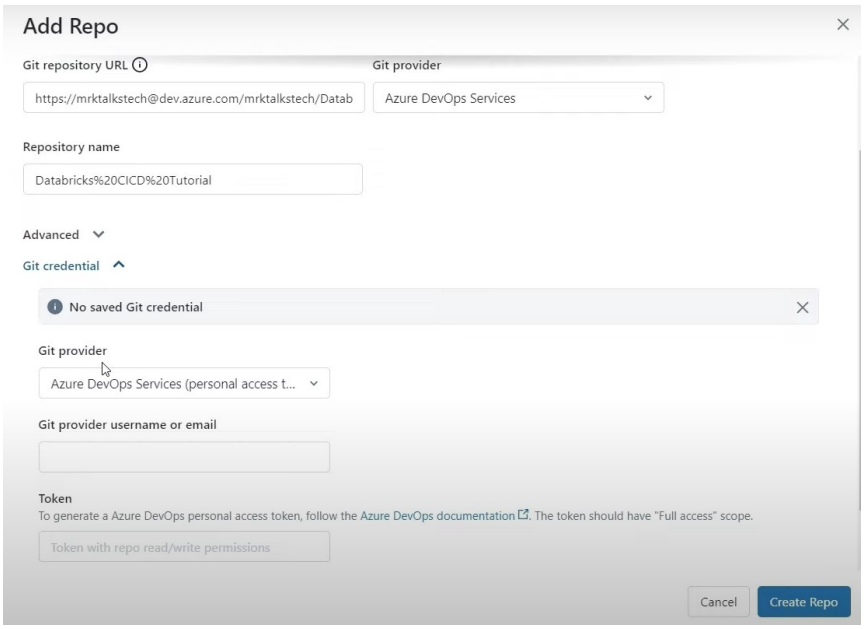
copy this URL and paste in data bricks repo→ Git repository URL as shown below.

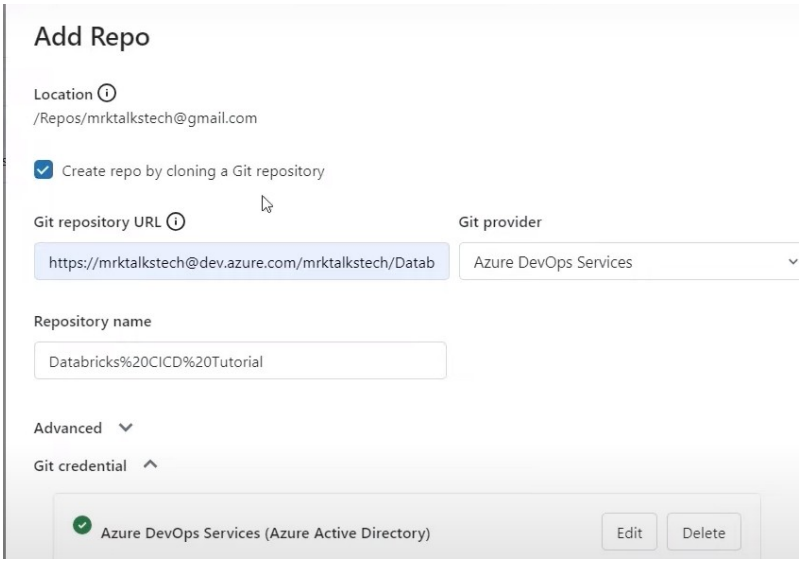


Sparse checkout mode can be used if we want to clone only a certain folder from the repository. In that case, we can enable this option and can specify folder path for cloning it. Default this option will be unchecked for cloning the whole repository instead of a particular folder

And if we expand git credential , this git credential is used to specify how we are going to authenticate to azure devops from this databricks workspace.

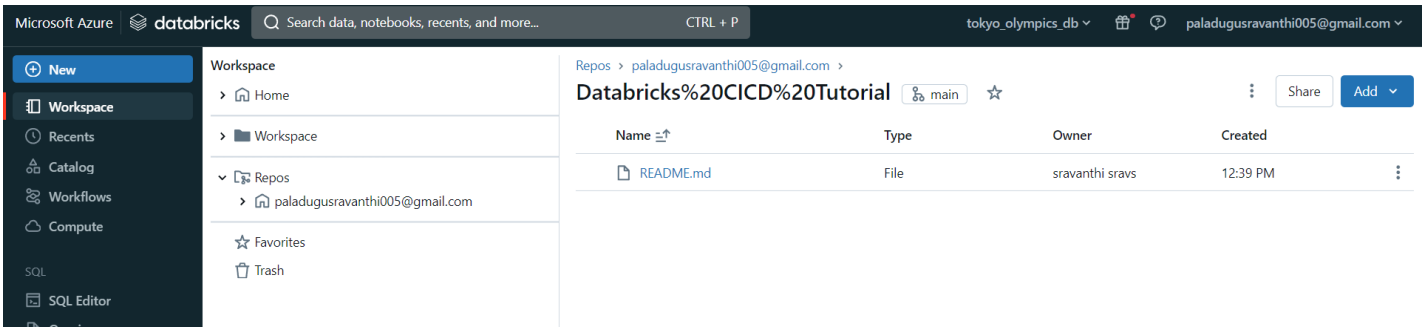
Git Provider : By default Azure Devops Services (Personal Access Token) will be selected. If we use this option to authenticate, then we need to create a token using azure devops and then we use that token inside dbws for authentication. Token based authentication is not so good since we have few complexities like managing the tokens such as updating the token frequently or creating new token if the token got expired and other overheads. Because of these reasons and complexities , instead of using token based authentication we can use Azure Devops Services Azure Active Directory which is a managed identity based authentication which means same user(same mail id) has access to both dbws and azure devops repository. We can use that account credentials for the authentication. So lets choose Azure devops services (AAD) option





We have an option create repo by cloning Git repository check box, By default this option will be enabled. what does this option mean is when this option is enabled by integrating azure devops with this dbws , we will be cloning a complete copy of the files that are in the repository to the dbws which means that if there is any code in repository that code will be copied over to the dbws as well. Since we have just created repository in azure devops and we have only 1 single file read me file . so, after the integration, only copy of read me file will be there inside this databricks workspace.

Now we have successfully integrated Azure Devops repository to the dev databricks workspace as shown below.



If we see repo name %20 there and we can rename this by right click on repository name and rename this repository name. renaming of this repository name is done only inside dbws and it will not update the actual repository name in the azure devops.

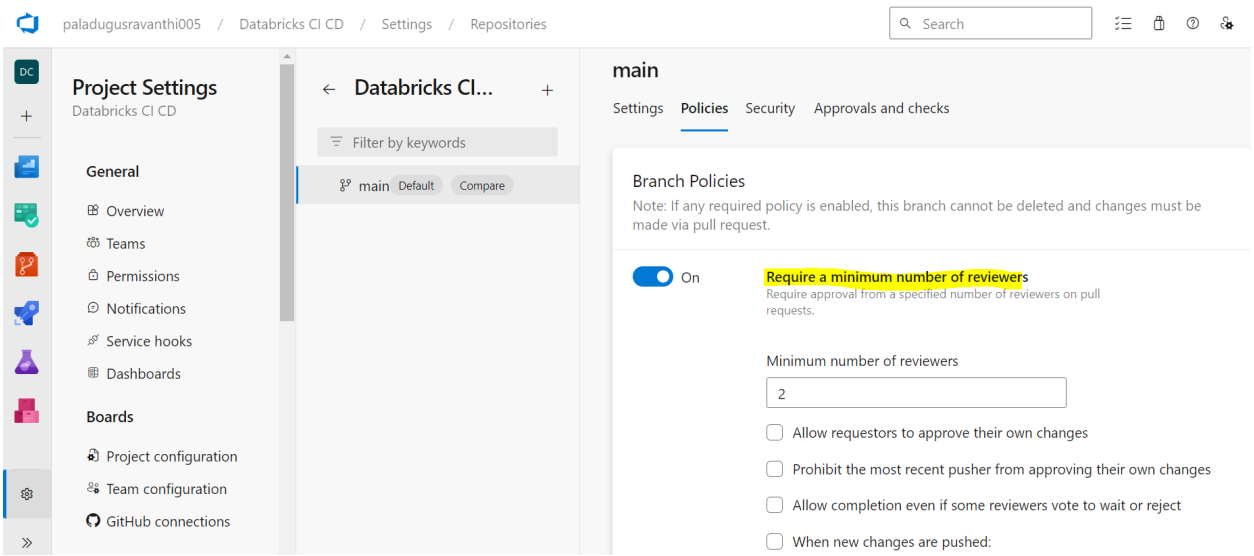
Now we can add our notebook changes file(from workspace file) in repo here directly in main branch file . if we click on main we can commit to azure devops. Now if we go to azure devops changes will be visible.

We haven't protected main branch , so file was committed in azure devops repository.

To protect main branch → in azure devops → go to branches → in main branch click on 3 dots → select Branch policies .

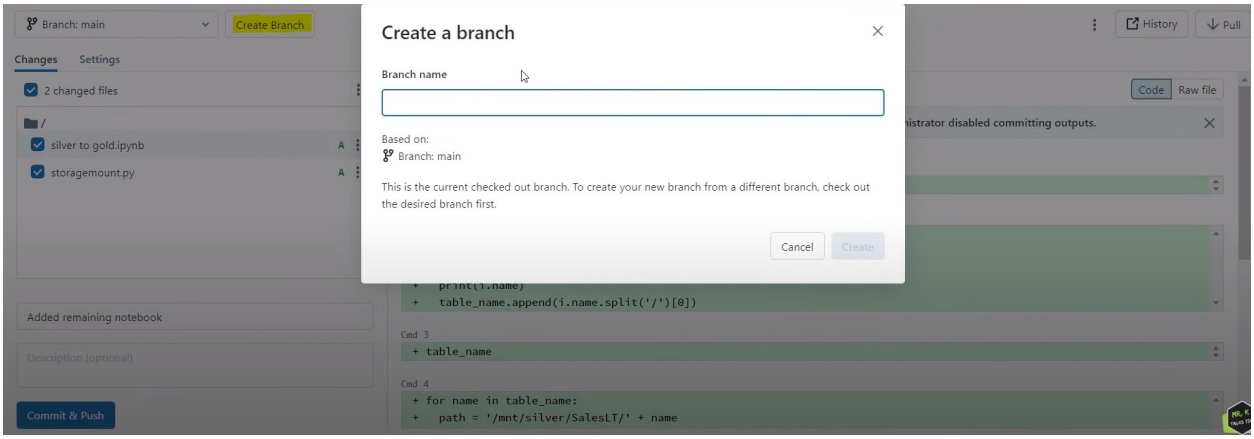
Now if we enable any of 4 branch policies, this main branch cant be deleted and also any changes to the main branch should be done via pull request. We cannot directly commit any changes to the main branch.

If we enable first branch policy , require minimum no of reviewers,



If we enable this option before making any changes to the main branch, a reviewer should approve the change that we have made. Say for example, a data engineer creates a pull request to merge the changes from his feature branch to main branch , then that pull request should be approved by a reviewer before merging those changes to the main branch.

Now we need to create feature branch, so for that click on create branch as shown below in db repo



Once feature branch is created and kept notebook files in feature branch. Now come to azure devops repository → in pull request → create pull request to merge changes to main branch and feature branch will be deleted if we kept that settings while merging data to main branch from feature branch.

Creating the Congtinuous Integration Pipeline

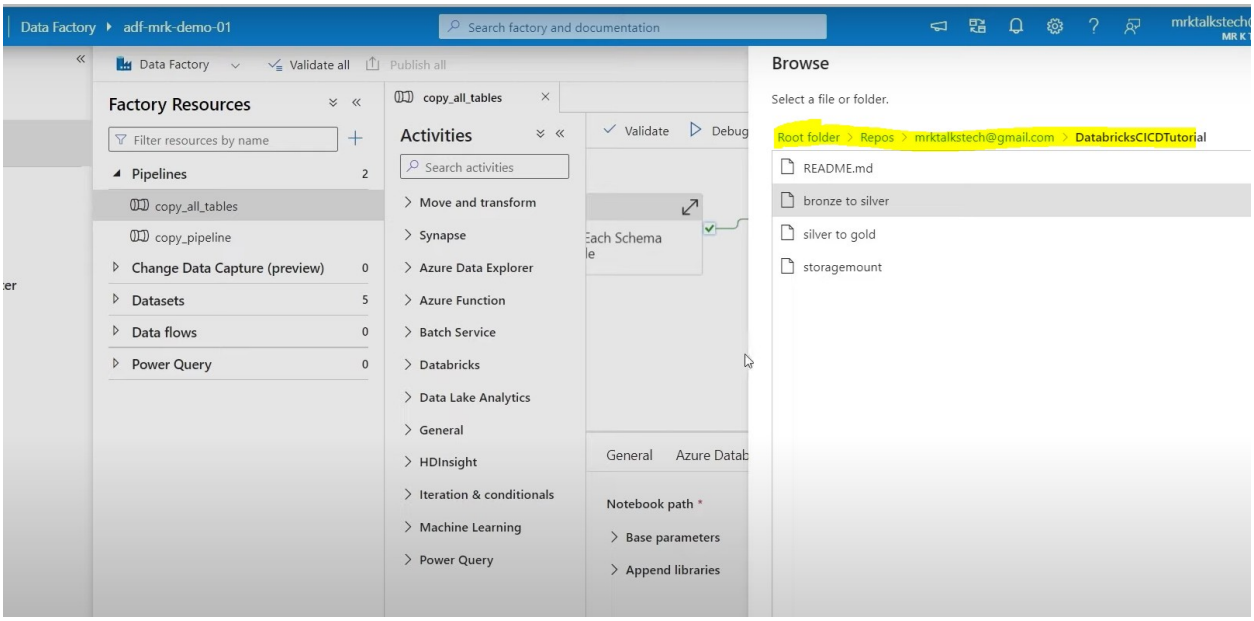
Let's say we have 2 environments, which are DEV and PROD and in the dev environment we will be doing all our development work and once we have done with our Dev work and done merging our changes to the main branch. then as soon as changes are merged to main branch , our ci-cd pipeline will gets triggered and it will copy latest changes from main branch and deploys it to the production environment. This is the process of ci-cd process.

