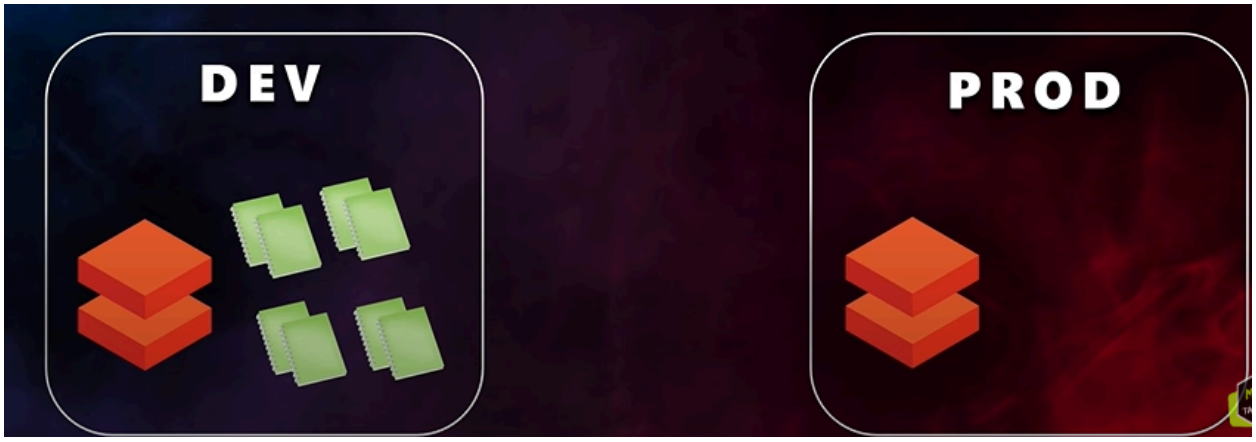
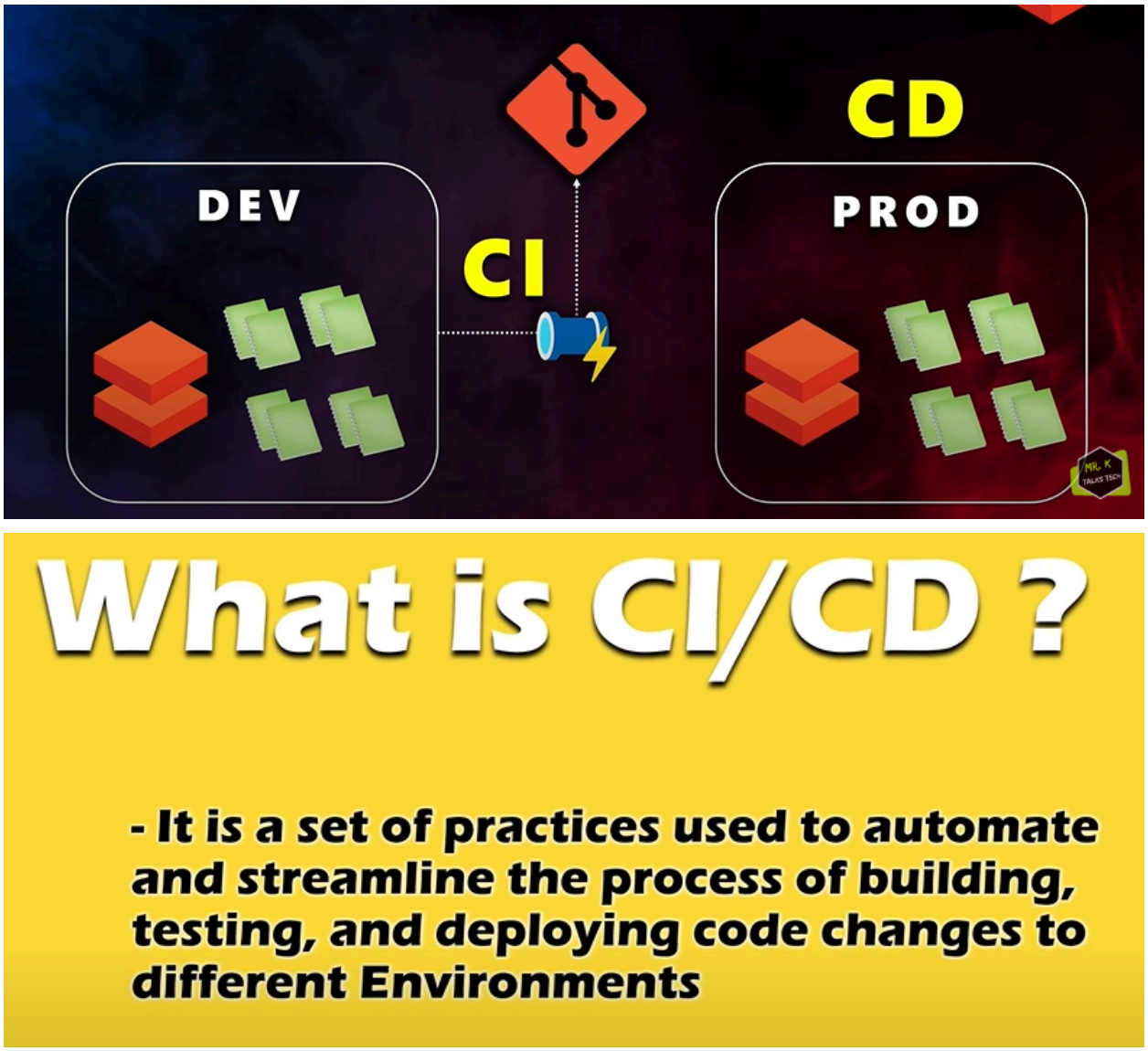
What is CI/CD?

1. Lets first see ..how we are going to implement CI/CD for azure Databricks
2. Imagine we have a dev resource group with databricks resource attached along with some notebooks in it 

3. So the main idea of CICD pipeline ..is to deploy the code from one env to another env
4. Here the another env is PROD…prod has similar resources as dev..but it does not involve developing the code



5. What it means..is..we'll write the code in dev env…and using CICD we need to deploy it in PROD env
6. So for doing the CICD ..the most imp thing is the git repository..and here we'll be using azure DevOps repo to implement CICD
7. In simple terms..the repos will save all our code …and we'll integrate our repo with dev env
8. Here…once we've completed all the work in DEV env…we save all our code and commit it in our REPO…now as soon we commit our changes…the CICD pipeline gets triggered and deploys all the DEV changes to the PROD

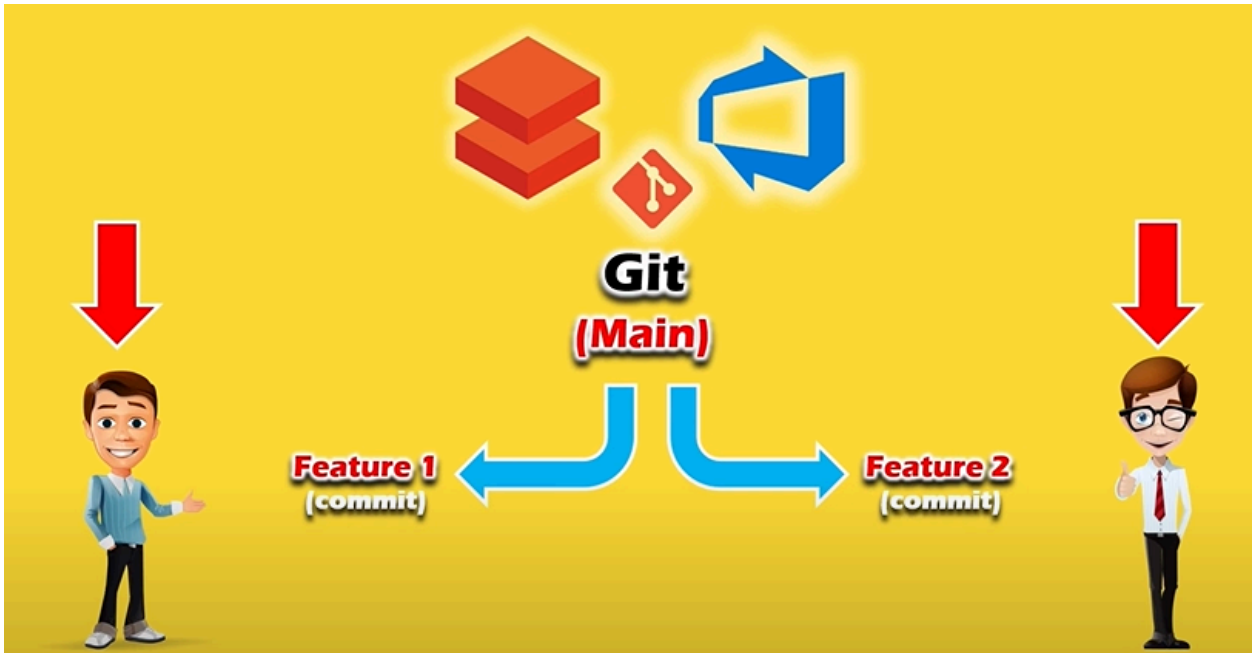


Merging/Branching Techniques

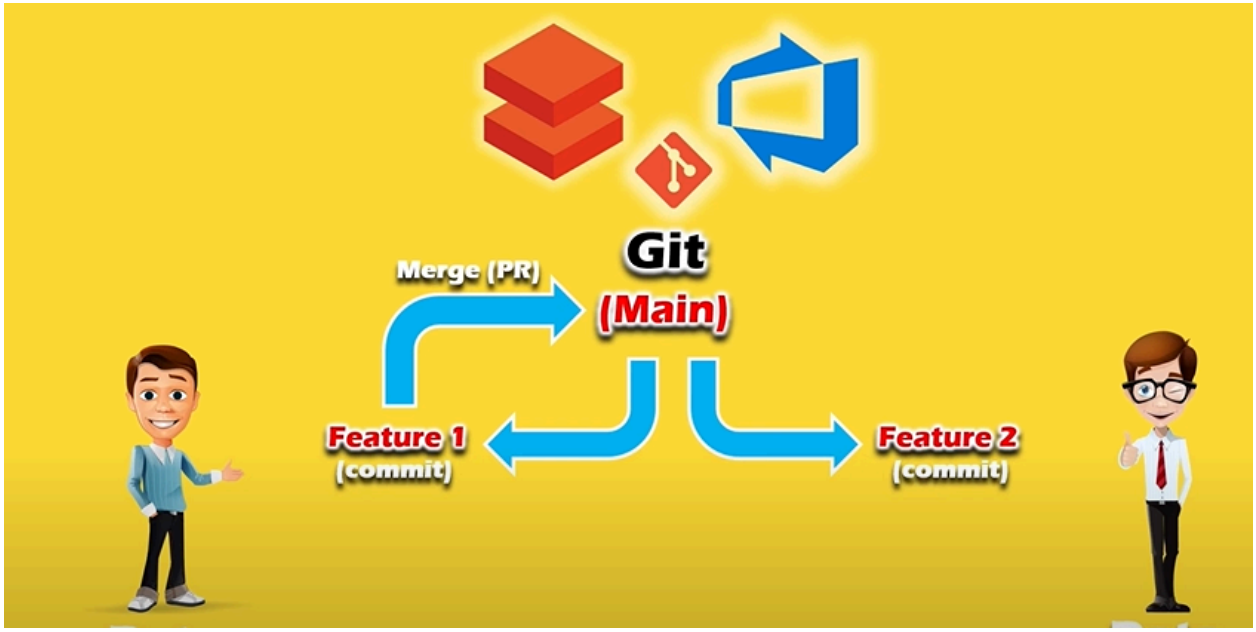
1. There are many ways for merging techniques..lets see one with practical example
2. Imagine we have two data engineer's and they are working together on one databricks workspace and this databricks is in DEV env

3. Important thing is to integrate azure devops repo to our dev env

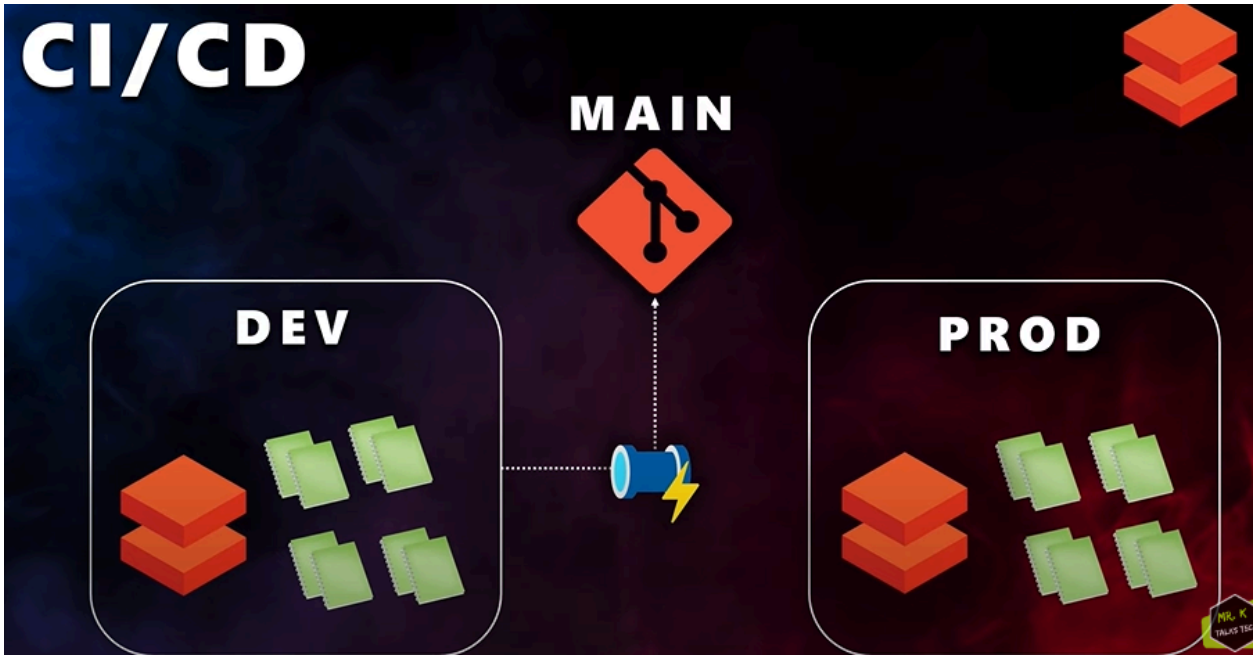
4. And while integrating ..we need to create a main branch in this repo ..this main branch is very imp..like whenever we commit anything in the main ..then CICD pipeline gets triggered
5. So we'll protect this main branch…no data engineers ..can directly commit to the main branch..and if incase they commit..we need to produce an error..so this is branch protection
6. Here each data engineer's must work on their own features by creating separate branches..and make commits in their own branches