As part of continuous integration, we integrated dev databricks workspace with azure devops repository. We have a repo folder inside data bricks workspace and all dev work will be done in repo folder . For example, we created feature branch inside dev dbws and in that feature branch we have made few changes such as adding new databricks notebook or any other stuff . Once we finish the changes and create a pull request and merge the changes to main branch. Then as soon as we merge changes to main branch, CI-CD pipeline will trigger as usual . Continuous integration will do one extra step before deploying the changes to PROD environment , it will first copy all the latest changes from main branch to dev dbws itself . so, this is the first step of continuous integration pipeline. Pipeline will create new folder called live inside dev dbws and it will copy all the latest notebook from main branch to this live folder.

REASON FOR COPYING LATEST NOTEBOOKS FROM MAIN BRANCH TO LIVE FOLDER:

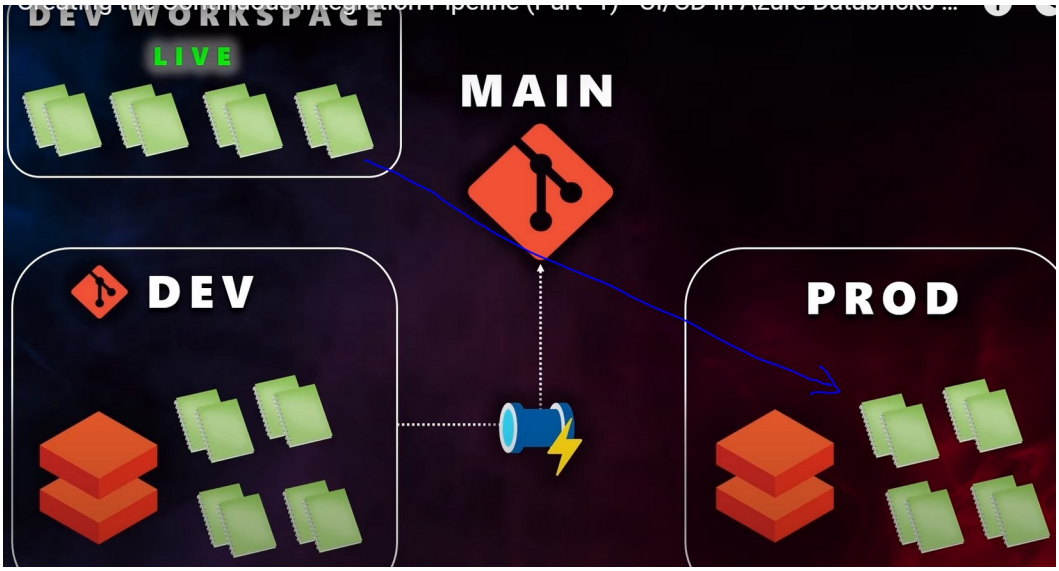
In ADF pipelines, lets assume we specified notebook activity with path mentioned as shared /users workspace location. Now we **moved** all notebooks from shared/user workspace location to repository location. Now there is no notebooks in shared/user workspace folder. Notebooks are in repository location. Now we run ADF pipeline again, we will get error actually since notebook is actually missing in shared /users workspace location. We know that main branch of the repository will have all the latest code but problem is referring actual main branch of devops repository cannot be done inside the adf pipelines.

Now in ADF pipeline, if we browse notebook path from repo location as shown below, there is no option to select actual main branch of azure devops repository which will have all latest code. Say for example, if data engineer makes some changes to main branch, then that changes will not be reflected into our account repo location until we pull up latest changes from main branch. In that case, we will be missing latest code so since we cannot refer main branch of devops repository inside ADF pipelines.

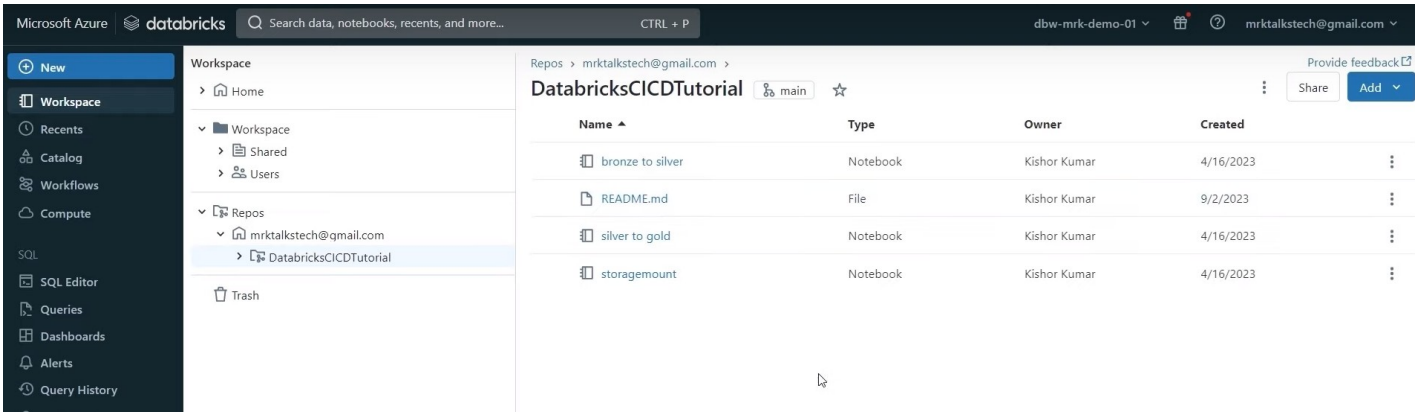


CI pipeline will copy all latest notebooks from main branch to live folder inside workspace. This ADF pipeline will always refer to the notebooks that are inside the live folder which will be the exact copy of the main branch. This live pipeline will be created by the CI pipeline itself . In some of the real time projects, we will be configuring some unit test functionality along with the CI pipeline so that unit test will have some test cases to test the code before actually merging the changes to the main branch.

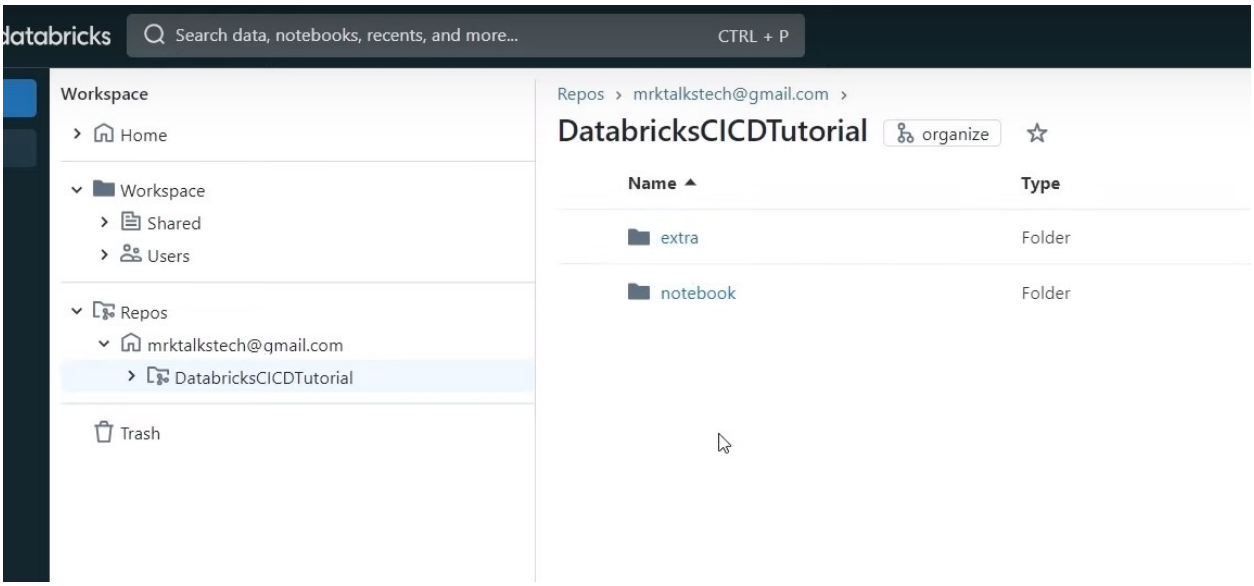
Once CI pipeline have copied all the latest code to this live folder, Continuous deployment pipeline will gets triggered and it will copy all the latest code from DEV live folder and deploy that it into PROD environment workspace with the same live folder so basically exact live folder will be deployed to production environment as well.



In order to make our CI CD pipeline work effectively in deploying changes from one environment to other environment, its always a good practice to organize available files in repository which means we have mix of notebooks and read me file in repository as shown below,



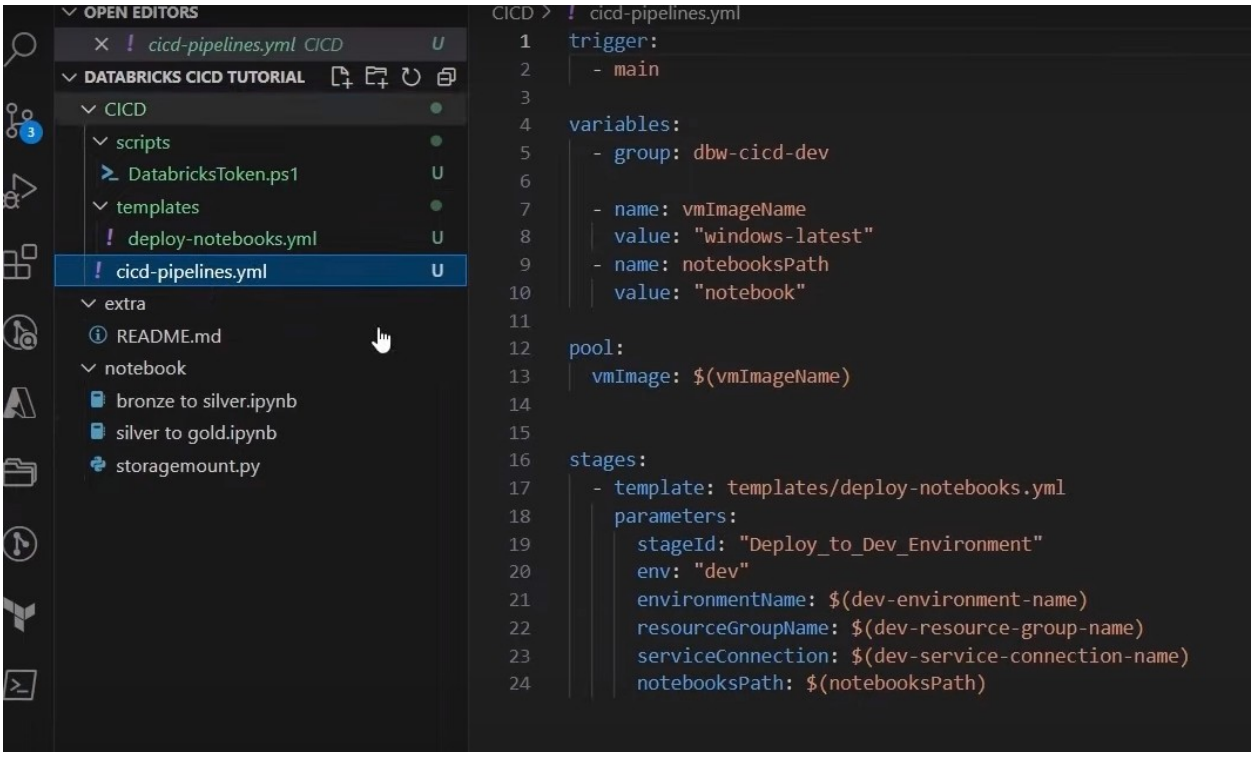
Here readme.md file is not the actual code and it is not necessarily important to deploy this file to the production environment so we need to organize the available files in a folder structure for easy development as shown below,



All 3 notebooks moved to notebook folder and readme.md file moved to extra folder. We will configure CI pipeline in the way that only files which are inside this notebook folder will be deployed to production environment. All Data Engineers will make use of this notebook folder for doing any changes like creating new notebook or making changes to existing notebooks which are already inside this notebook folder.

For creating CI pipeline, we need to write and merge that into main branch of the web repository. Code we are going to use for building CI CD pipeline is called YAML code. YAML code mainly used for building CI CD pipeline

In order to get our entire repo to visual studio, go to azure devops repo → clone to vs

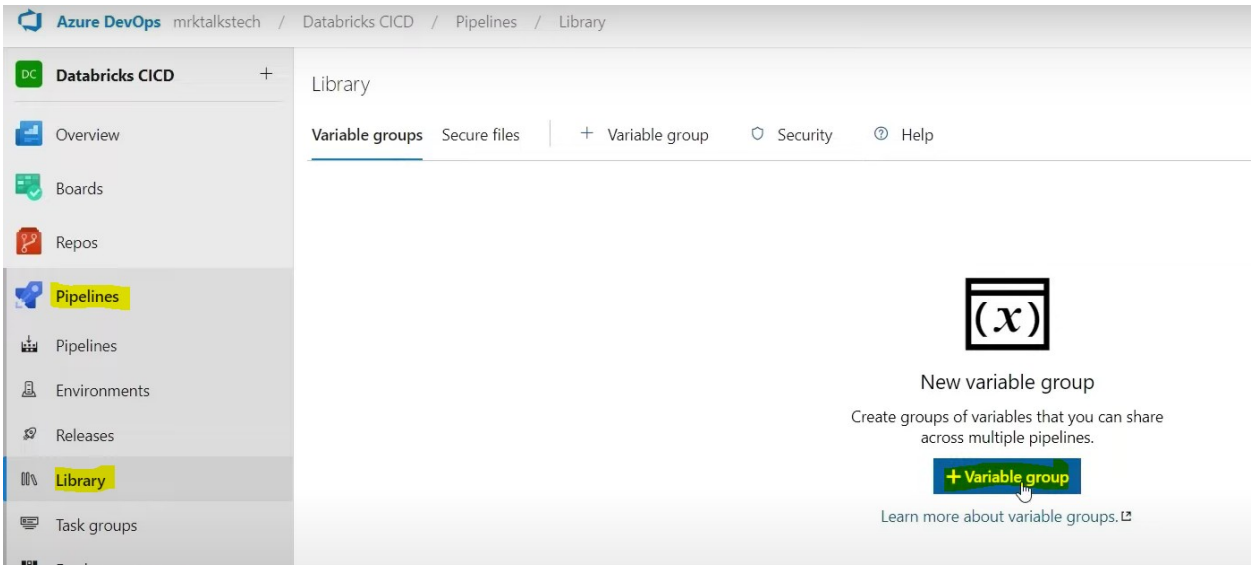


This file is the master file for CI-CD pipeline. Master file means when we actually create CI-CD pipeline in azure devops we will use this master file to create it. This master file will call template file which is **deploy-notebooks.yml**. This **deploy-notebooks.yml** will call the script file **DatabricksToken.ps1** which is a partial script file. totally we have 3 files master file, template file and script file.

The above master file looks like mostly in the form of JSON structure . most of the companies using YAML file to create CI-CD pipelines since this can be easily reused in different repositories such as azure devops or github.

In master YAML file as shown above , most imp thing is first 2 lines . it was given property as **trigger** and value given as main. What does this mean is YAML file will be triggered only when there is a change happens in the main branch. Master file used to create CI CD pipeline in azure devops. CI-CD pipeline will be triggered as soon as we update a change in the main branch. That's the main idea of trigger function.

Next thing we have is **variable groups**: In Azure Devops , we will be creating variable groups. Main use of variable groups is say for example, in dev environment, data bricks name would be a different, resource group name will be different when compared to data bricks name and resource group name in PROD. So for this reason we need to create a variable group specific to each environment and variables related to that environment should be saved in that particular group. Now let's create a variable group for the dev environment with the name dbw-cicd-dev. Go to pipeline → Library → click on Variable group to create new one for dev.



Now in master file, line 7-10 . This is just like setting parameters in the code. We have a parameter called vmImageName and value for this windows-latest. We have a parameter called notebooksPath and its value is assigned as notebook. This parameter can be used anywhere in this file. For example vmImageName is used under pool section, the value is passed as a parameter to the VM image property and the value windows-latest will be assigned to this VM image property.

```
variables:
  - group: dbw-cicd-dev

  - name: vmImageName
    value: "windows-latest"
  - name: notebooksPath
    value: "notebook"

pool:
  vmImage: $(vmImageName)
```

In simple terms, pool is just a compute power. For example, to execute CI-CD pipeline in azure devops , we need some sort of compute power. That consume power will be consumed from this pool. Microsoft provides different types of pools that can be configured to run CI-CD pipeline in azure devops.

Here vmImageName can be as below as shown in screenshot.

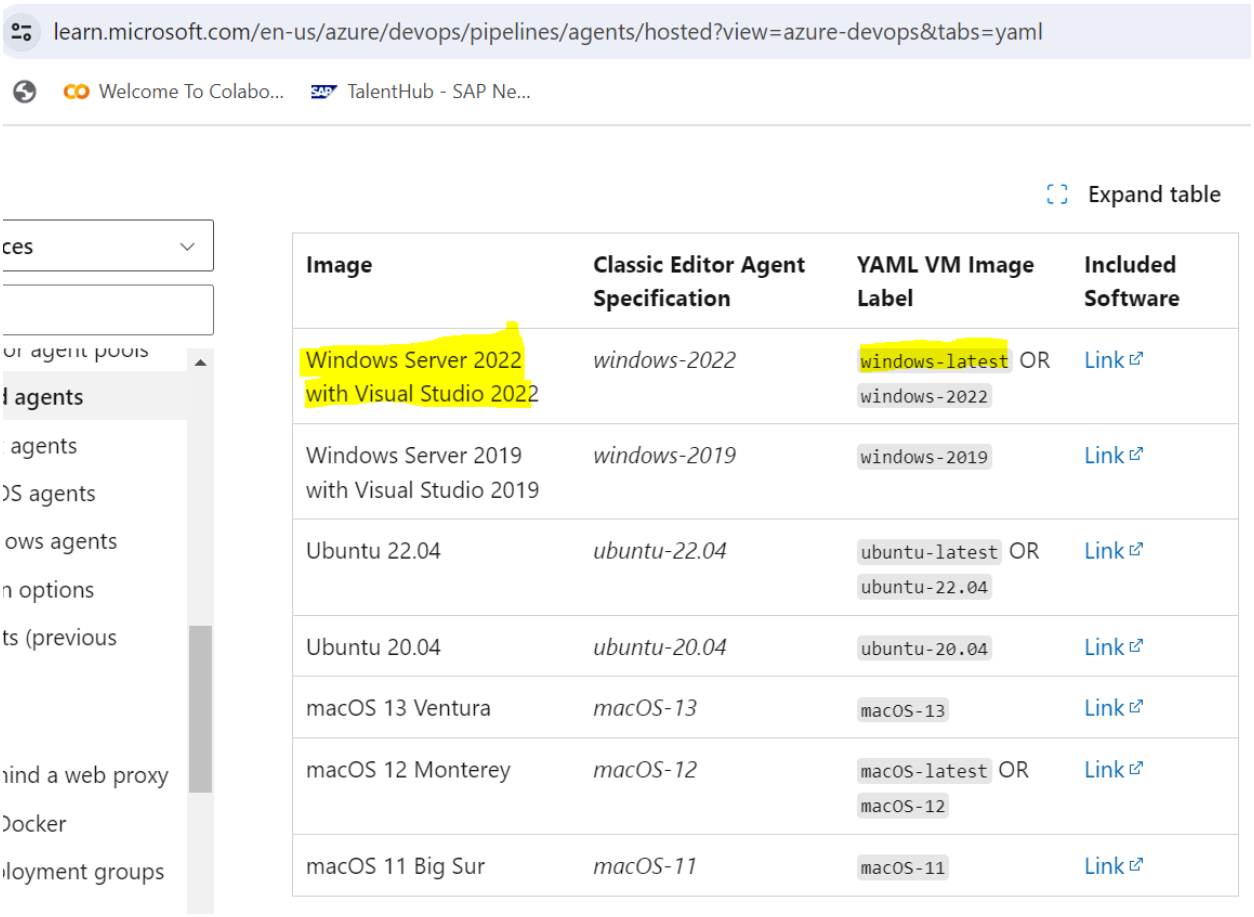


Image	Classic Editor Agent Specification	YAML VM Image Label	Included Software
Windows Server 2022 with Visual Studio 2022	*windows-2022*	`windows-latest` OR `windows-2022`	Link
Windows Server 2019 with Visual Studio 2019	*windows-2019*	`windows-2019`	Link
Ubuntu 22.04	*ubuntu-22.04*	`ubuntu-latest` OR `ubuntu-22.04`	Link
Ubuntu 20.04	*ubuntu-20.04*	`ubuntu-20.04`	Link
macOS 13 Ventura	*macOS-13*	`macOS-13`	Link
macOS 12 Monterey	*macOS-12*	`macOS-latest` OR `macOS-12`	Link
macOS 11 Big Sur	*macOS-11*	`macOS-11`	Link

And notebook path value as notebook means file inside notebook path are pushed to the PROD environment. That's why we specified notebook as a value in this notebook parameter.

Next one is stages.

This is the most important part in the master file. This is the segment that does the actual deployment. Stage should be configured for each environment specifically . In stages , we have a first option called templates. This is where we are actually calling template file which is deploy-notebooks.yml. While calling template file, its going to send all the parameters from 19-24 lines as shown below. Different parameters are:

stageId: This is just to represent the use of the stage. Given value as Deploy_to_Dev_Environment

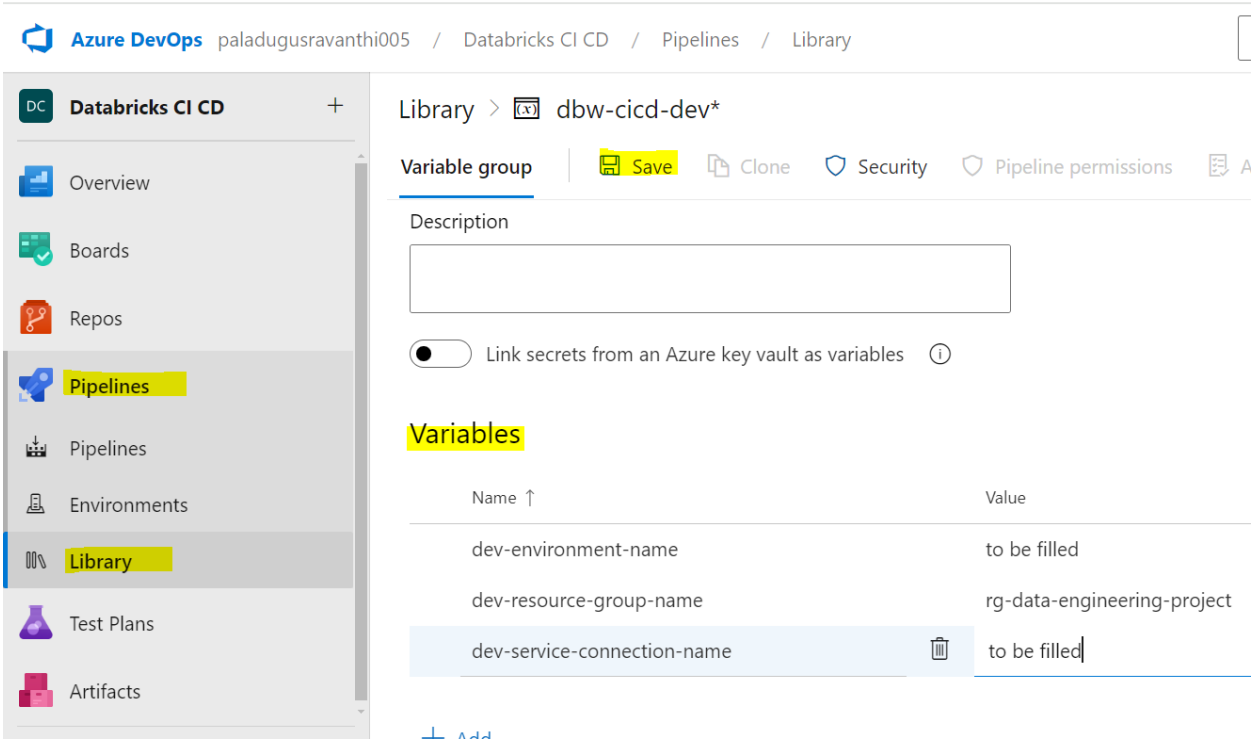
env: 2nd parameter called as env which means for what environment we are creating this stage for.

And next parameter NotebooksPath passed as a value as notebook (line 9 ,10)

And all these parameters will be passed as a parameter to the template file for the deployment logic.

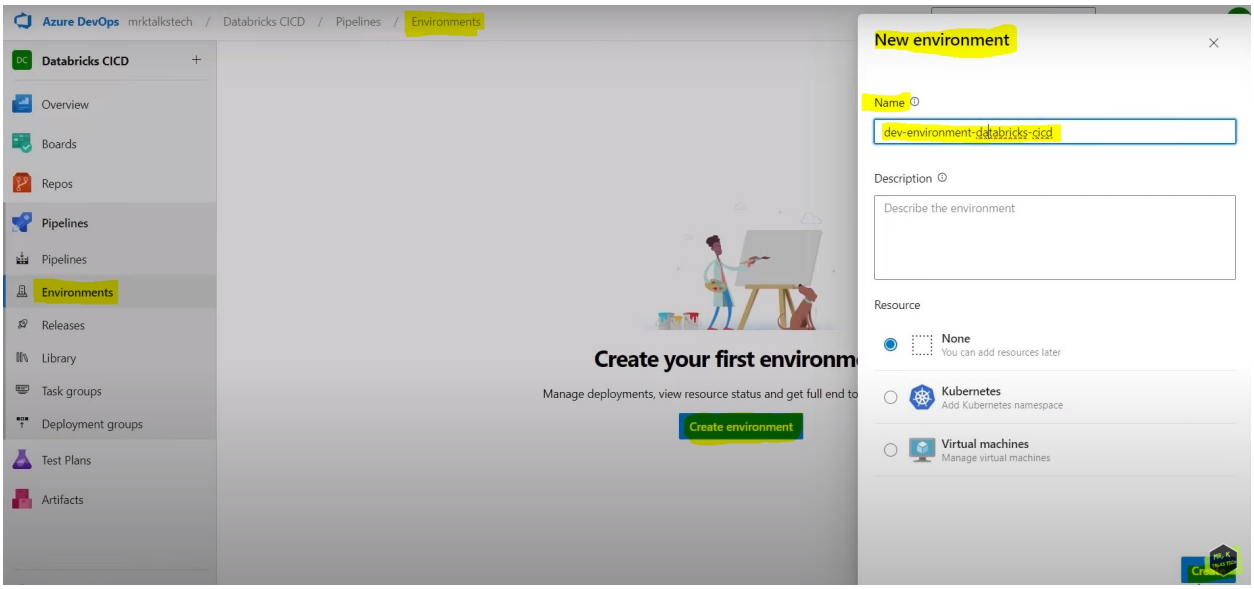
These 3 parameters(21-23) , we are not directly specifying value in the code. Instead, we are going to add these as a variable in the dev variable group that we created in azure devops. We need to create 3 variables , one for environmentName, 2nd for resourceGroupName , 3rd for serviceConnection. So name of the variable should be exactly specified as same as in YAML file code.

```yaml
stages:
  - template: templates/deploy-notebooks.yml
    parameters:
      stageId: "Deploy_to_Dev_Environment"
      env: "dev"
      environmentName: $(dev-environment-name)
      resourceGroupName: $(dev-resource-group-name)
      serviceConnection: $(dev-service-connection-name)
      notebooksPath: $(notebooksPath)
```