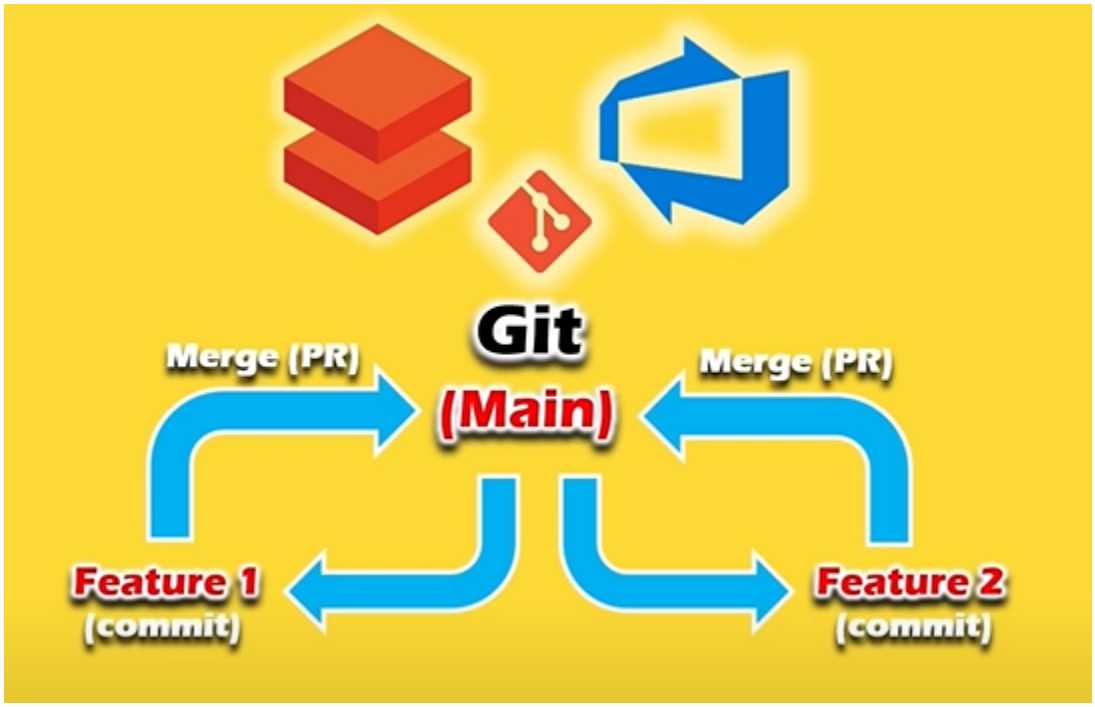
7. Now the question may arises..like which branch should be merged with main branch..then the team of data engineer's will discuss among themselves about the priorities and decide which feature to merge
8. Let's Suppose we have decided to go ahead with feature 1
9. Now using the pull request ..we will merge our feature branch with main branch



10. Now once we have merged with main branch..then the CICD will get triggered..and will make these changes in the prod env



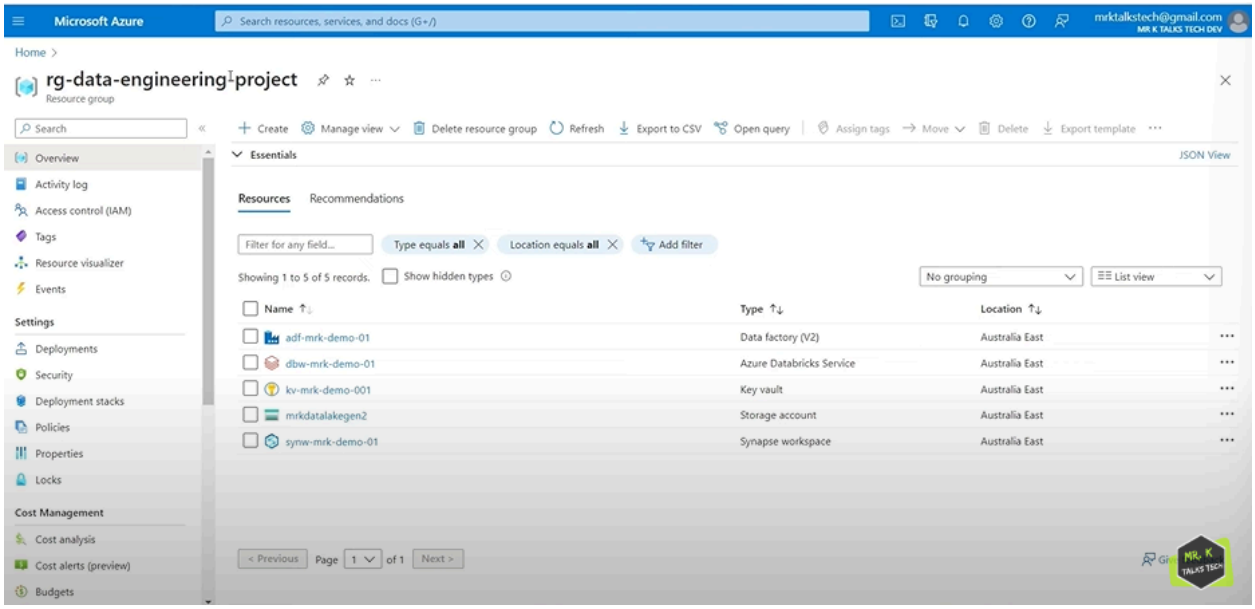
11. Once it is done..now data engg2 will merge his branch with main branch



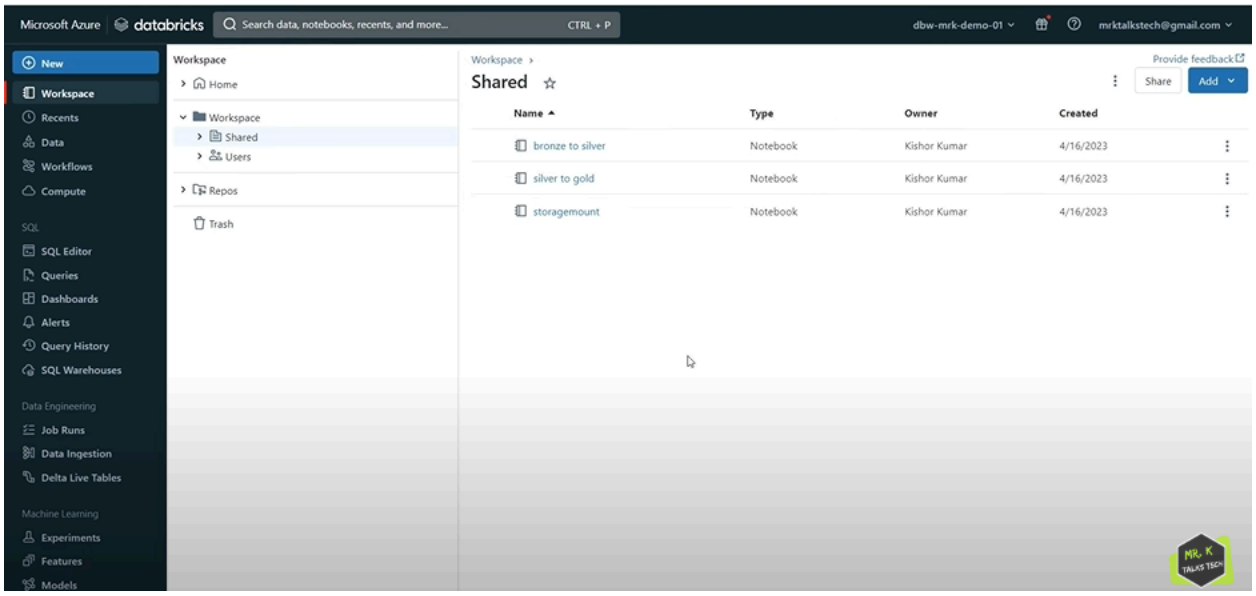
12. So this is what CICD is

GIT Integration in Azure DataBricks(Env Setup)

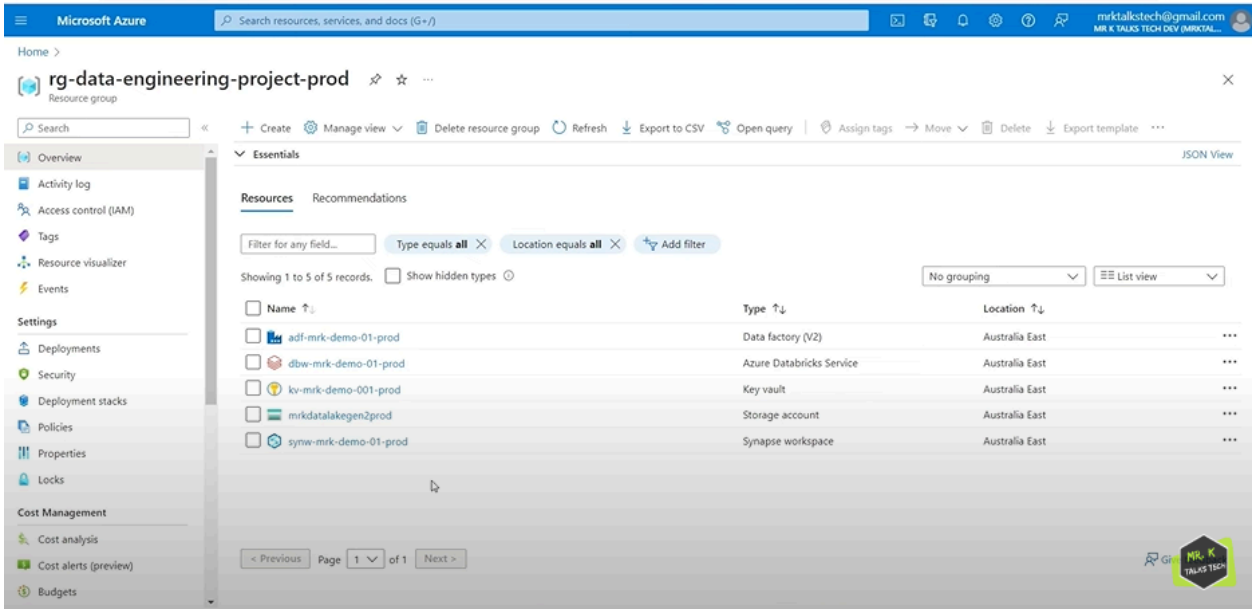
1. Lets go to the resource group of our project



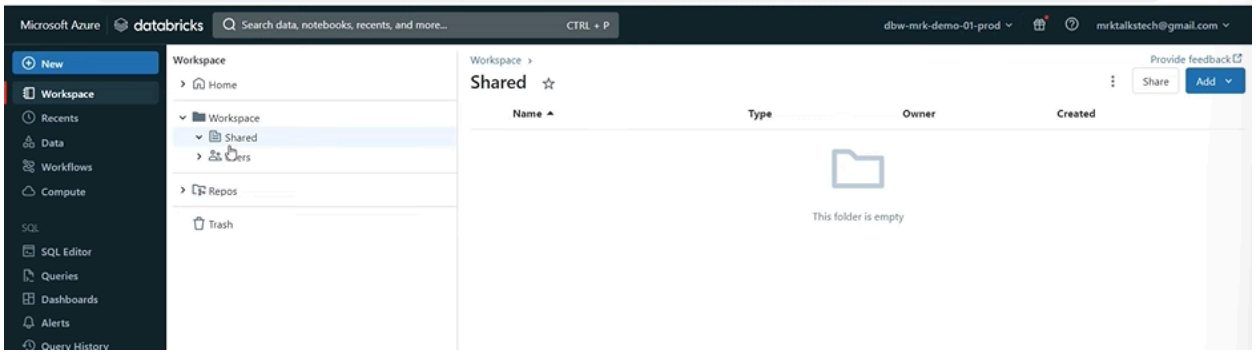
2. This are the notebooks we created for our project



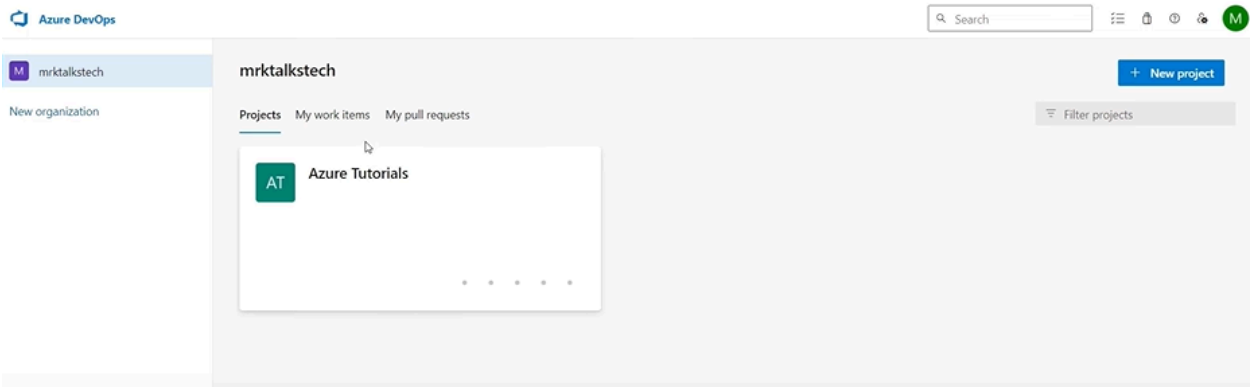
3. We have another resource group for prod env…



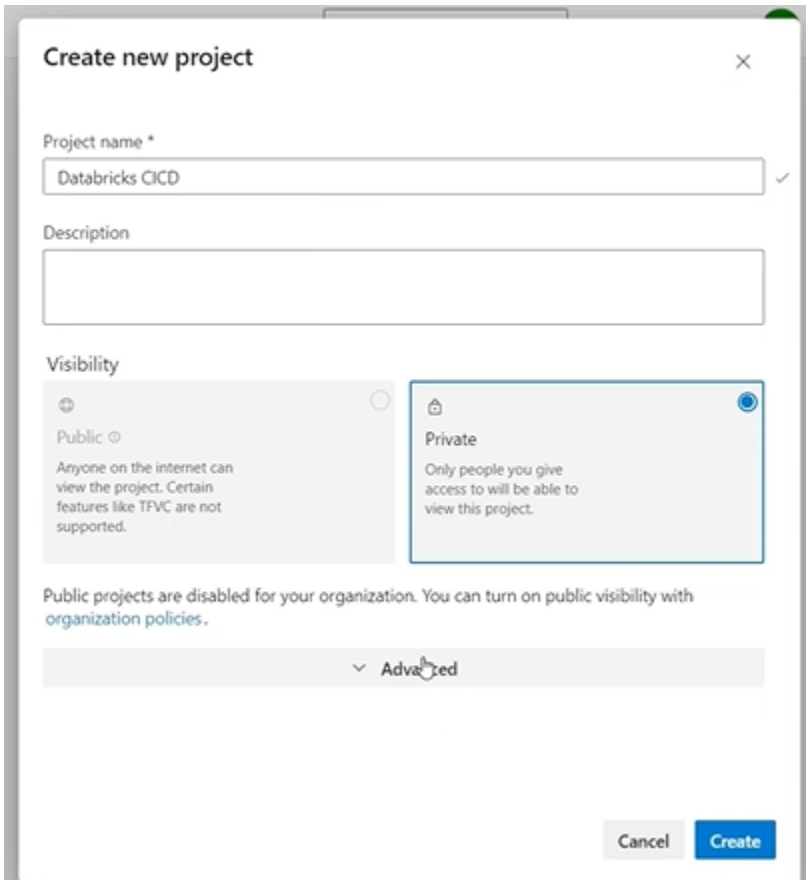
4. And now inside databricks resource in our prod..we dont have any notebooks



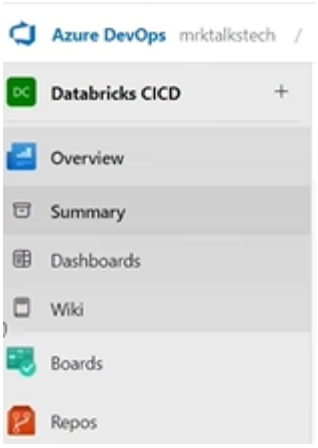
5. So now we'll be creating a repo just for DEV env
6. Lets begin integrating azure devops repo..with dev env of ours
7. We'll head to Azure devops



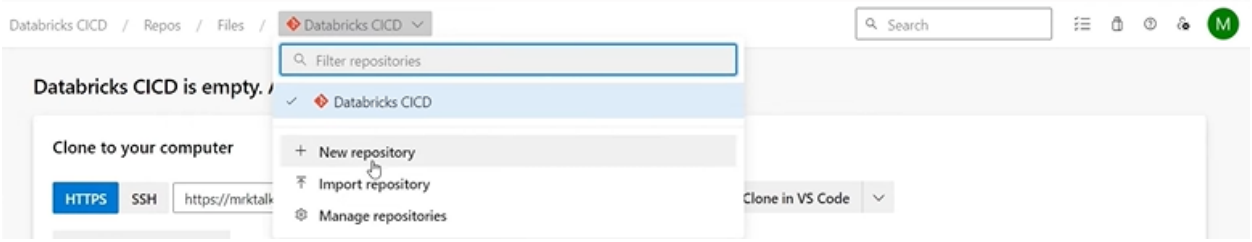
8. Now we'll create a brand new project..