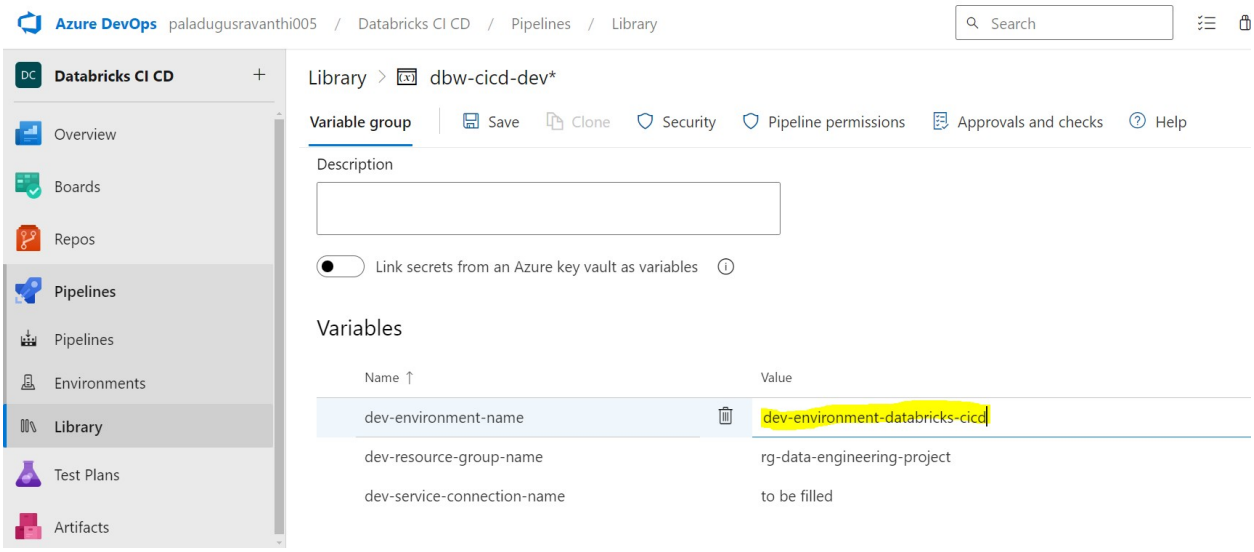
We need to fill dev-environment-name and dev-service-connection-name as shown above. So for that we need to go to Environments in pipelines → create environment → here environment means all the deployment which is performed by CI-CD pipeline can be managed in one place using this environment.

Say for example, if we create an environment for dev , then when ci-cd pipeline runs and deploys changes to dev environment, we can come to environment section and see the status of the deployment such as if the deployment is successful or failure or other information related to the deployment , so that's the main idea of creating environment .



Going forward, we can manage all the dev related deployments from this place as mentioned in above screenshot.

Now go to library section again, go to variable group, in environment text box , we can paste that value as shown below . Now this value dev-environment-databricks-cicd will be passed as a parameter to the code as an environment name and this value will be passed to the template file for further deployment operation.

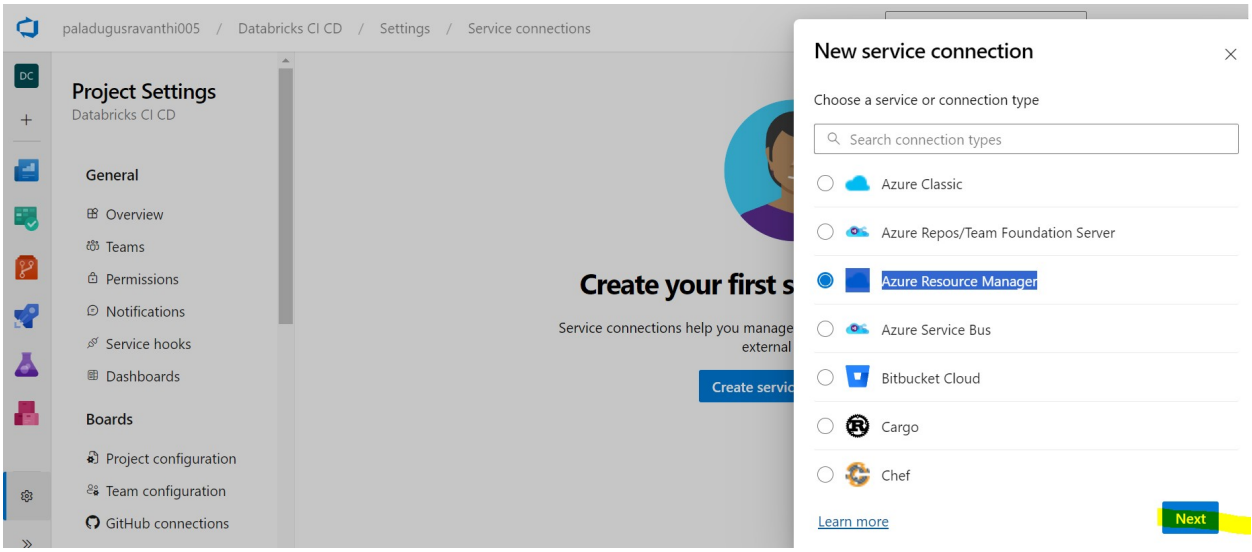


Now service connection, we will create ci-cd pipeline in azure devops, this ci-cd pipeline should have access to the azure resources for making the deployment . In order for ci-cd pipeline to get access to azure resources, we need to establish some sort of connection between azure devops and azure resources. So that can be exactly done using service connection.

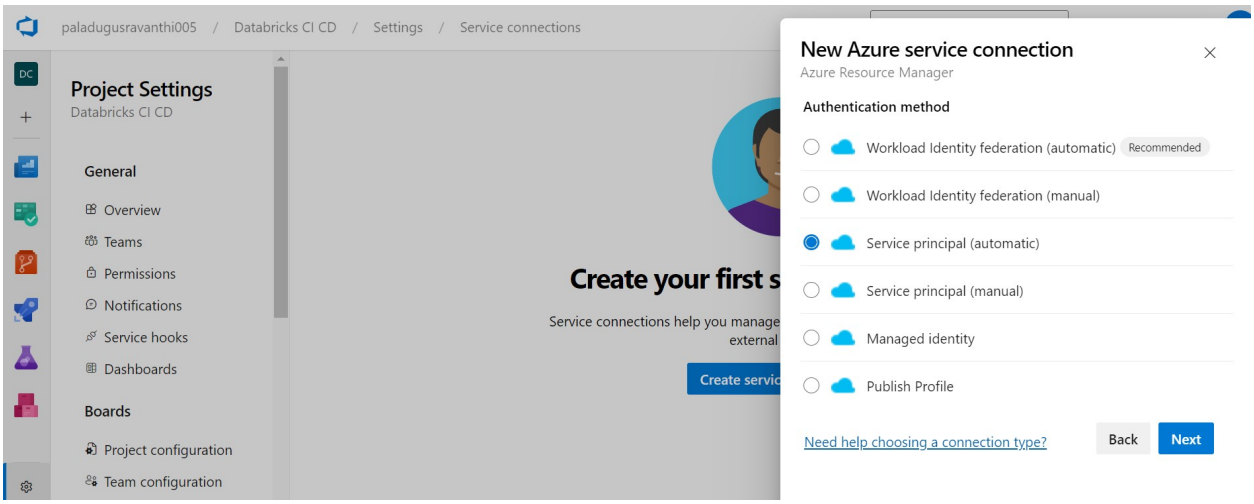
When someone has contributor or owner access, they can access the resources such as databricks workspace which is inside the resource group. So, we can't directly add azure devops to give access to

resource group. So that's the reason we are using service connection which will automatically configure required access to azure devops using service principle .

Bottom left → go to project settings → go to service connection → create service connection → select Azure resource manager(we need to connect to azure resources, so we choose the option azure resource manager). Click on next.

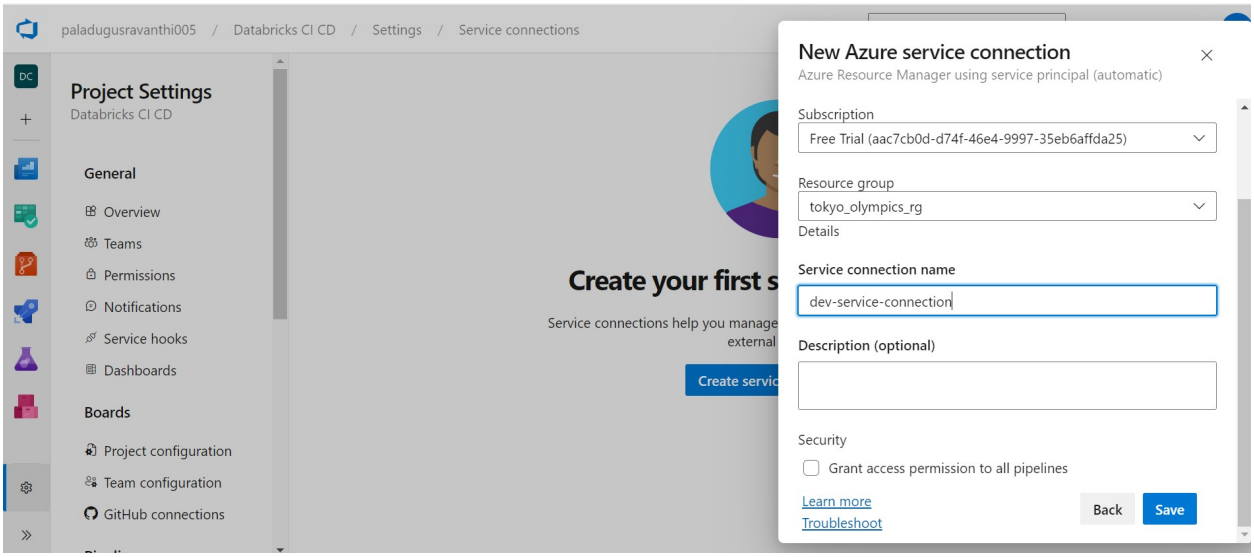


Here select service principle (automatic). Basically, service principle is a kind of third-party service i.e.; we cannot directly add azure devops to give access to this resource group. When we create service connection, it will automatically create a service principle and that service principle will be automatically given contributor access to the resource group. Using authentication of that service principle , azure devops will get access to this resource group and can access any resources inside the resource group for making the deployment. So that's what we are trying to do for service connection.



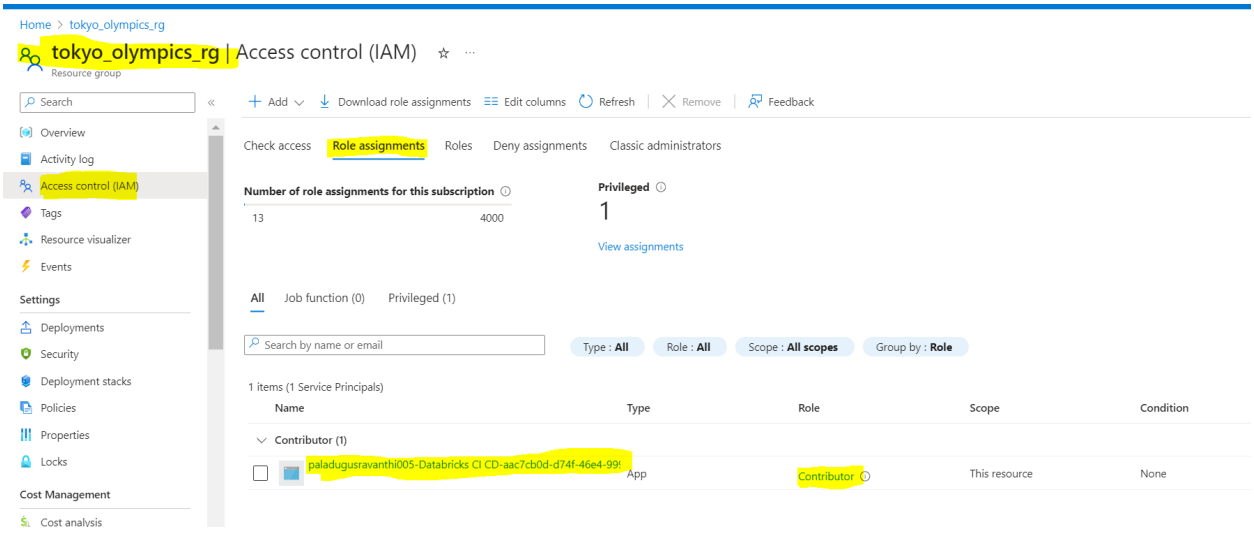
Click on next , here we need to scope service connection to our dev resource group.

Here check box grant access to all pipelines, if this is checked, then all the pipelines that is created in this azure devops account will have access to this service connection which we don't want it. So, we will disable checkbox button as shown below.