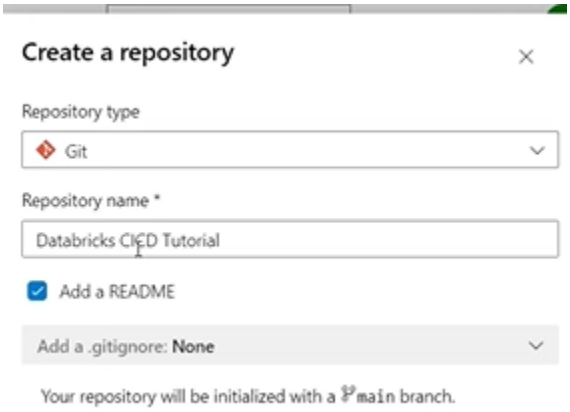
now we have created a project with the name CICD

9. Next we'll create a new repo inside the proj 

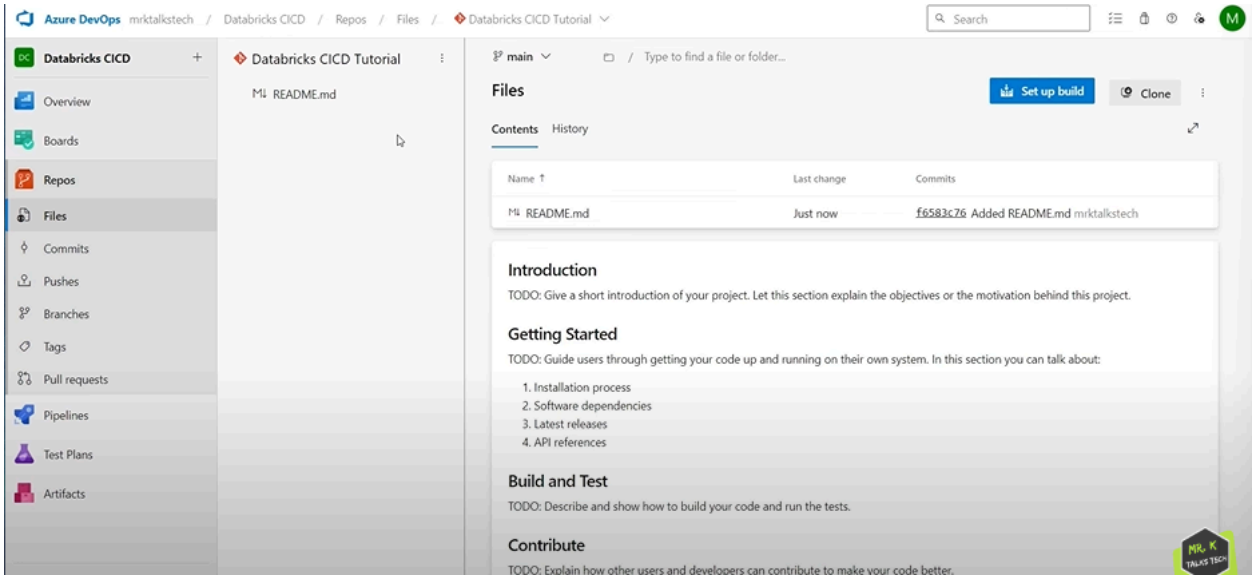
go to repos and click in new repo



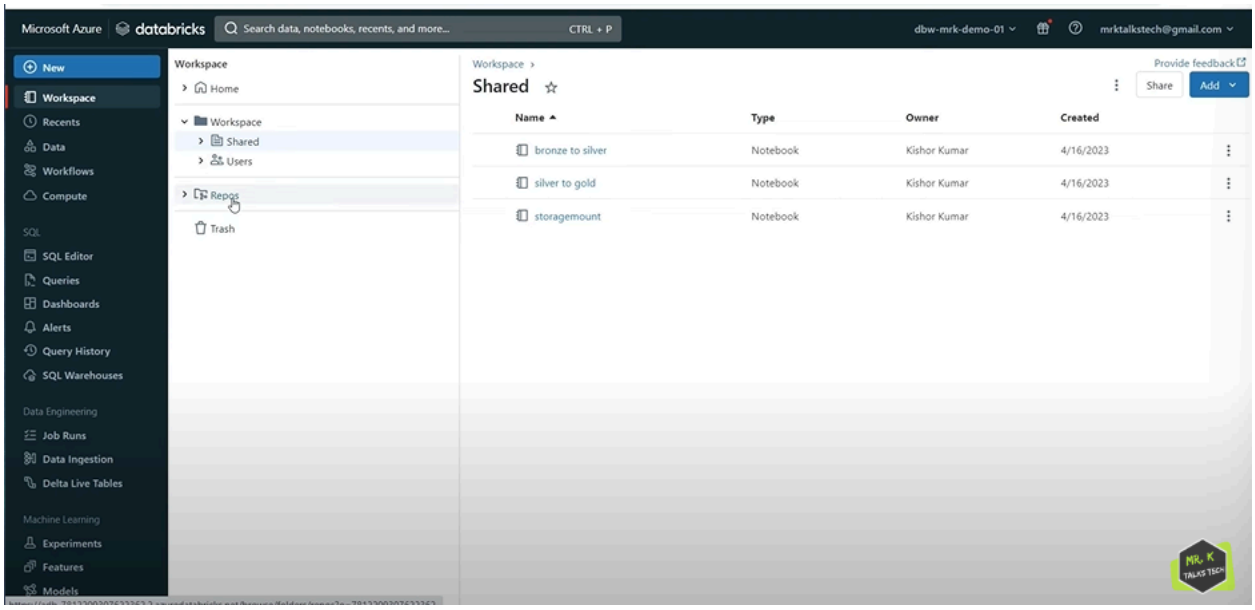
10. We'll give details about the new repo and click create



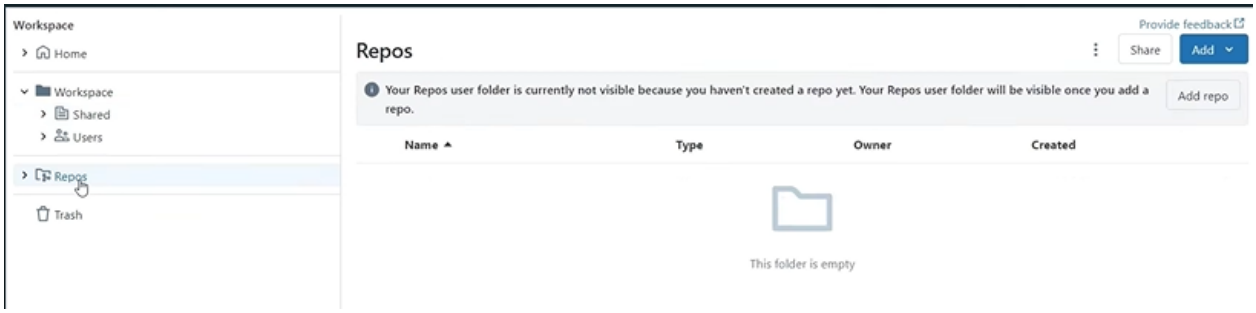
11. Now here we have created a new repo with main brancg



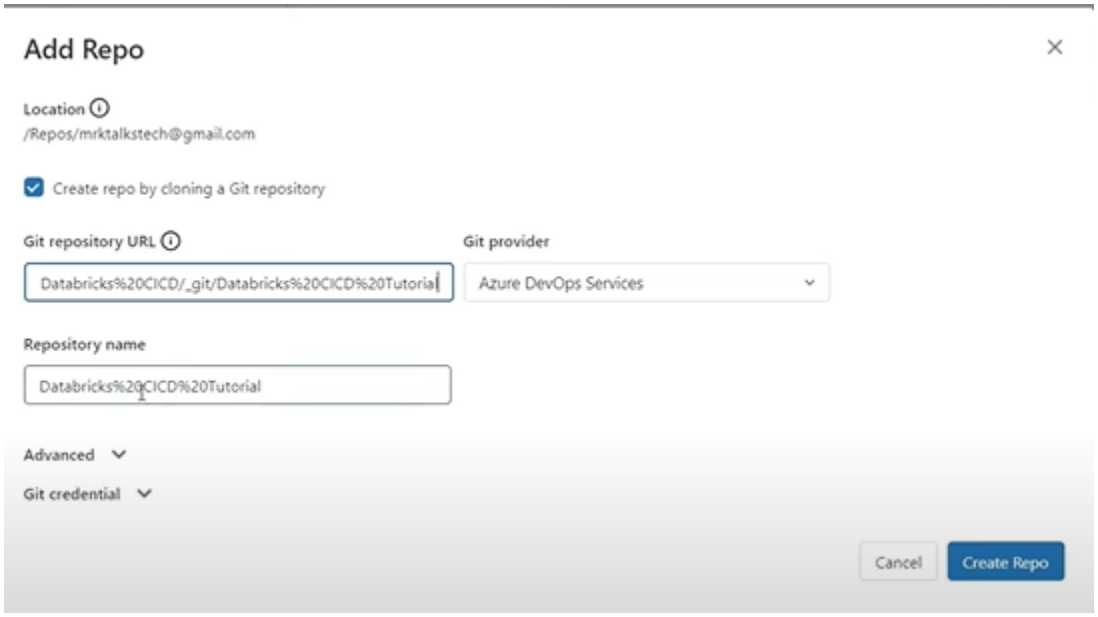
12. Lets go back to dev workspace



13. Here we have repos and click on add repo

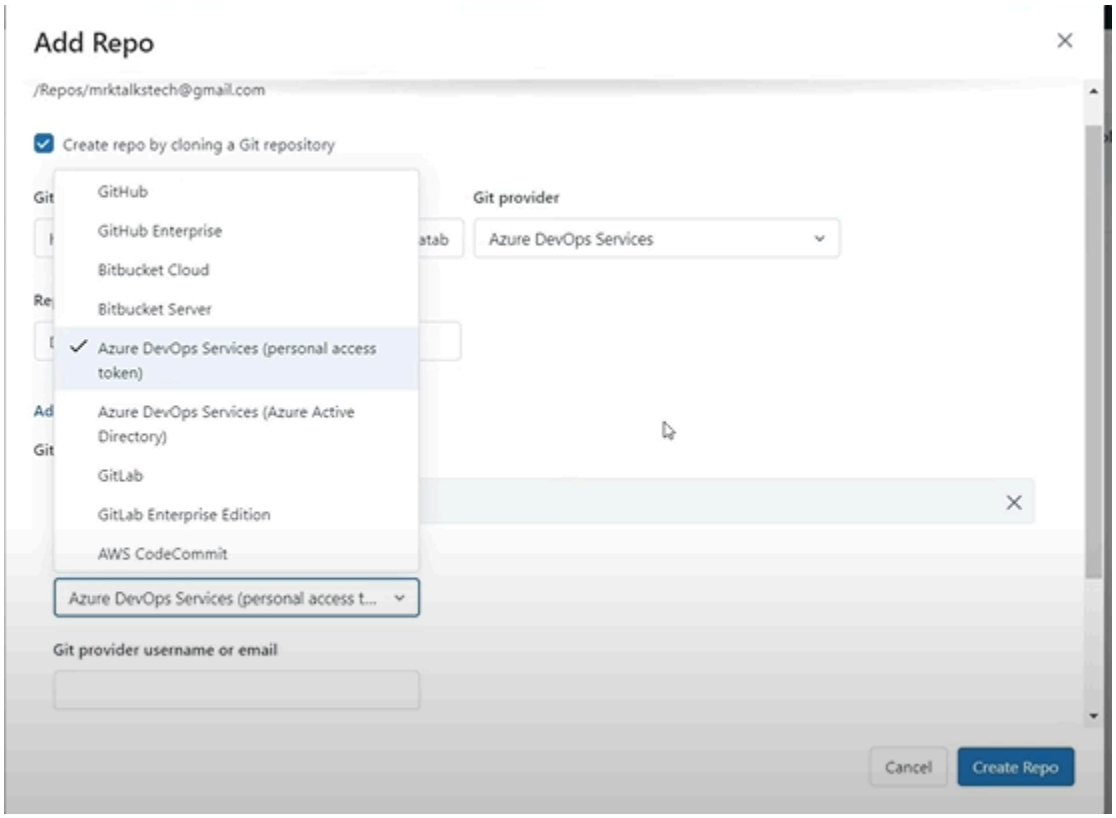


14. Now we'll provide necessary details

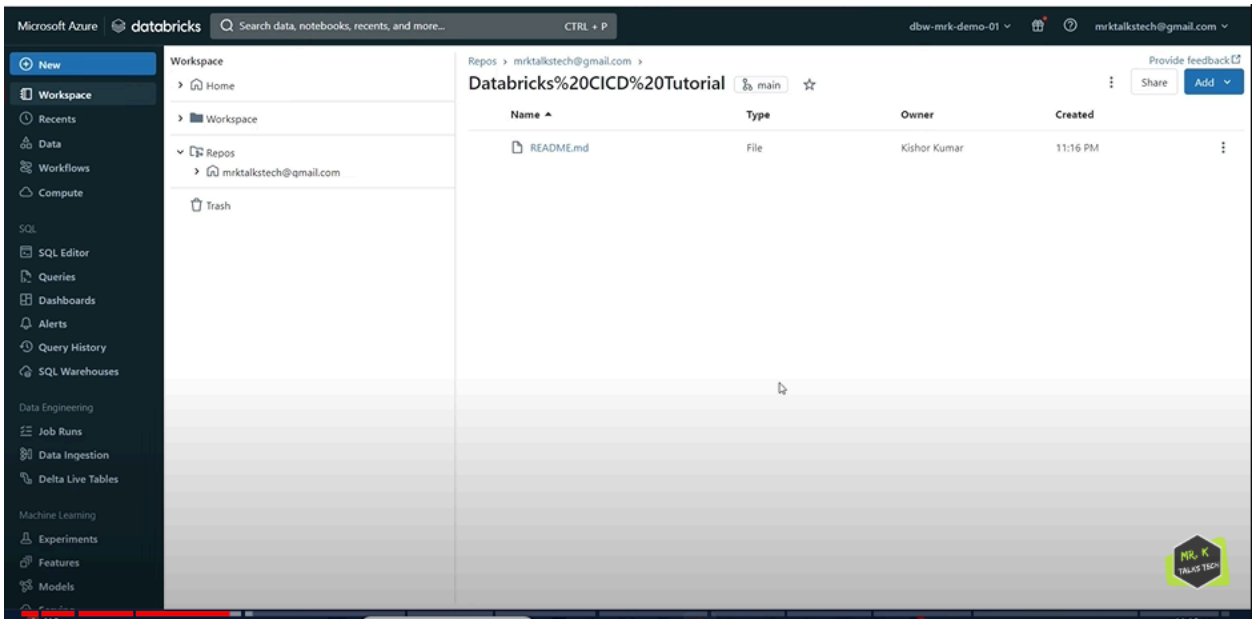


Here we have git credentials..which is very important..it used to specify ..how we'll authenticate to AzureDevops from this workspace

15. Click on git credential..in git provider ..we use Azure Devops Services(Azure AD) as provider ..it means that..as we are currently logged in as MrK in both Azure databricks and azure devops..it can provide login based on the account credentials..since this account is logged in both …and click save

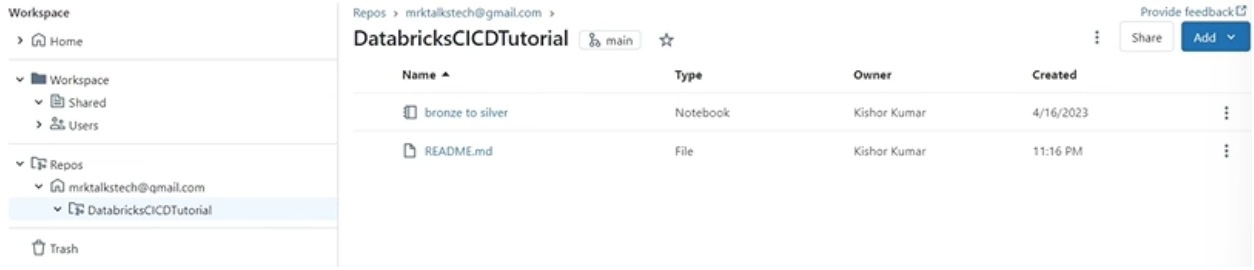


16.  Here what this means ..is ..if there is any code in Devops repo..it will be cloned in the azure databricks workspace
17. Since there is only readme file in our repo..it will be cloned into our databricks workspace
18. Now we have successfully integrated the azure devops repo in our databricks workspace



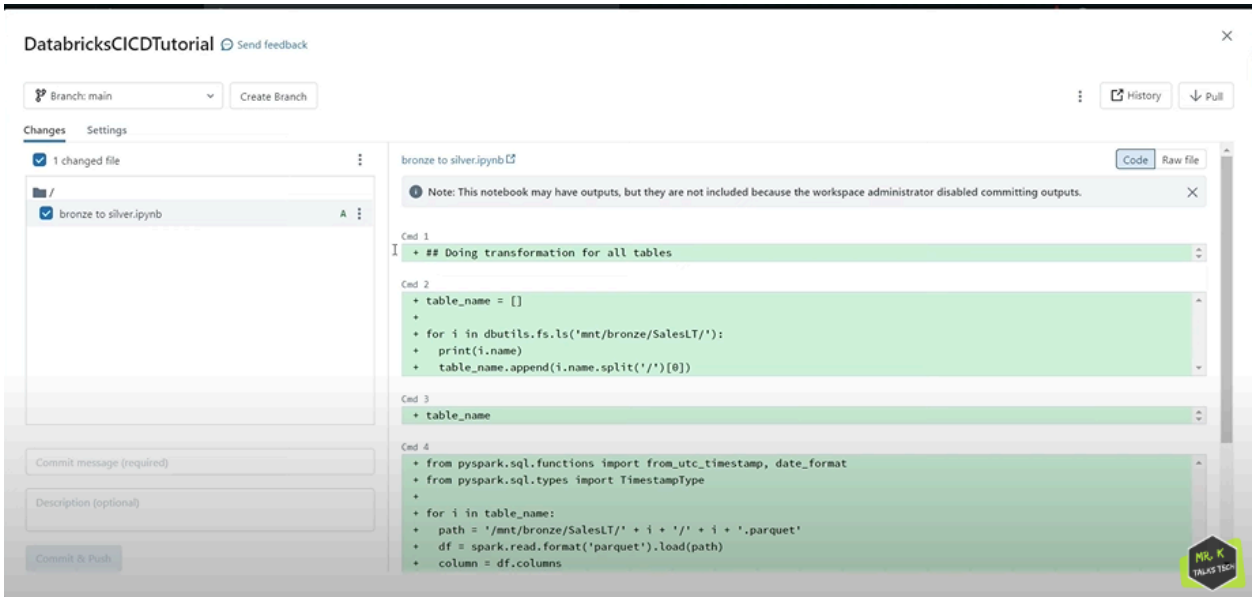
Branch Protection

1. So now we have our main branch in the databricks workspace…and we need to protect it from unnecessary commits
2. Now we'll test our main branch..by moving a notebook from our workspace to main branch

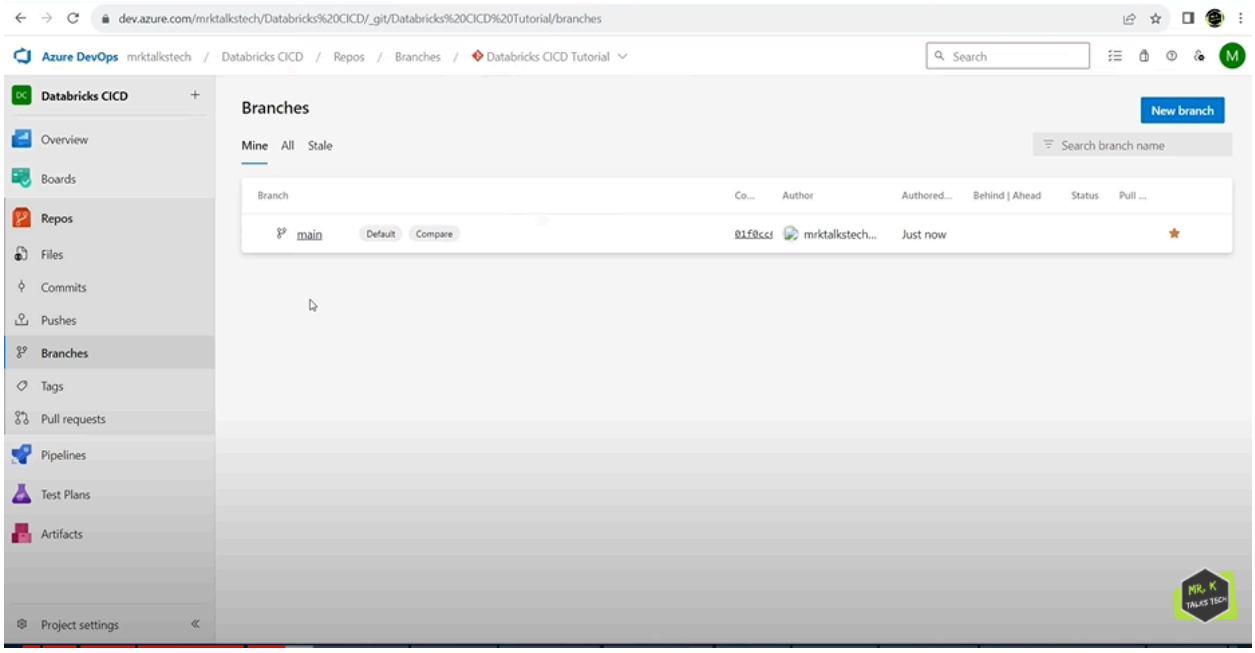


3. Since we did not added any protection to the branch..it does not give any error…while committing added
4. For committing the changes ..we need to click on 

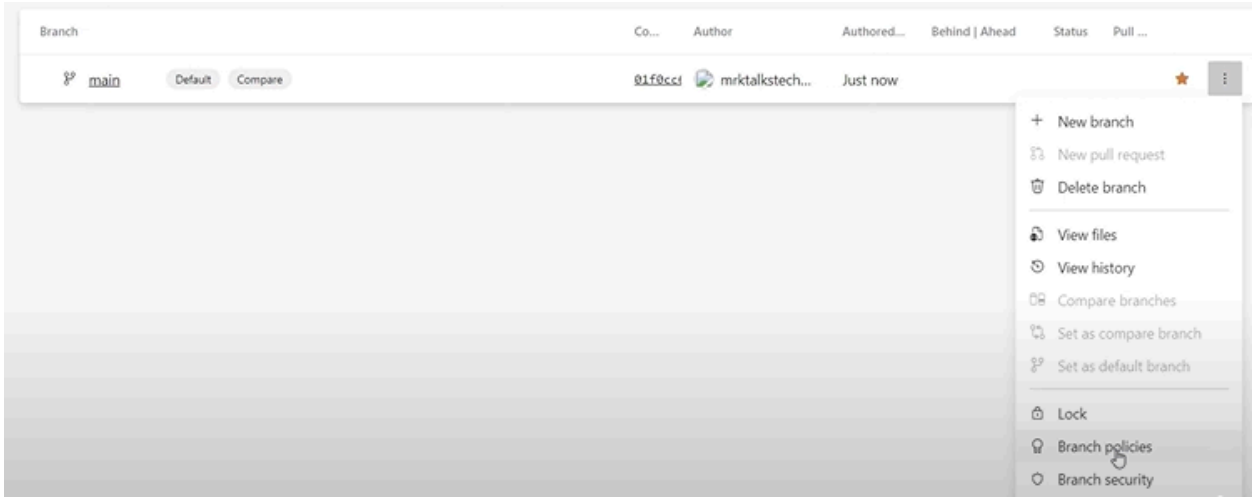
5. After committing the main repo identifies..what has been changes



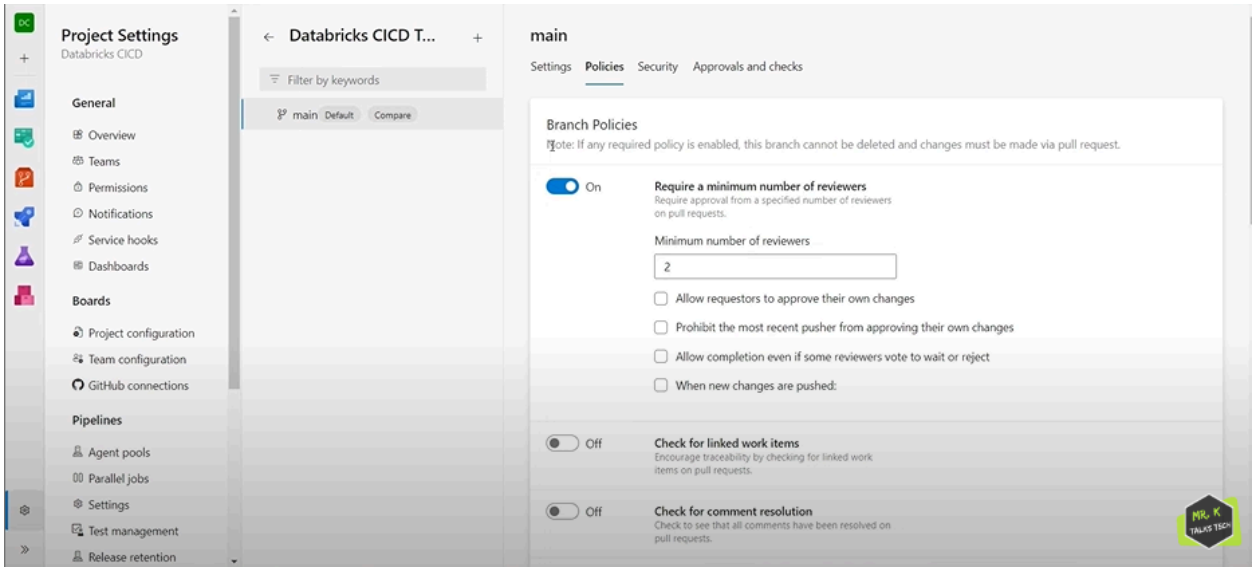
6. Now we'll give commit message and click push
7. Lets see how we can protect our main branch
8. For that go to branches



click on branch policies

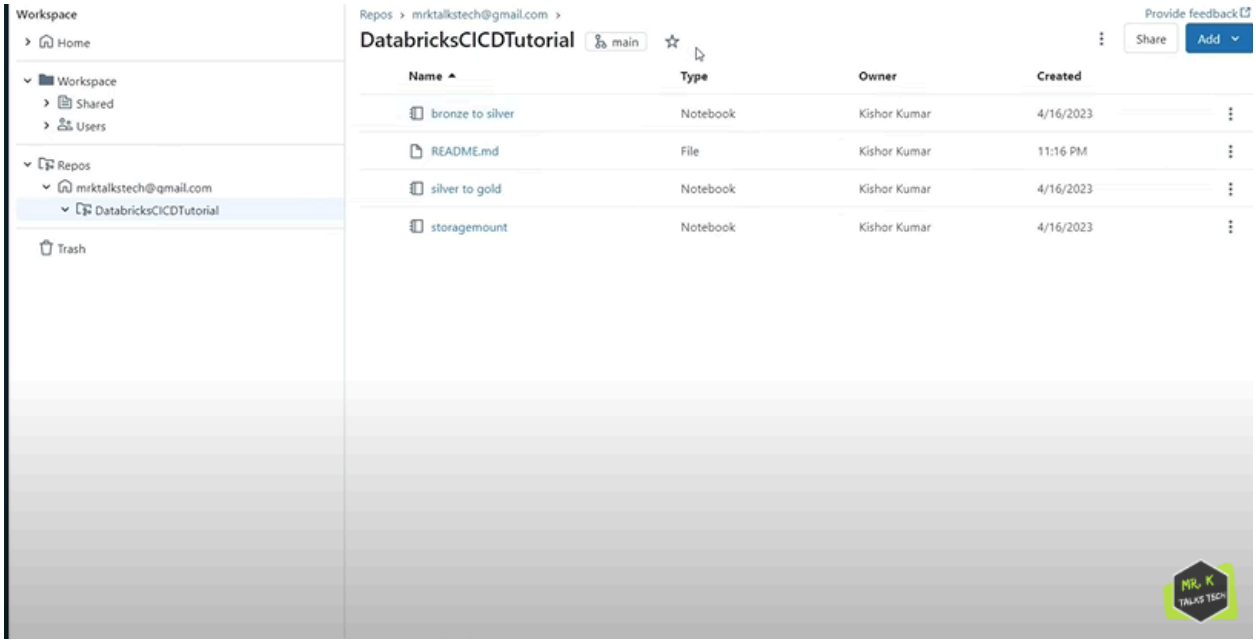


9. In the branch policies..we choose “required min number of reviewers”

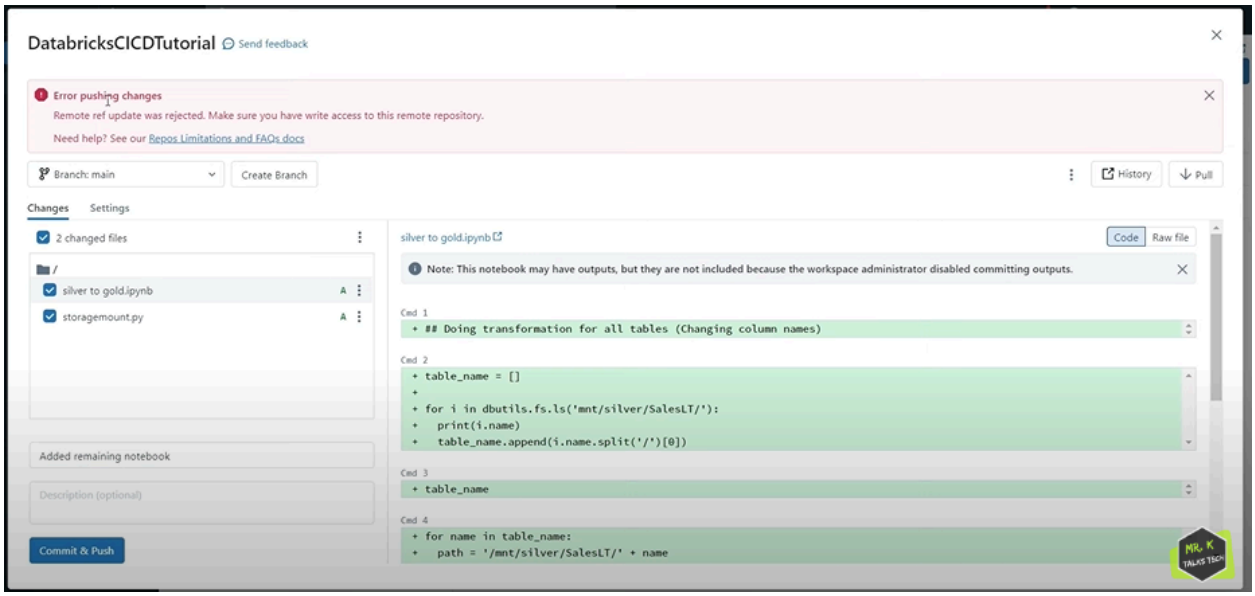


10. Lets understand this..”if a developer made changes in his branch and made a pull request to main branch for merging…then there must be 2 reviewers who has to approve this changes before ..merging with the main branch”
11. Now we have done protecting our main branch..lets test this now