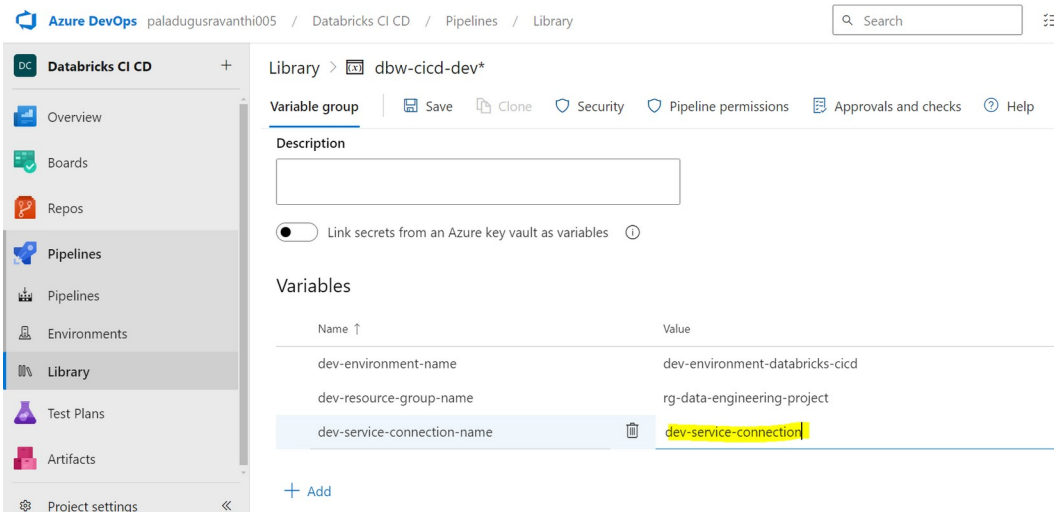


Click on save.

Now it will create new service principle and give the service principle contributor access to the resource group as shown below. We have a new service principle created automatically and given contributor access to this resource group.

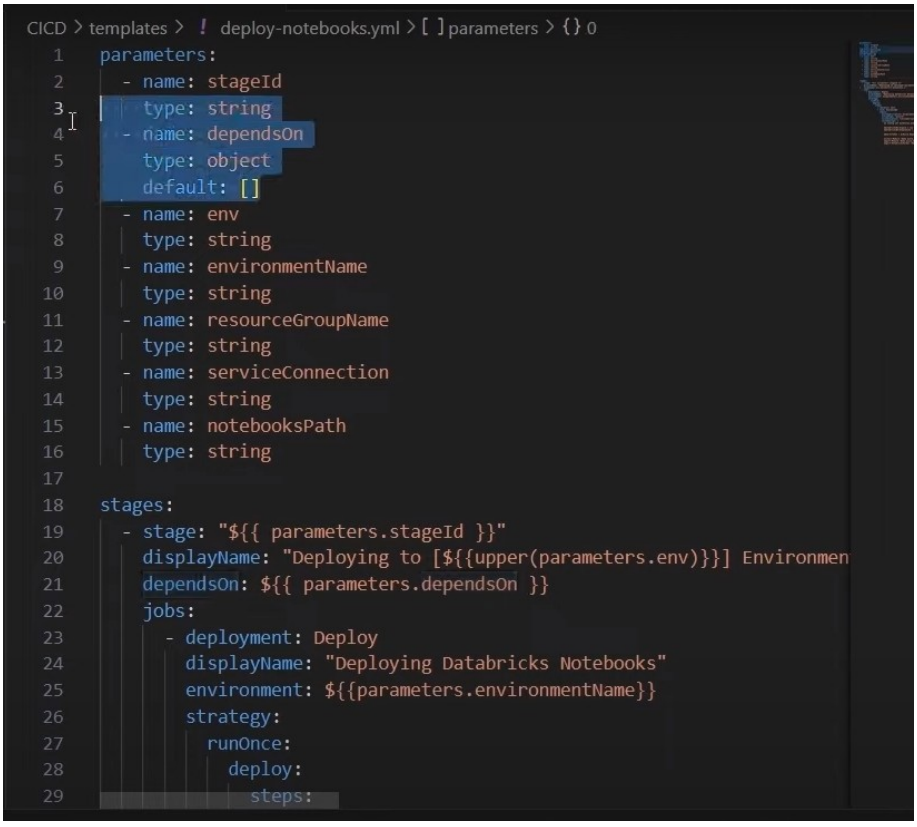


Now azure devops will use this service principle access as an authentication mode to access any resource inside this resource group. Now go to pipeline → library → go to variable group → we will paste the created service connection there as shown below.



Now all these variables are passed to the parameters as code.

Now go to deploy-notebooks.yml file, this is just a template file where same file can be used to deploy notebook changes to multiple environments. Now all the parameters from dev stage(YAML file) will be passed to this template file(deploy-notebooks.yml). dependsOn parameter is not passed from dev stage (YAML file). so dependsOn parameter is just defined in this template file itself with the default value as null. We will be needing this dependsOn parameter only during production deployment. Once we received all the parameters from master file , this template file will then use these parameters and will do further operation for the deployment



Now in template file, we received all the parameter values from the master yaml file. firstly, we have stages in template file as well. Here we are using stageid that is passed from the master file and

we have **display name property** : this is the display name of the pipeline that will be created to deploy to the dev environment. Here env parameter is used here so display name for the dev pipeline will be deploying to dev environment. When the same template file is used again in prod stage, value for the display name will be deploying to prod environment.

dependsOn property: dependsOn property will not be required for continuous integration pipeline.since default value is null, only null value will be passed over here.

Jobs: This will have functionality to deploy the notebooks and display name is Deploying Databricks Notebooks.

Environment: This is the one that we created in azure devops. So the dev environment name has been already added to the variable and it will be assigned to this environment property.

Main function of continuous integration pipeline would be copying files from notebook folder to the live folder in the dev environment workspace , that's going to be done using code section from inline power shell .

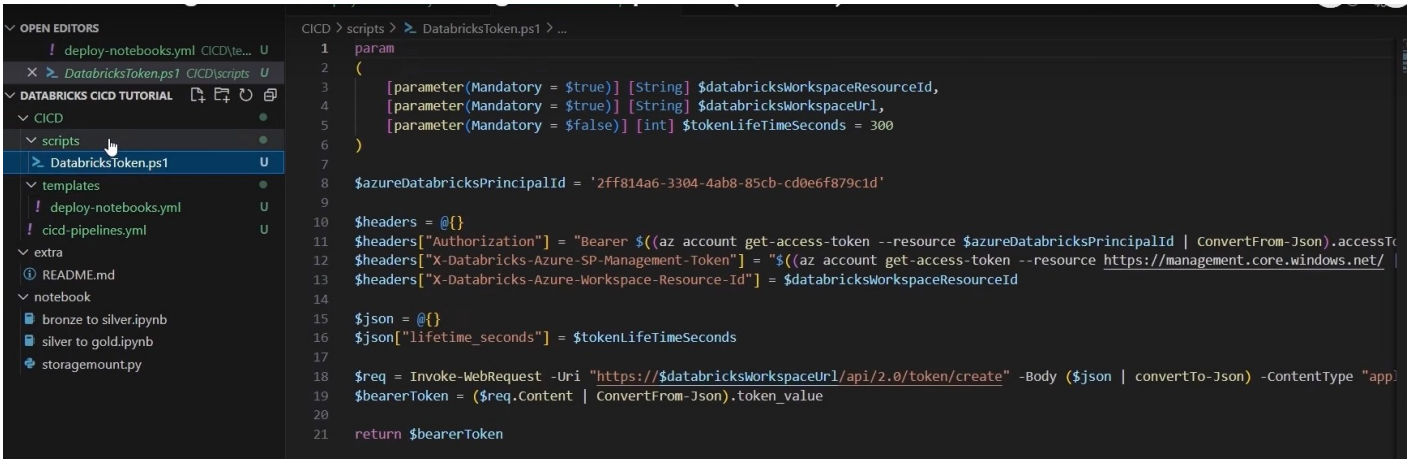
```
27          runOnce:
28            deploy:
29              steps:
30                - checkout: self
31                - task: AzureCLI@2
32                  inputs:
33                    azureSubscription: ${{parameters.serviceConnection}}
34                    scriptType: "pscore"
35                    scriptLocation: "inlineScript"
36                    inlineScript: |
37                      az config set extension.use_dynamic_install=yes_without_prompt
38
39                      $databricksWorkspace = (az resource list --resource-group ${{parameters.resourceGroupName}} --query "[?type=='Mi
40                      $databricksWorkspaceInfo = (az databricks workspace show --ids $databricksWorkspace.id | ConvertFrom-Json)
41
42                      $bearerToken = $(Build.Repository.LocalPath)/CICD/scripts/DatabricksToken.ps1 -databricksWorkspaceResourceId $da
43
44                      Install-Module -Name azure.databricks.cicd.tools -Force -Scope CurrentUser
45                      Import-Module -Name azure.databricks.cicd.tools
46                      Import-DatabricksFolder -BearerToken $bearerToken -Region $databricksWorkspaceInfo.location -LocalPath $(Build.R
47
```

Firstly, we need to know in which subscription dev databricks workspace is created. So for that we are passing service connection that we created in azure devops which will have information about subscription and resource group.

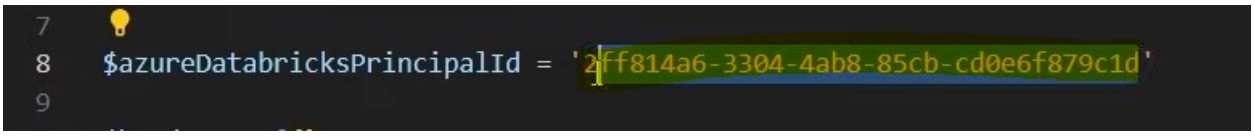
Using service connection credentials , the above powershell code (from 37-46) lines can access the dev databricks workspace . In the power shell script, firstly we need to get information about dev azure databricks workspace so for that we are using power shell script function like az resource list which will list the resources in the resource group that we are passing as a parameter and then we are specifying the type as a data bricks workspace and there by retrieving the details of databricks workspace and that details will be assigned to this variable **$databricksWorkspace .**

After that we will get the information about dev databricksWorkspace by using powershell function az databricks workspace by passing id of the databricks workspace that we got in the previous step(39 line).

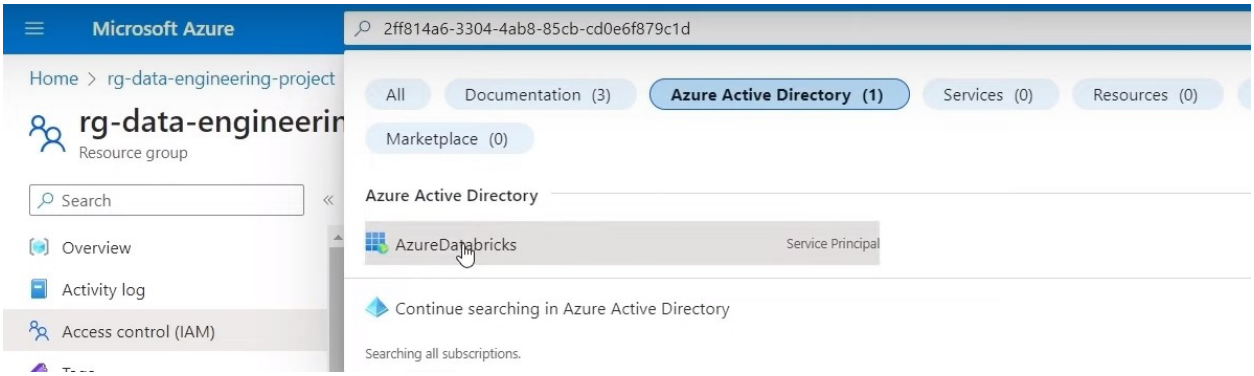
Once this step is done, we will have all the information about dev databricks workspace . Now we have all information about databricks workspace but in order to access databricks workspace, we need to generate a bearer token.so for generating the token, this template file will call DatabricksToken.ps1 file as we are specifying the path of DatabricksToken.ps1 file along with some of the other parameters that will be needed to create the token which are databricks workspace ID and workspace URL. This workspace ID and workspace URL was retrieved from the previous two steps. Once this parameters are sent to the DatabricksToken.ps1 file, bearer token for accessing data bricks workspace will be created using below powershell script.



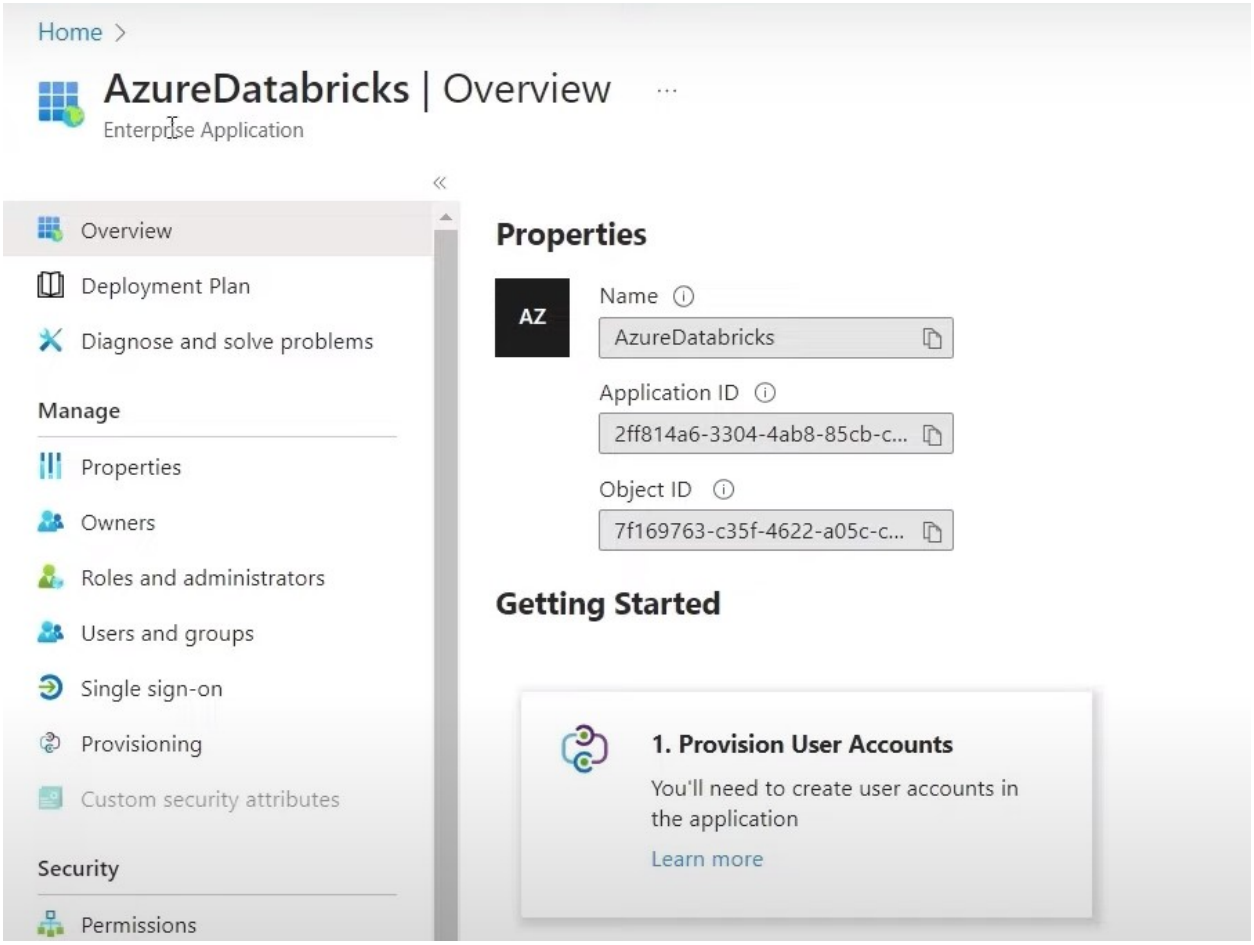
Parameters used to create token in DatabricksToken.ps1 file are databricks workspace Id and URL and then it uses dedicated principal ID for azure databricks which is common for every one across the azure tenant.