12. Here we have moved the remaining two notebooks to the main branch

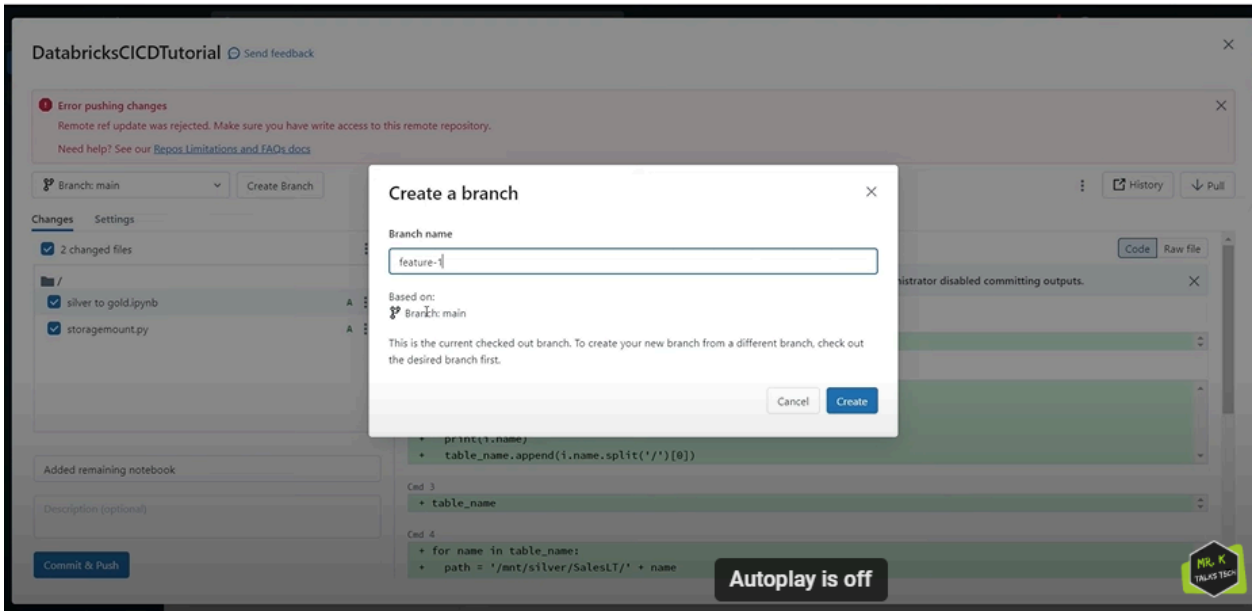


13. If we successfully protected our branch..then while committing to main branch ..we'l receive this error

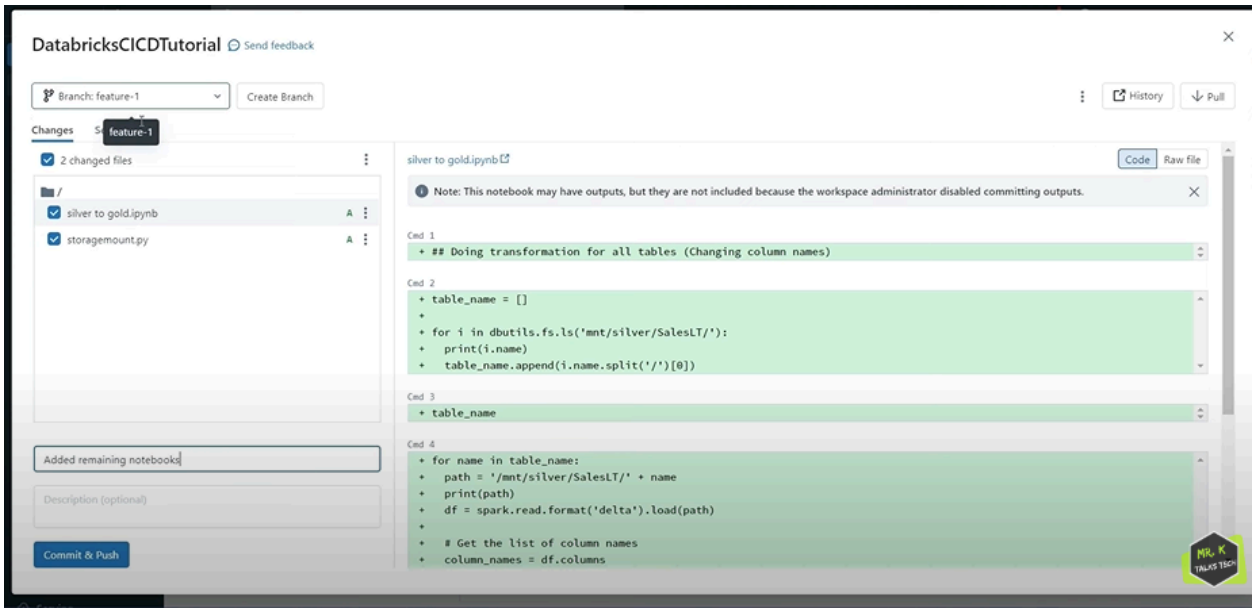


14. In order to commit to main branch..we have to use pull request and merge with main

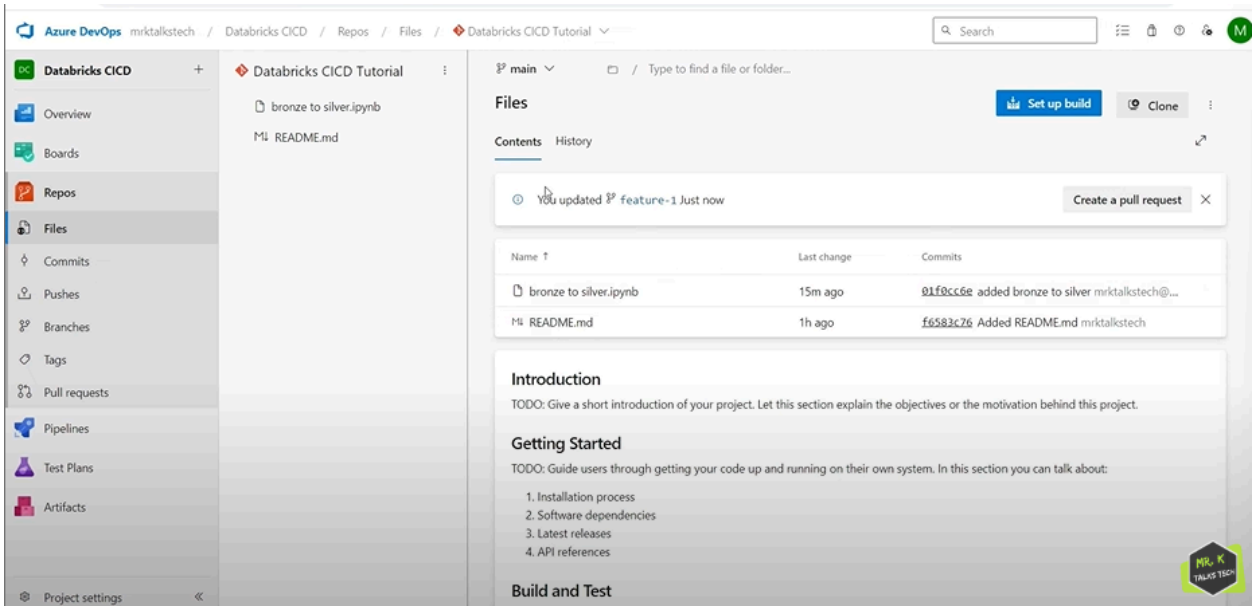
15. Now we will create a new branch from main



16. And commit changes to the feature-1 branch..and click on

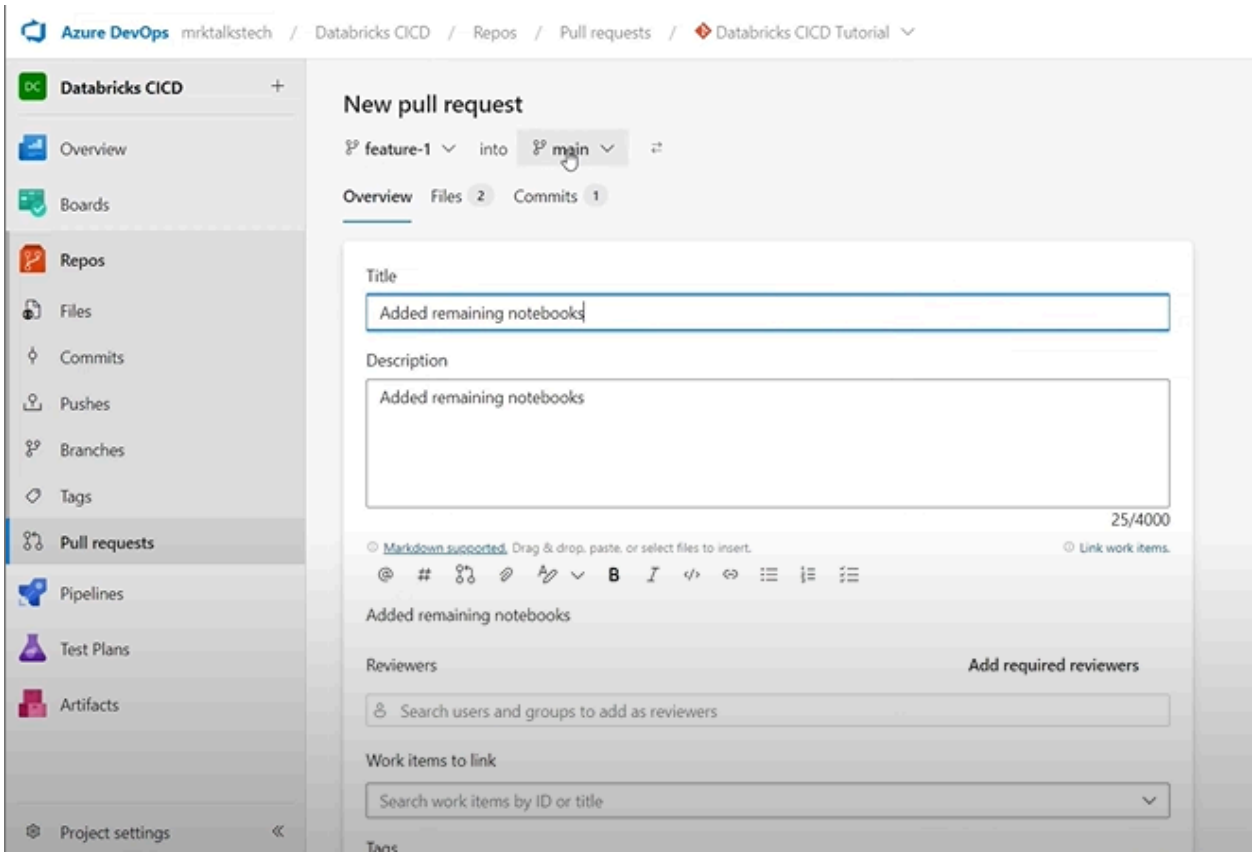


17. Now using pull request we'll merge this with main branch..for that..go to azure devops and click repos

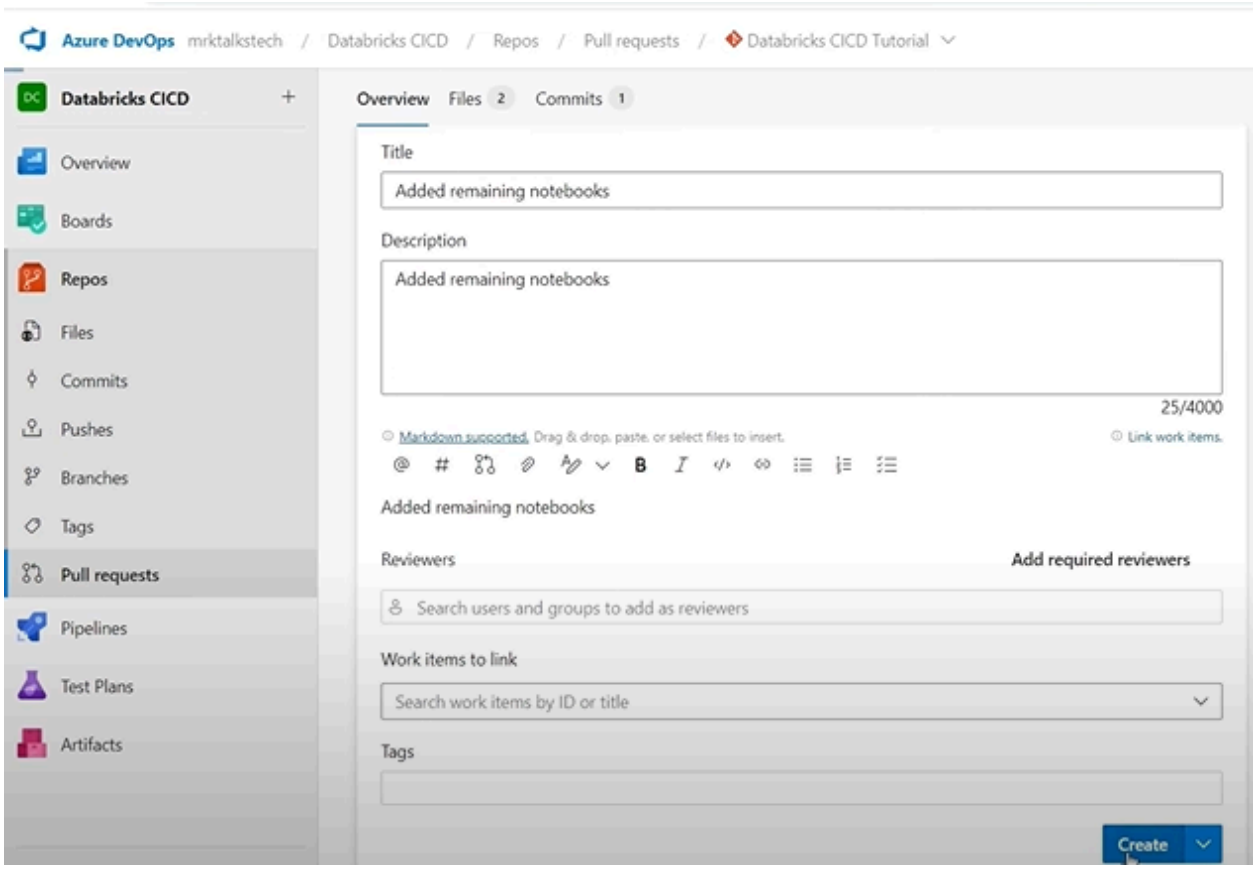


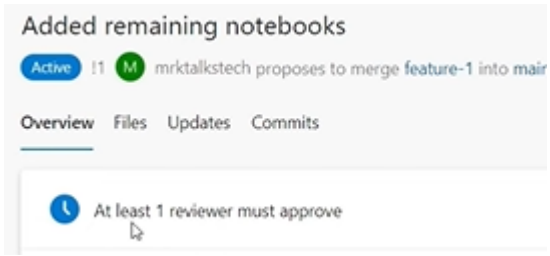
click on create pull request 

18. Now here the most imp thing to check source and target branches

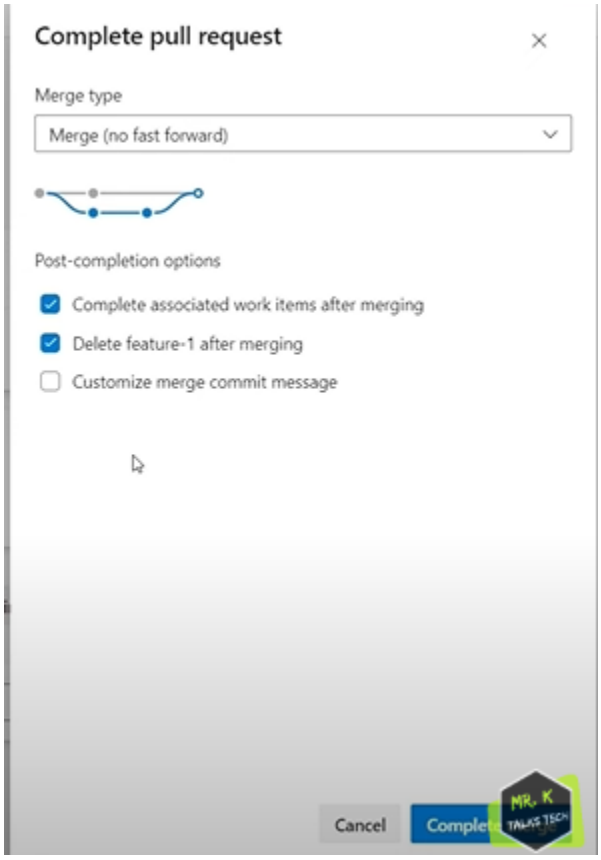


19. Created pull request

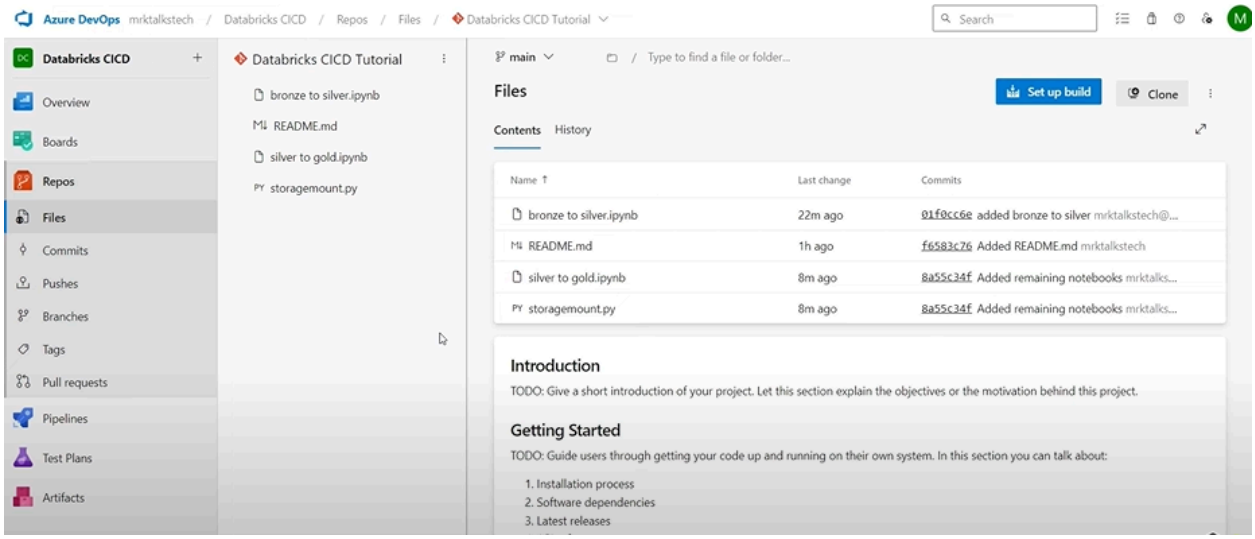


20. Now we can see at least  and we approve this request

21. It is always thw best practice ..to delete the feature1 branch after merginh

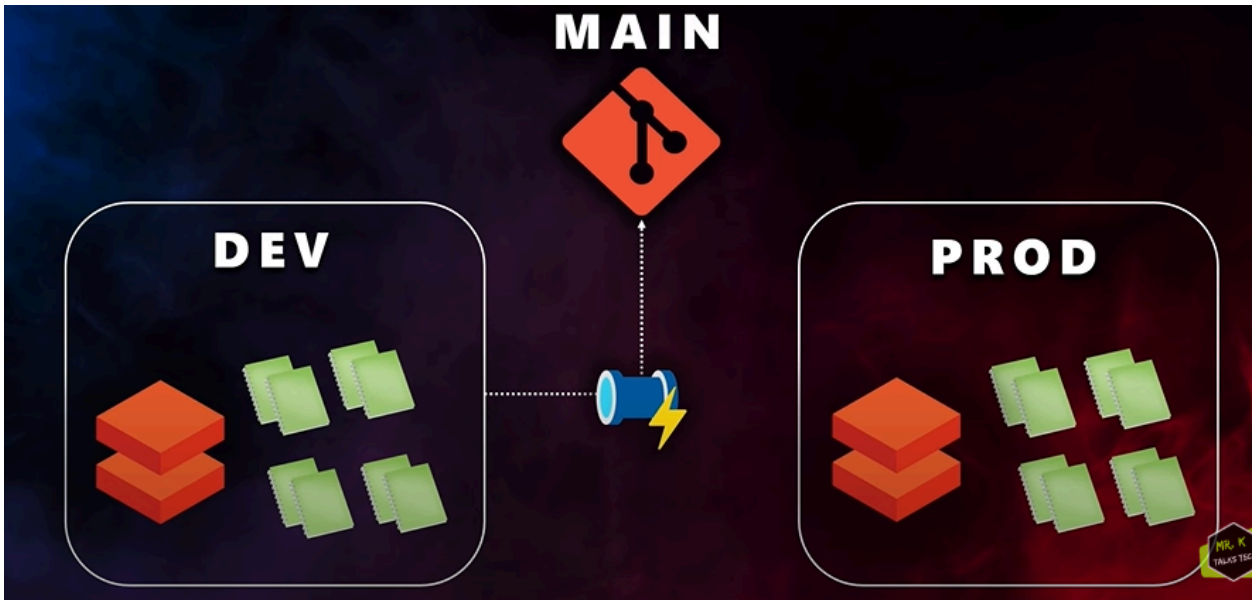


22. Successfully merged with main brancg

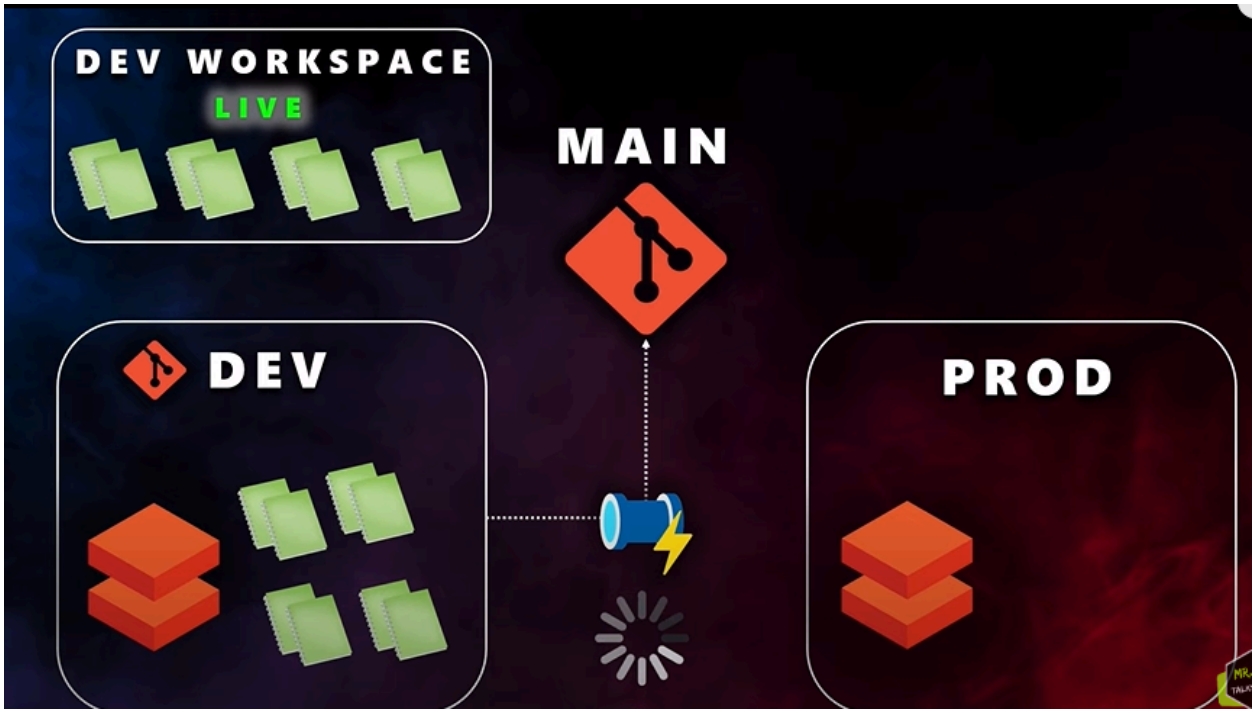


Creating Continuous Integration Pipeline

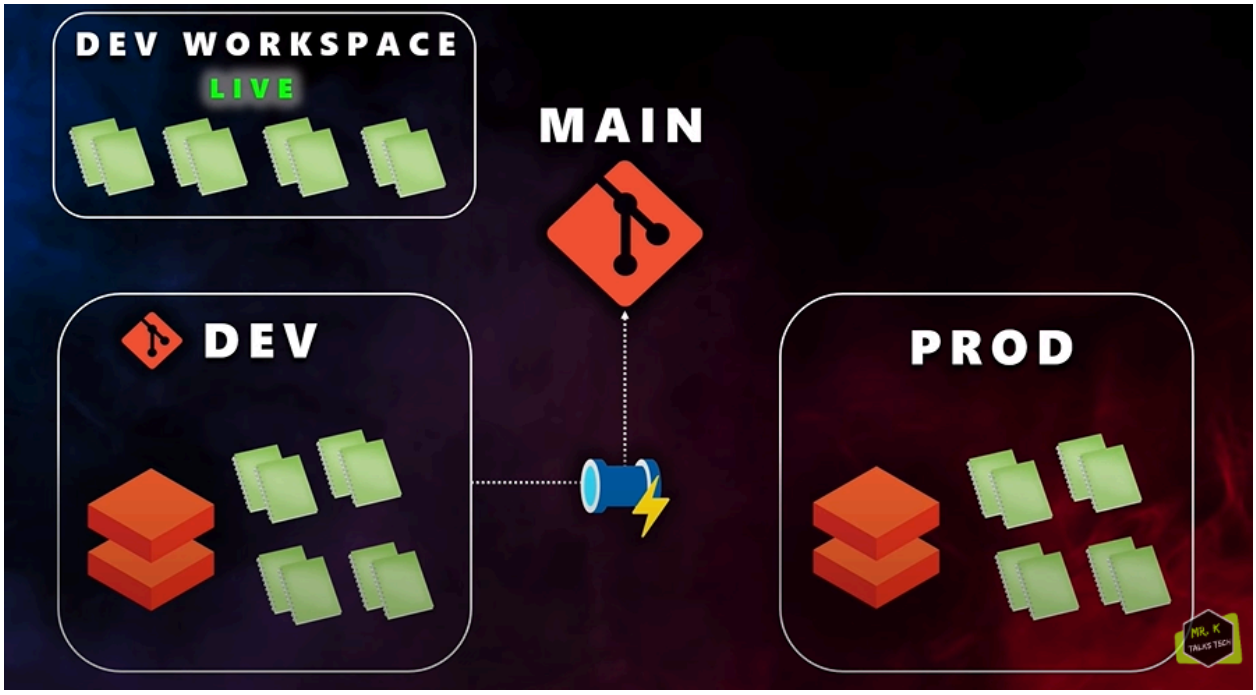
1. Here in our CICD pipeline…when ever we commit in the main branch..CICD will get triggered and commits the changes to prod env



2. Now while creating CI ..we need one more thing
3. While committing the changes to the prod env…what CI does it creates a live folder inside the dev workspace

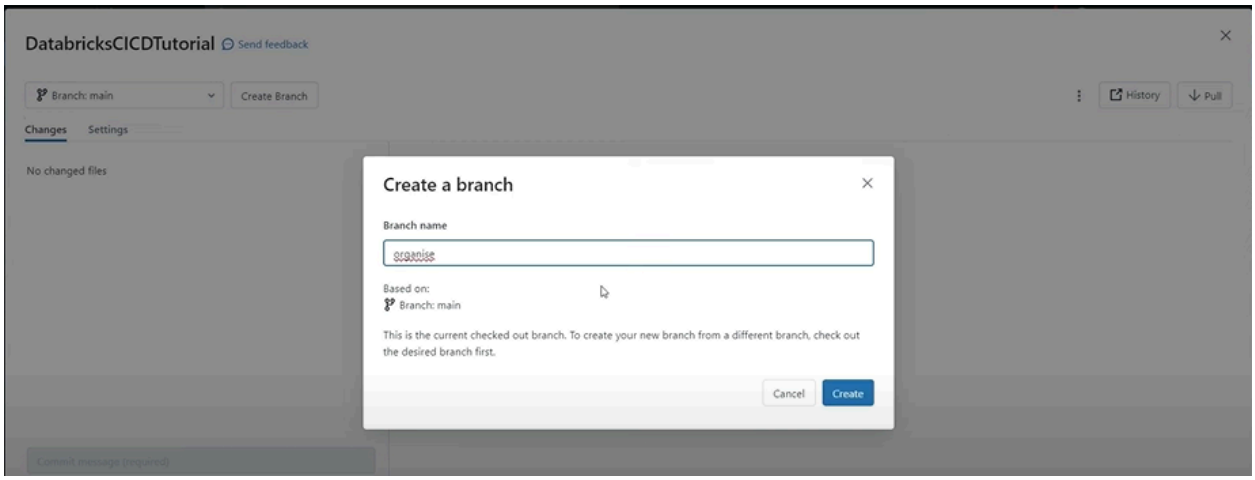


4. Lets see this practically
5. Actually whats the use of live Dev Workspace?..lets go to azure DF studio (refer vid)

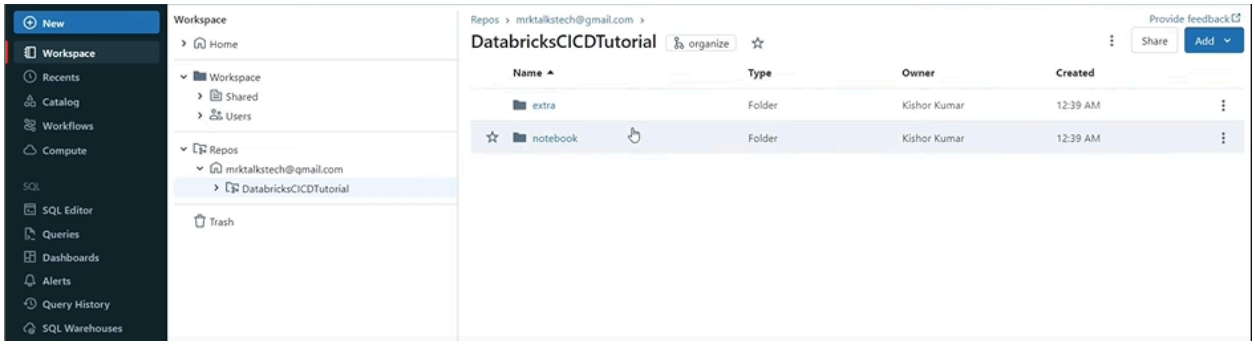
6. 