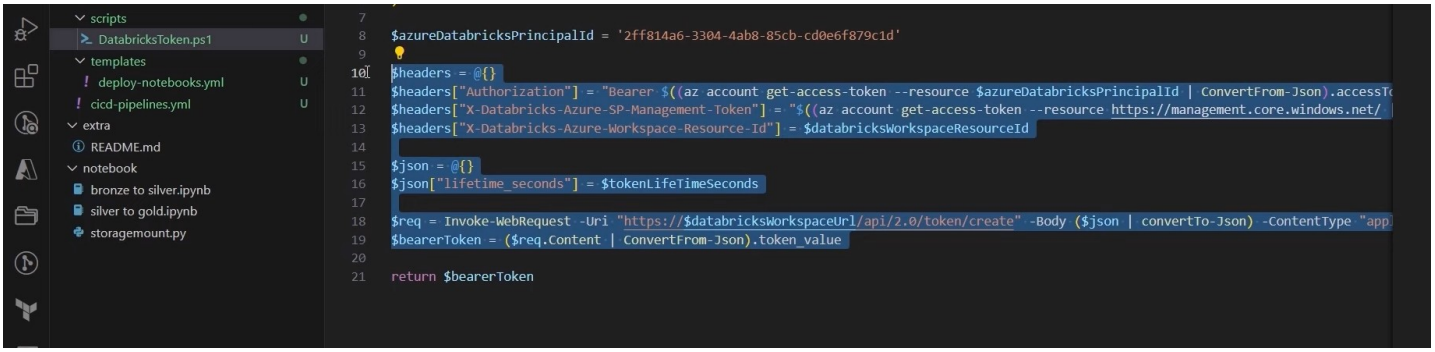
Will copy the above id , will go to azure portal, now we will paste this id in search box, we can see azure databricks in azure active directory as shown below.



Once we open that, this is the enterprise application in the tenant level which represents the databricks resource in azure.



So, we will be using all of these parameters in the below powershell script to generate the bearer token and that token will be returned back to the template file and will be assigned to $bearerToken in deploy-notebooks.yml file.

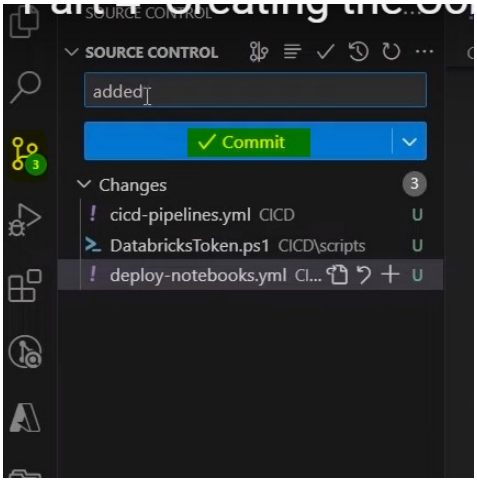


Now using this token, we need to perform the actual deployment functionality which is copying all notebooks inside notebook folder and put that in live folder in dev workspace .

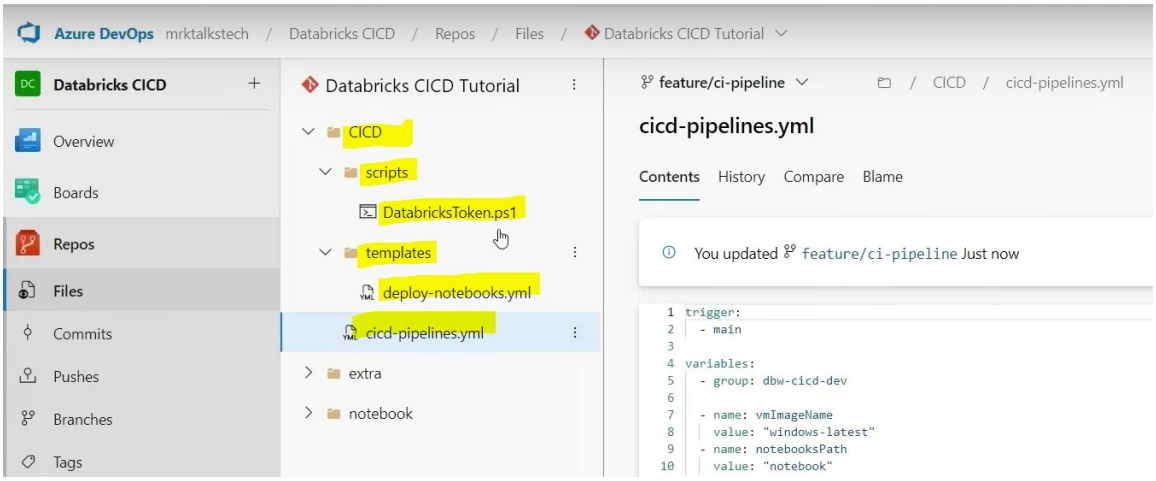
```
Install-Module -Name azure.databricks.cicd.tools -Force -Scope CurrentUser
Import-Module -Name azure.databricks.cicd.tools
Import-DatabricksFolder -BearerToken $bearerToken -Region $databricksWorkspaceInfo.location -LocalPath $(Build.
```

SUMMARY: we have a master yaml file which is cicd-pipelines.yml. In this file, we are getting all these variable values from variable group and these are passed as a parameter to the template file which is deploy-notebooks.yml. so, this template file will use service connection to get information of dbws in the dev resource group and then using that information like the ID and the workspace URL we are generating a bearer token by calling another file DatabricksToken.ps1 . This fie will use both dbws id and URL along with the principal ID and will generate bearer token and finally return back the token to the template file. now this bearer token we use to access dbws to copy all the notebooks that are inside notebook folder and deploys it to live folder in the dev dbws. So, this is the continuous flow of the continuous integration pipeline.

Now we have committed all our changes in our feature branch. Click on commit and then click on publish branch.

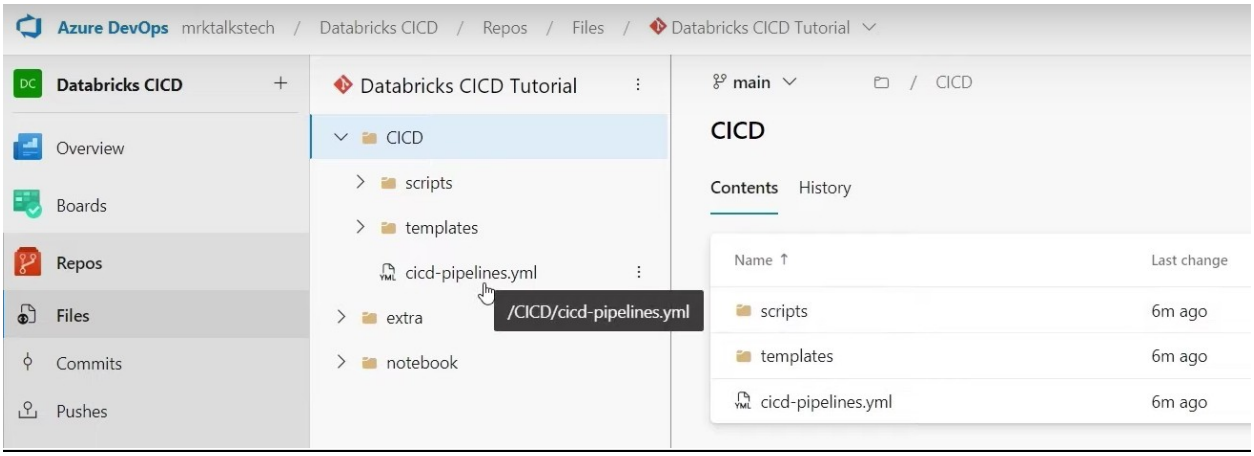


Now we can go to azure devops and we can see our changes was reflected in devops repo as shown below.

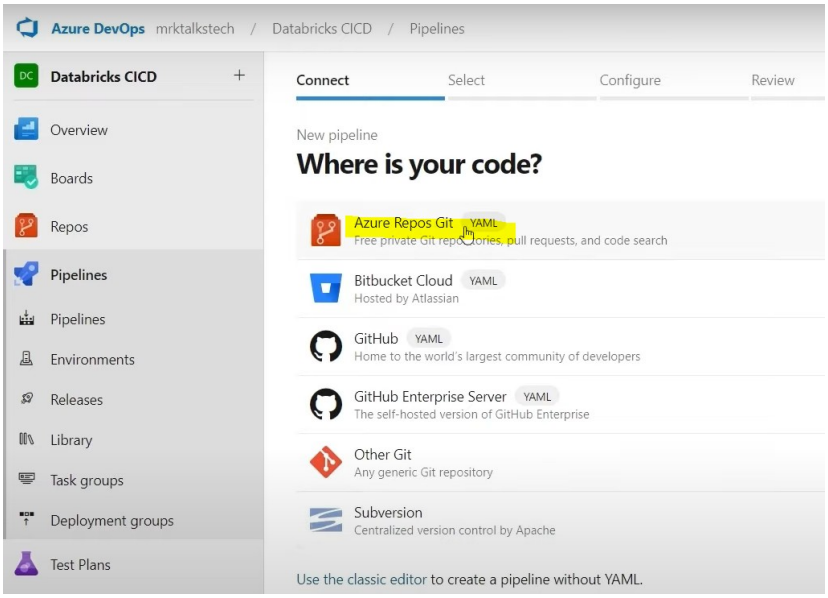


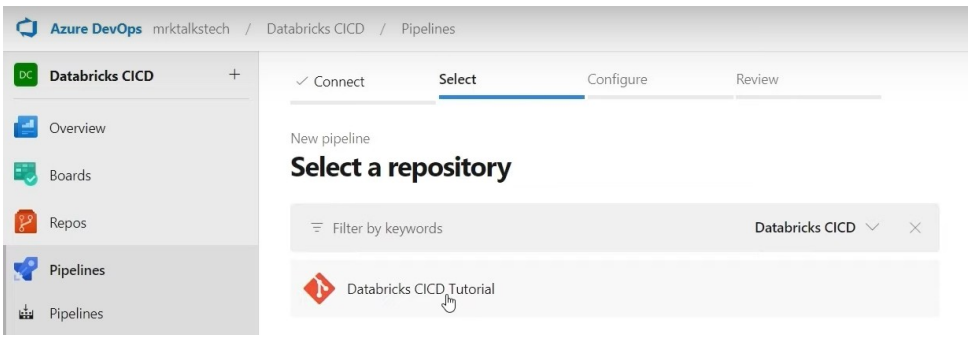
Now before creating pipelines, we need to merge above changes to main branch. The reason for this is if we create pipeline first and then merge the changes to main branch then the pipeline would start immediately since there will be an update happening in the main branch so for that we will merge changes to main branch first.

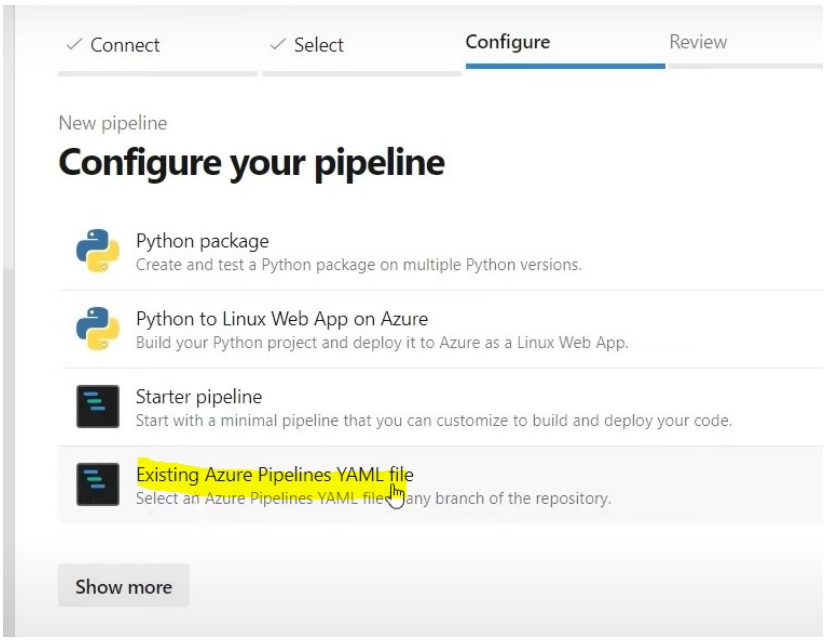
We will create pull request and will merge changes to main branch.