Organizing the folder structure for CICD

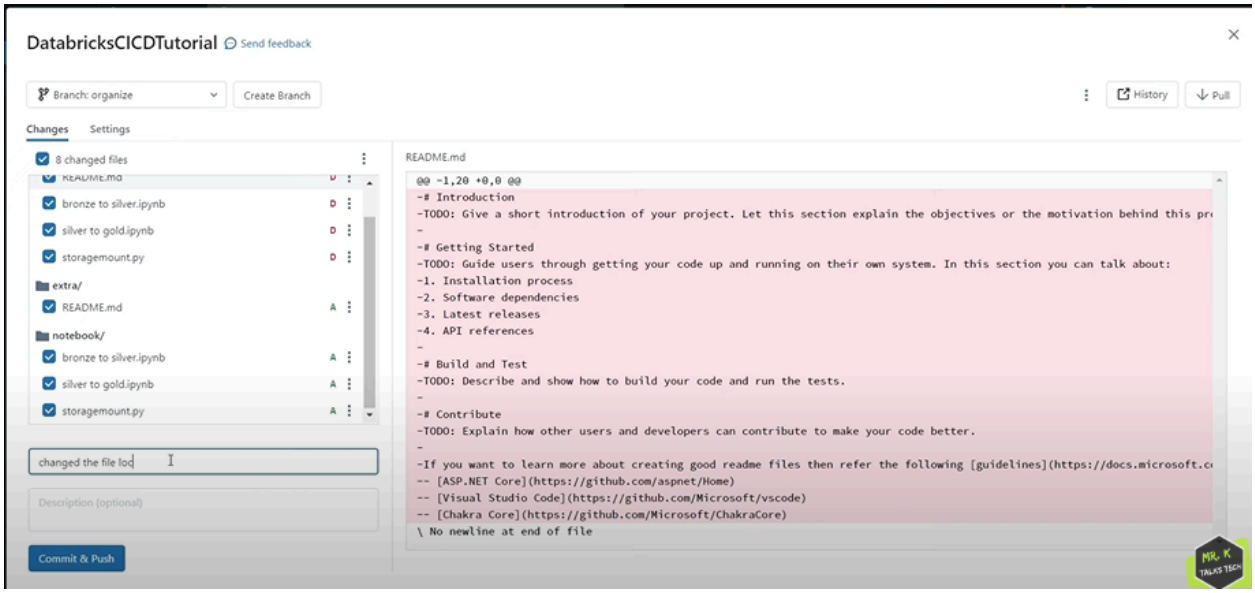
1. Lets create a new branch called organize ..



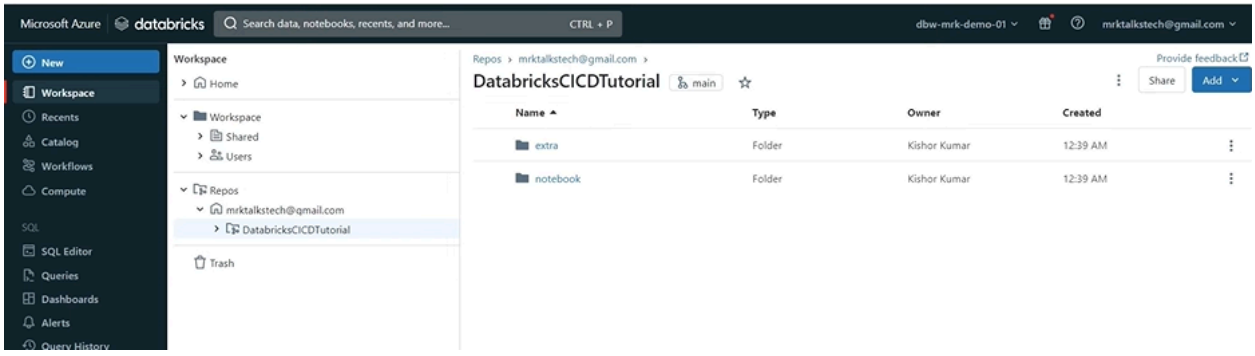
2. In this branch..we have created new folders for notebooks and extra files



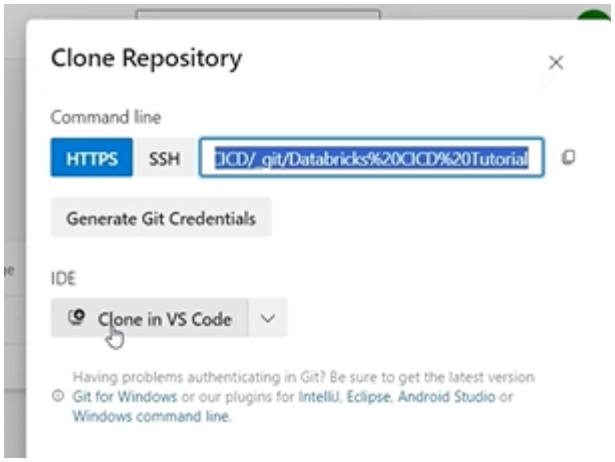
3. So we'll configure the CI pipeline in a way that only…Notebook folders get deployed in Prod env
4. Now we'll commit our changes in the Organize branch



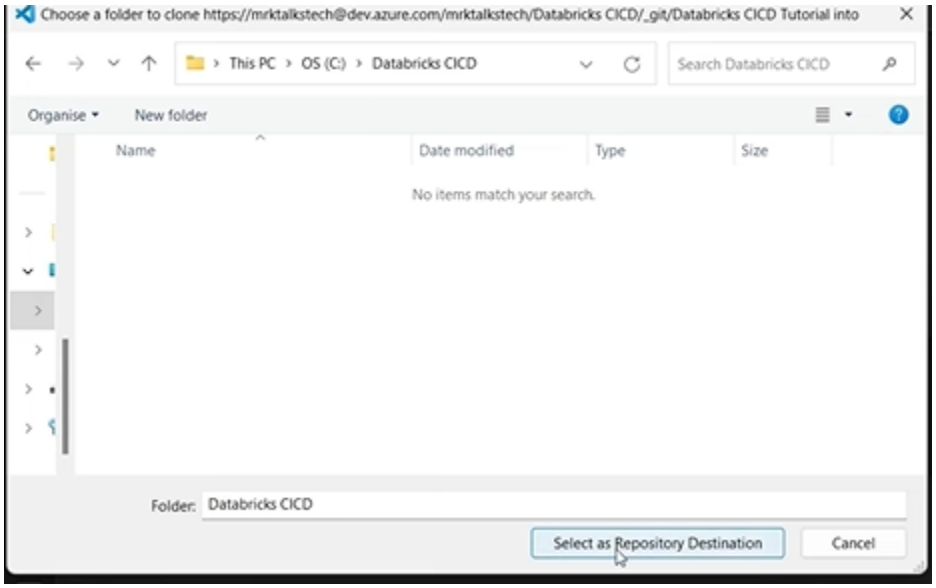
5. Now we'll create the pull request and approve it
6. We can see our main branch has folders now



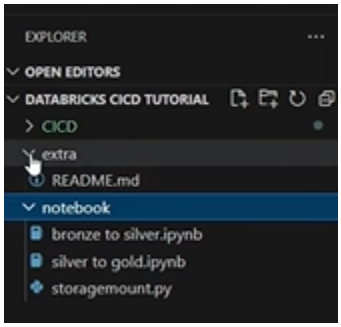
7. Now we have to code in YAML for configuration CI
8. We'll use VScode to write the YAML code..for that we need to clone this repo in our local setup

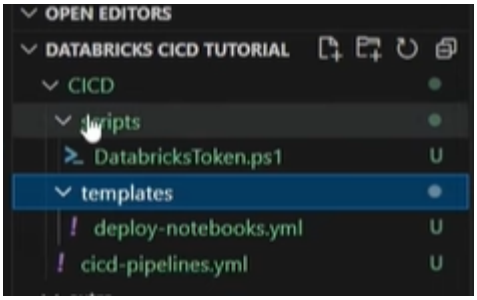


9. Click on Clone and click VScode
10. Now we'll select the dest and clone

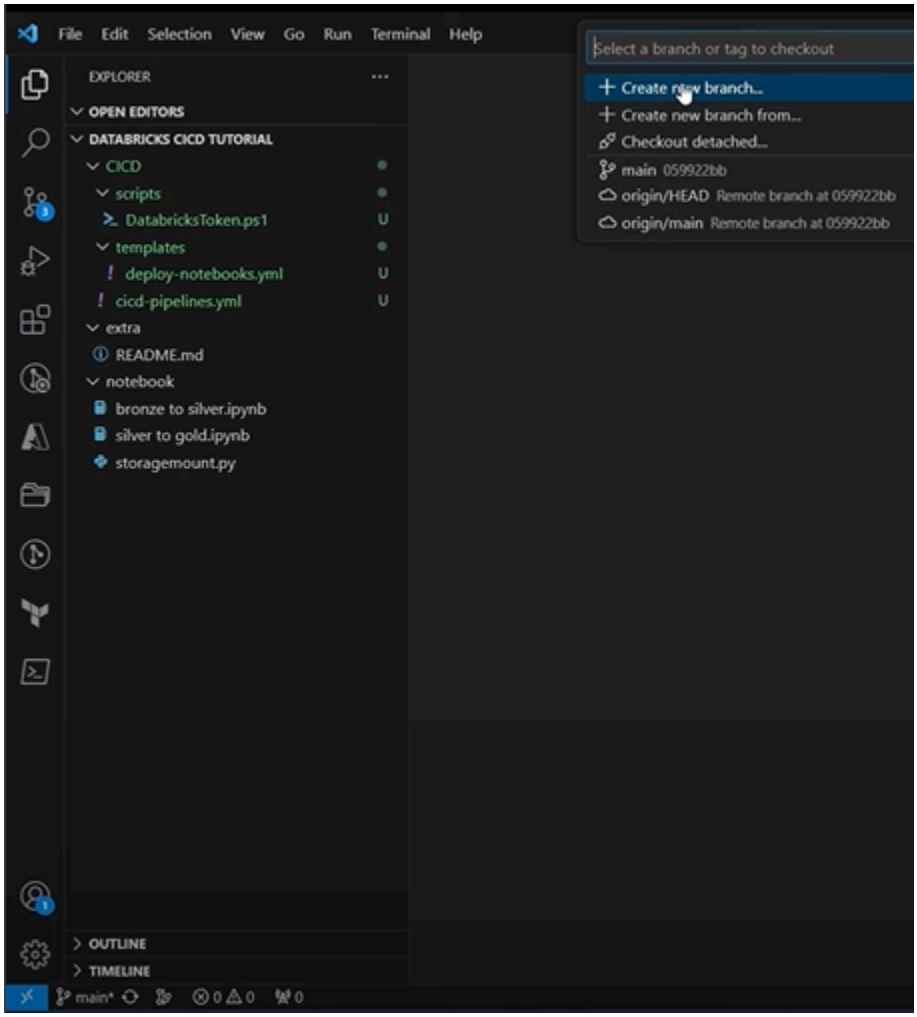


11. Now we will import the CICD folder which has YAML code to our local repo

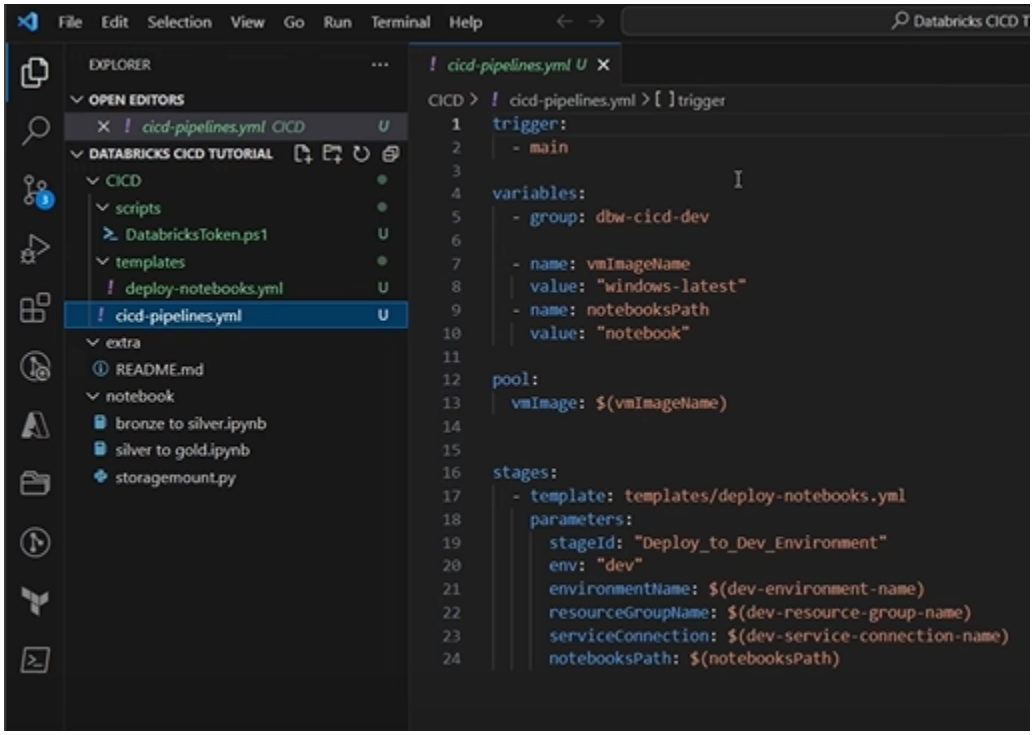


12. Here 

deploy-notebooks.yml is the one which we use to implement CI
13. Now here we will create separate branch in vscode