Now go to pipeline section in azure devops → Create Pipeline → will select the below things

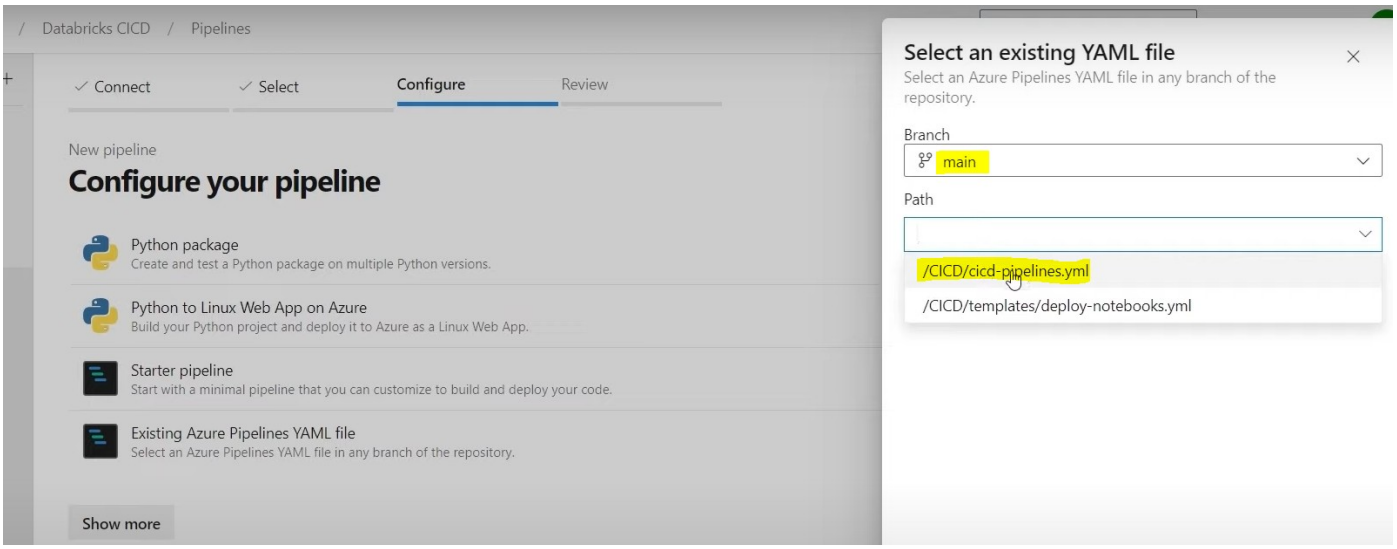




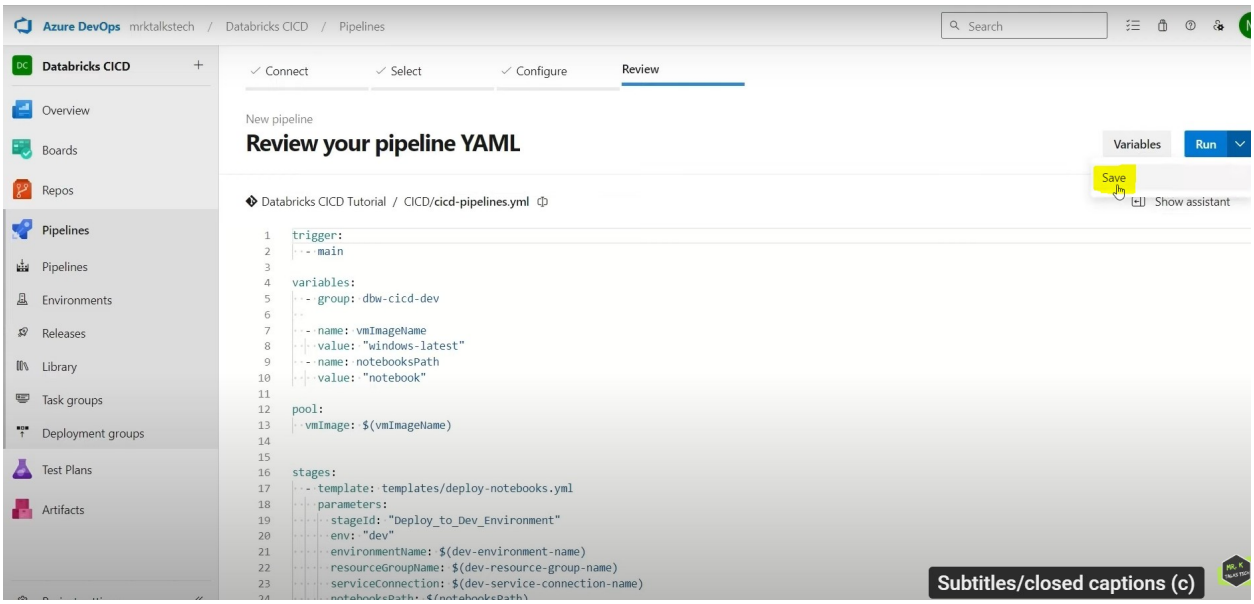


Here we are selecting Existing Azure Pipelines YAML file, the reason for this is since we have already created master file YAML file, we can use the same existing YAML file to create this pipeline. We can also use starter pipeline if we are creating pipeline from scratch using azure devops itself.

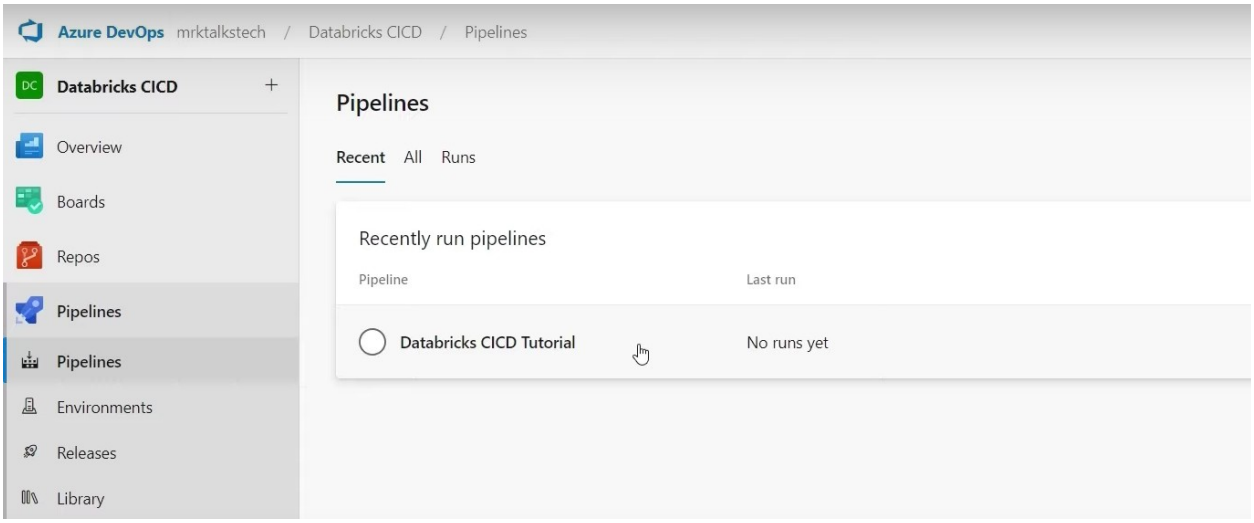
After selecting above highlighted one, we need to select master yml file from drop down



Before creating pipeline , we can review the code in the next step as shown below.



After reviewing the code, we need to click on save as shown above. If we click on run, it will run pipeline straight away.



Now our pipeline is created as shown in above screenshot. Before testing the pipeline, we need to give access to dev environment so for that go to pipelines → Environment → open our created environment → click on 3 dots at top → security → there in pipeline permissions section we need to add our created pipeline name.

Similarly we have created variable group for dev, pipeline should have access to this variable group as well to read the variables that are created inside variable group.

Go to Library → open created variable group → go to pipeline permissions → here we can add our pipeline to this variable group. Now our pipelines can read all the variables from this variable group .

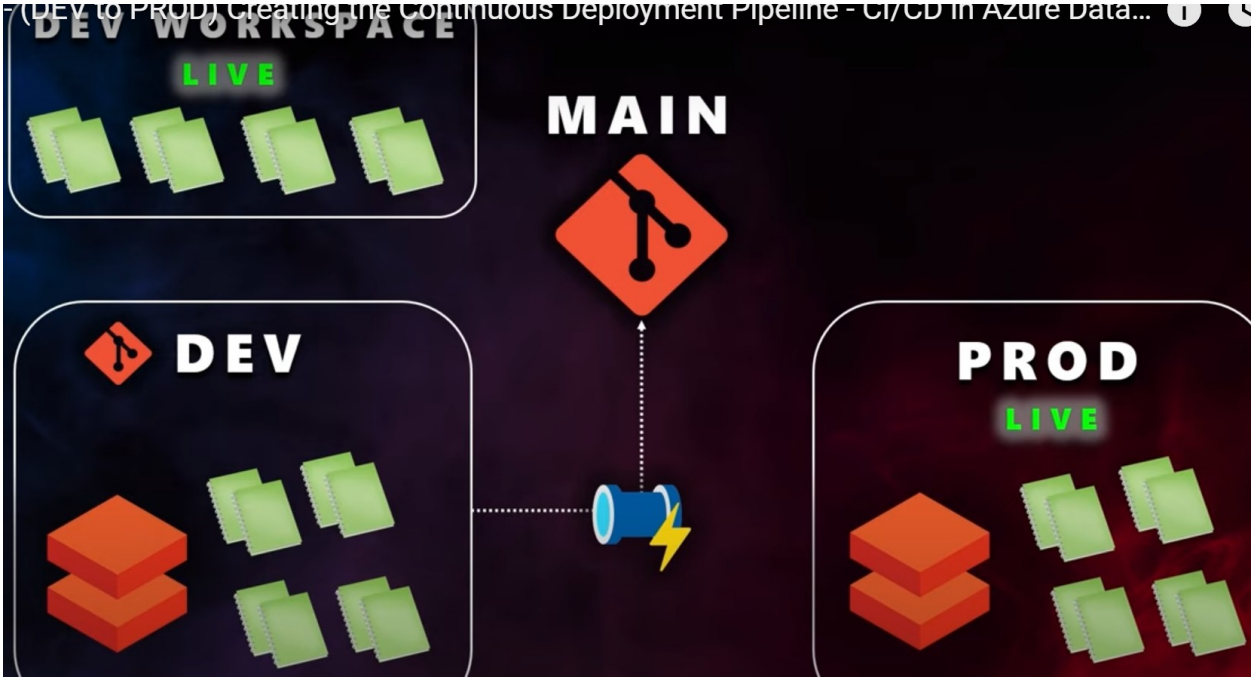
Similarly go to service connection in project settings and give access to pipeline.

Now pipelines permissions was given for environment , library and service connection. Now CI-CD pipeline have access to all of these services that we have created such as environments, variable groups and service connection.

Now we can test the pipeline. As soon as we merge changes to the main branch, our CI-CD pipeline should gets triggered and should copy all the latest notebooks from main branch and should deploy it to live folder in the dev dbws. Now in ADF pipeline, we need to change notebook paths from workspace shared/user folder to repo live folder path so that pipeline will use always latest code and now publish changes in ADF pipeline.

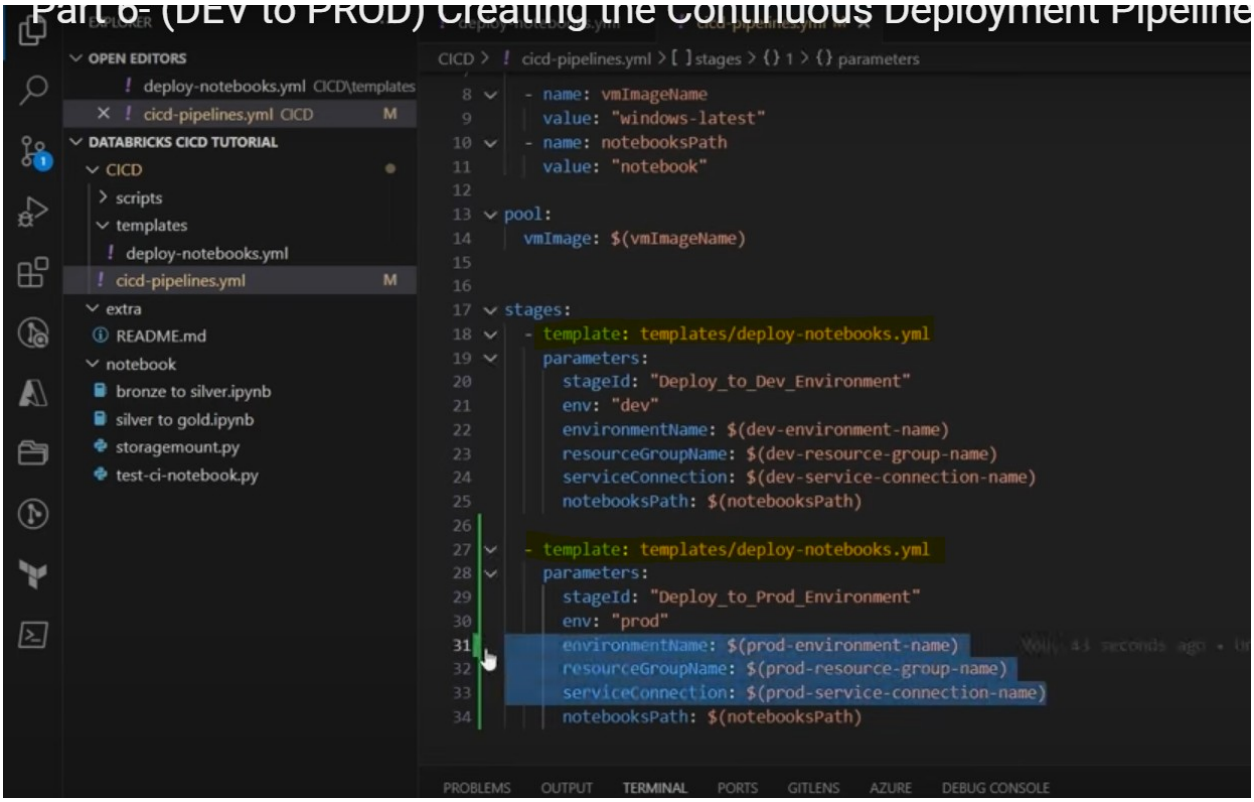
(DEV to PROD) Creating the Continuous Deployment Pipeline - CI/CD in Azure Databricks

As soon as continuous integration pipeline run is finished , continuous deployment pipeline should get triggered and this pipeline will again take latest notebook from main branch and copies it to live folder in the prod dbws. Here, both the live folder in the dev dbws and prod dbws will be exactly having same code which is exactly in the main branch code.



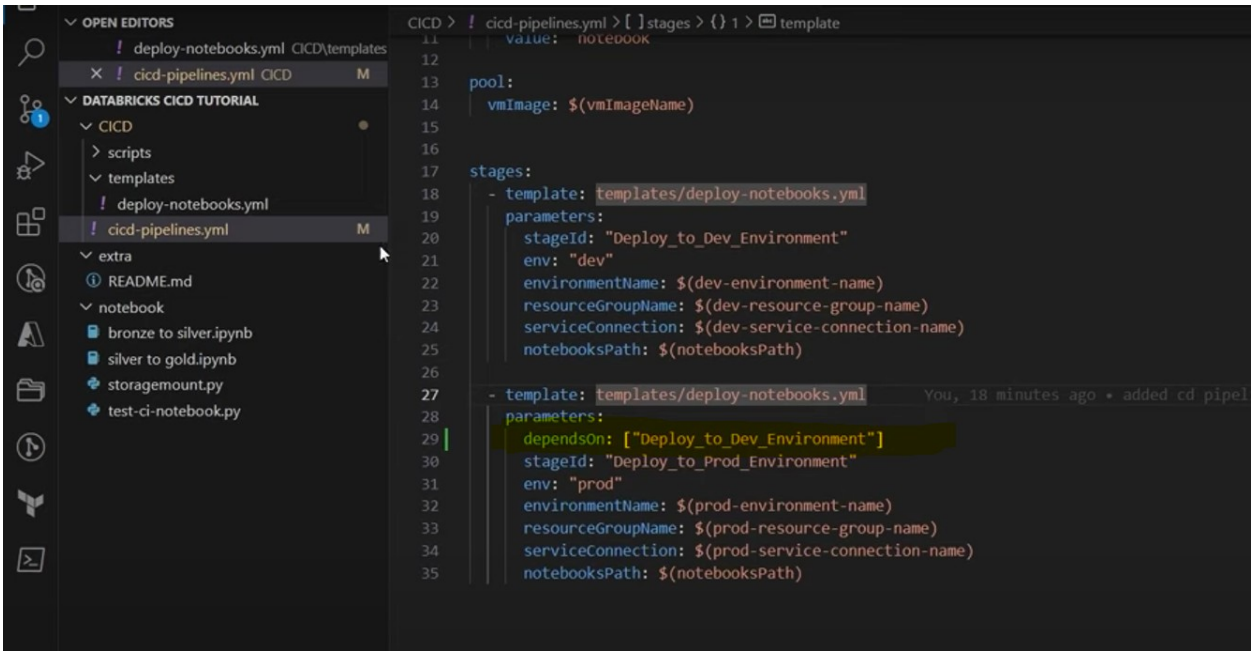
Similar to CI, for CD also we will do same process.

First creating feature branch to make changes in yml file by adding prod details in the files.

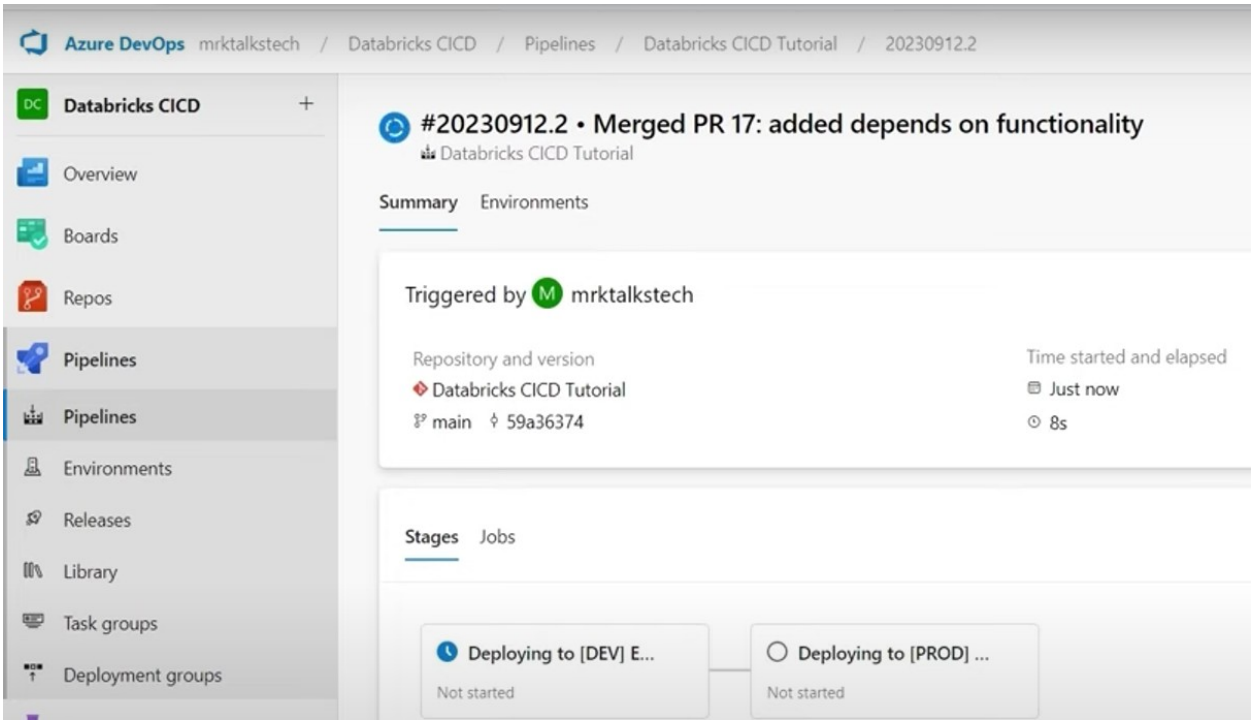


Now also we need to create variable group, environment, and service connection in azure devops in prod environment and finally add those names as a variable to the prod variable group. Give pipeline permissions to environment, service connection and library. Now we need to commit above changes in feature branch in dev environment and create a pull request to merge changes to main branch. Once we merge changes to main branch , our CI-CD pipeline will get triggered and it will again copy all the notebooks which is inside main branch and deploy it to live folder in dbws in both dev and prod . once dev environment deployment is completed then prod deployment should start. But here it's not happening like that because we haven't mentioned dependency in code of dev and prod environments.

So we need to include extra 1 line code dependsOn in master yml file as shown below.

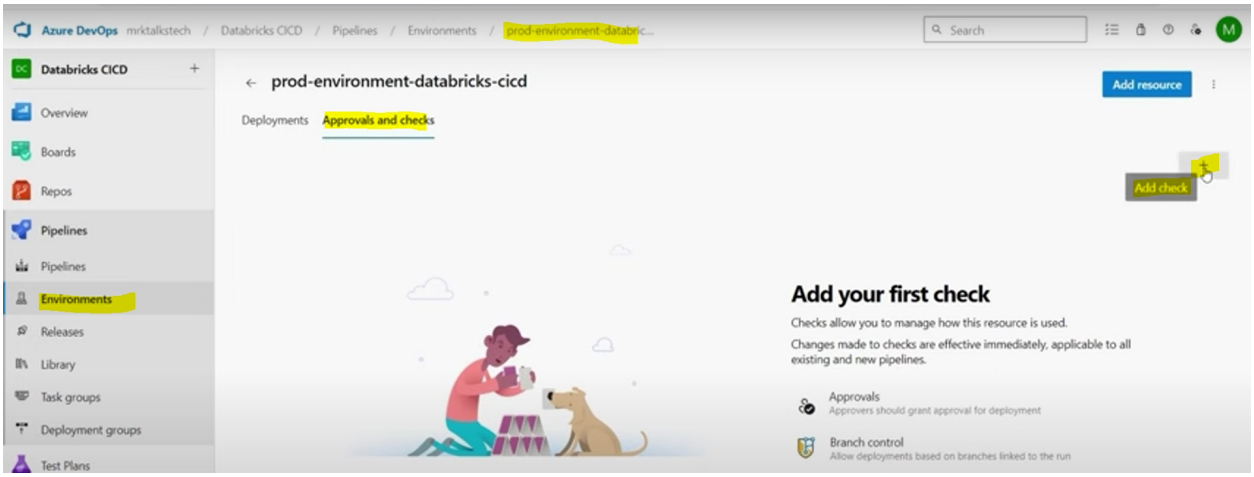


Now after adding above 1 line code and committed changes in feature branch and merge changes to main branch ,pipeline will start and its set up has changed as shown below.

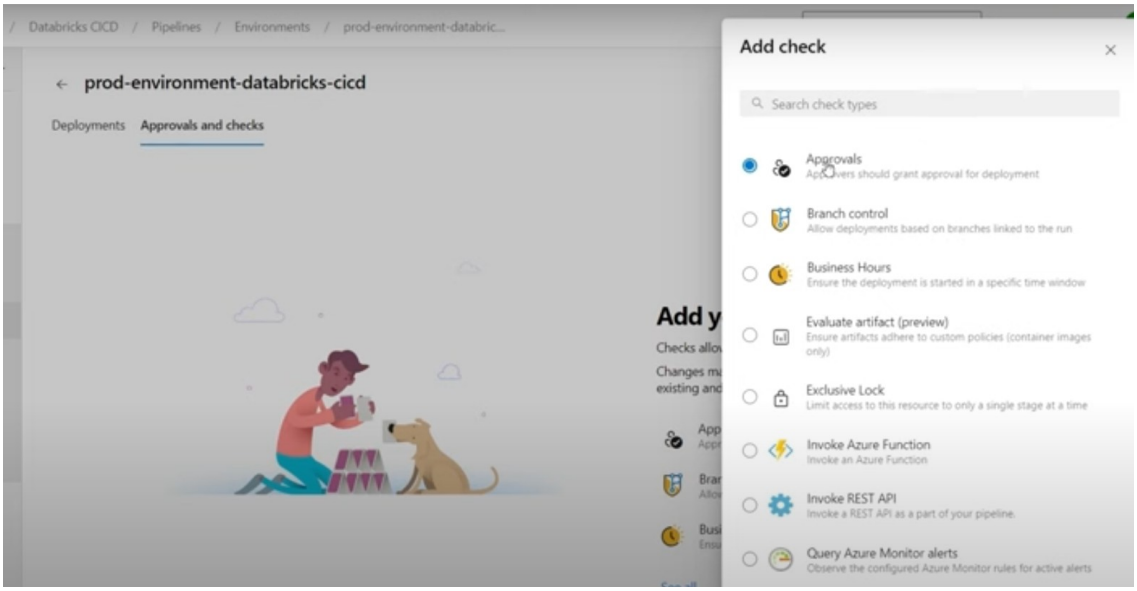


Here as soon as our dev stage is completed, prod stage is beginning to deploy changes to prod environment. So, this pipeline setup is not an optimal solution in the real time projects. Deployment to prod environment should not be happening immediately unless there is a proper planning or approval from higher technical or operational team in a company. In most of the organizations, any deployments to prod environment requires a kind of change request to be raised and approved. In some companies, deployment to prod environment should happen only after office hours say for example 5 pm. In this case we need to make this pipeline to wait before actually deploying this changes to prod environment. So, this can be achieved by configuring environment protection which means for example, once the dev stage is completed, an email will be sent to team lead or someone from operational team which says pipeline is waiting to deploy the changes to the prod environment and only when they approve it, prod stage will run the deployment otherwise pipeline will wait until they approve it. Also, they will have privileges to reject the pipeline run for some reason , changes will not be deployed to prod environment. This is the main functionality we need to configure in CI-CD pipeline.

We need to go to below path



We have different types of checks to configure ci-cd pipeline



Click on continue, we need to select approver of higher person like team lead . once we add them as a approver after the dev stage is completed, approver will receive a notification via email asking them to approve CI-CD pipeline to deploy the changes to PROD environment.

Now we protected prod environment, now when our CI-CD pipeline runs , dev stage will work as usual and once the run is finished , approver will be notified via email and only when approver approves the pipeline , prod stage will run and deploy the changes to prod environment.

Summary:

Firstly, we will create our feature branch based on main branch, and will move our notebook changes to feature branch and will commit our changes in feature branch. Now we will create pull request in azure devops repo . Also, we will create pipeline in dev environment where there will be yaml files for deployment. Also, we will create environment variables, service connection and variable group in library section and we will provide pipeline permissions also in that 3 environment variables, service connection and variable group services.

once pull request is approved, changes will be merged to main branch and feature branch will be deleted . And latest changes in main branch will be moved to dev dbws in live folder. Now for deployment to prod, mail will go to higher person in team(TL) . once he approve the request, then only changes will be deployed to prod environment. Once changes deployed to prod main branch, files in main branch of prod will be copied to live folder in prod dbws .