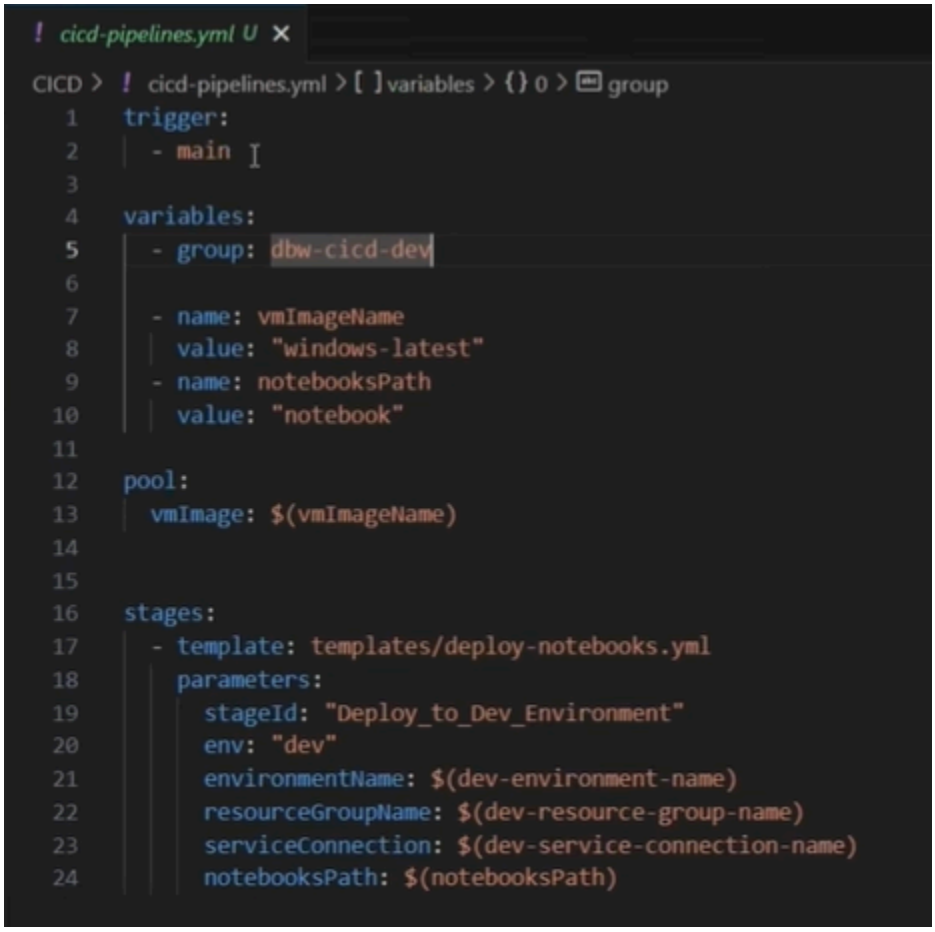


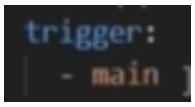
14. Here CICD pipeline.yml is the master file



15. We'll be usng this master file..for creating CICD pipeline in azure DEvops
16. This master file will call deploy-notebooks.yml…and this file will call the script file

17. Lets understand master file



18. Here the most imp is first 2 lines… 

..here main is the value of trigger…which means this YAML gets triggered..if there are any changes in the main branch
19. Next thing we have is variable groups
20. Main use of variable groups

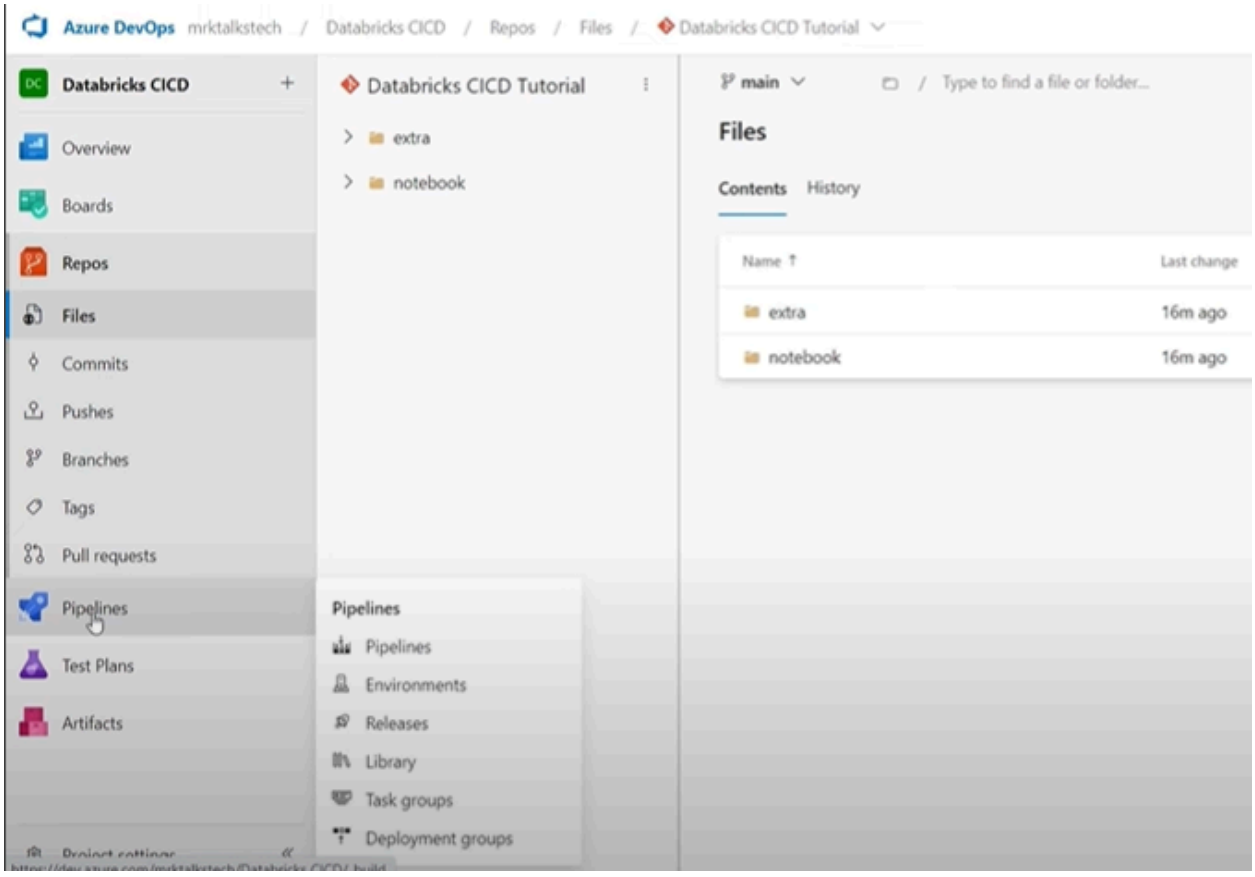
Imagine you have separate pipelines for building and deploying your application to development, staging, and production environments. Each environment requires different database connection strings.

By using variable groups:

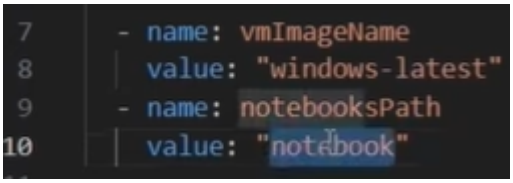
1. Store the database connection strings for each environment in separate variable groups (dev-db-connection, staging-db-connection, prod-db-connection).
2. Reference the appropriate variable group in each pipeline based on the target environment.

This approach keeps your pipelines clean, secure, and easier to maintain as your environment configurations evolve.

21. Lets create a group…go to pipelines in azure devops



22. Go to  for creating variable groups



23. Here we hav ..it is just like key-value pair in dict
24. As we are only concentrating on notebooks..we used a variable just for notebook

25. Here vmImage contains the value : :windows-latest”..lets understand it…

The `pool` section in a CI/CD YAML code defines the execution environment for your pipeline jobs. It specifies the resources, like virtual machines (VMs) or containers, used to run the tasks within your pipeline.

Here's a breakdown of what the `pool` section typically includes:

- `vmImage`: This property defines the virtual machine image used by the agent that executes the pipeline jobs. The image determines the operating system (Windows, Linux, etc.) and software pre-installed on the agent.
- `demands`: (Optional) This property specifies requirements for the agent that runs the jobs. You can define constraints based on factors like available memory, CPU cores, or specific software installations.

Imagine you have a CI/CD pipeline that builds and tests a Python application. Here's an example `pool` section in your YAML code:

YAML

```yaml
pool:
  vmImage: ubuntu-latest  # Use an Ubuntu image
  demands:
    - agent.jobName == Build  # Only use this pool for jobs named "Build"
    - RAM >= 4GB              # Require at least 4GB of memory
```

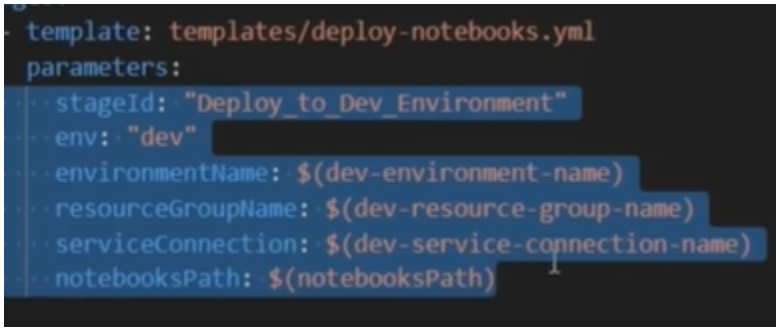
Use code with caution.

Explanation:

- This `pool` section defines an execution environment using the latest Ubuntu virtual machine image.
- The `demands` section specifies two constraints:
  - The first constraint ensures this pool is only used for jobs named "Build" (potentially differentiating it from other jobs like deployment).
  - The second constraint requires that the agent running the job has at least 4GB of RAM to handle the build process.

26. Coming to the stage..

27. Here we call our template along with this parameters

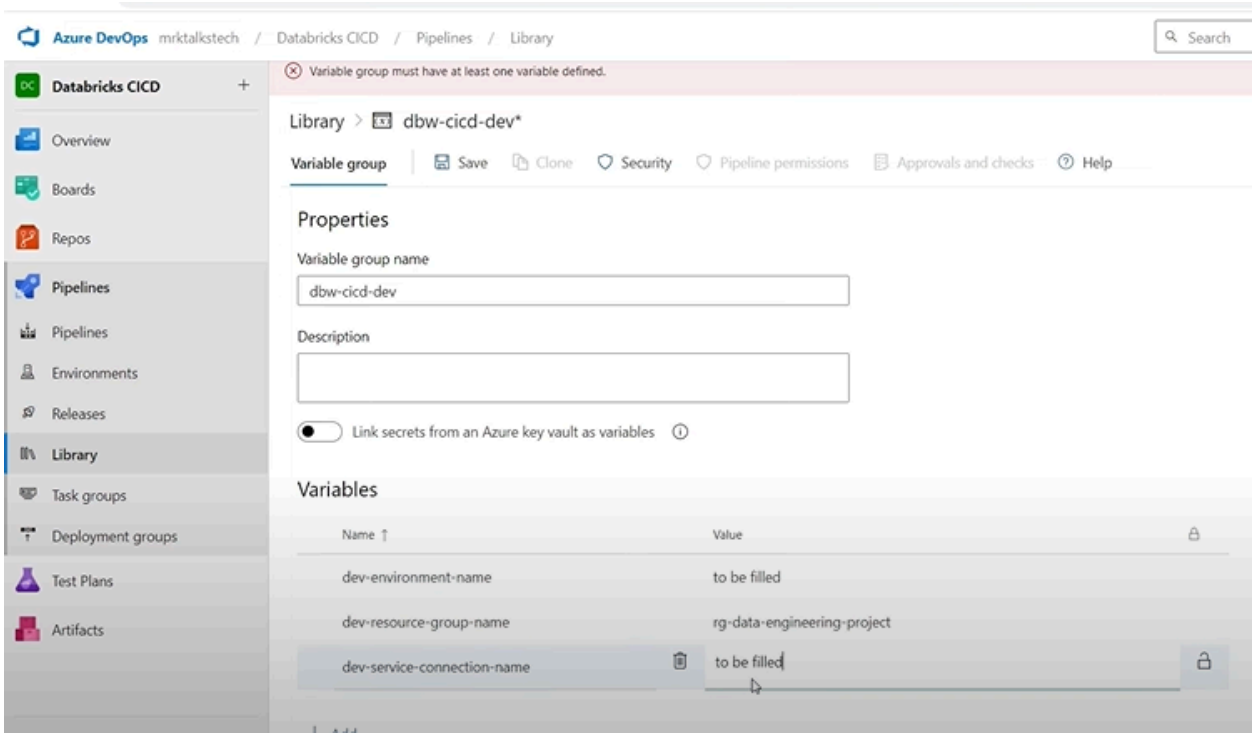


```yaml
template: templates/deploy-notebooks.yml
parameters:
  stageId: "Deploy_to_Dev_Environment"
  env: "dev"
  environmentName: $(dev-environment-name)
  resourceGroupName: $(dev-resource-group-name)
  serviceConnection: $(dev-service-connection-name)
  notebooksPath: $(notebooksPath)
```

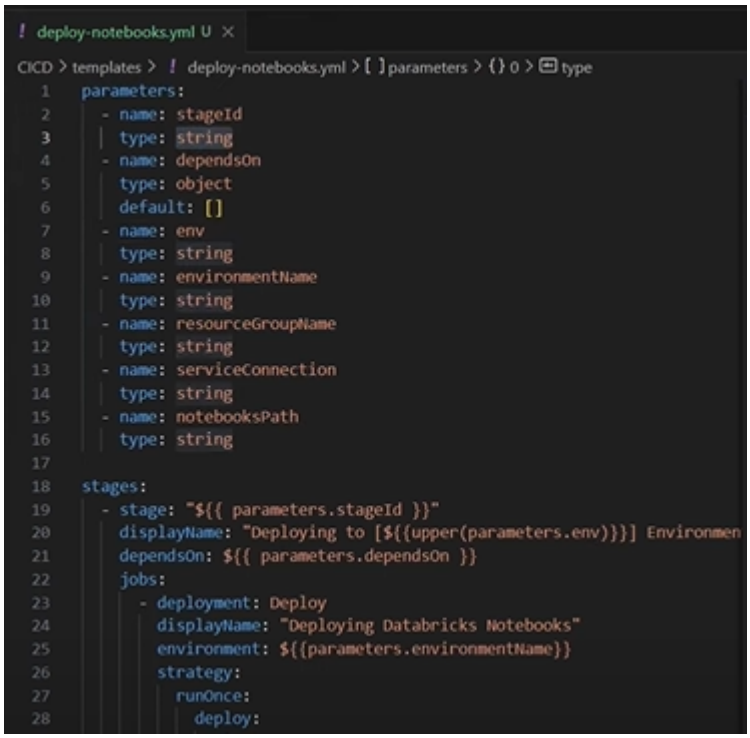
- **Template Reference:** The stage references an external YAML template named `templates/deploy-notebooks.yml`. This template likely contains the specific steps to deploy the notebooks.
- **Parameters Passed:** Several parameters are passed to the template when it's included in the workflow. These parameters provide configuration details for the deployment process:
  - `env`: This sets the value of an environment variable named `env` to `"dev"`. This might be used to differentiate between development, staging, or production environments.
  - `environmentName`: This likely references a variable named `dev-environment-name` that must be defined elsewhere (not shown in this snippet). It specifies the name of the Azure environment where the notebooks are deployed.
  - `resourceGroupName`: This likely references a variable named `dev-resource-group-name` that must be defined elsewhere (not shown in this snippet). It specifies the resource group where the Azure environment resides.
  - `serviceConnection`: This likely references a variable named `dev-service-connection-name` that must be defined elsewhere (not shown in this snippet). It might be a service connection used to authenticate with Azure resources.
  - `notebooksPath`: This references the `notebooksPath` variable defined earlier. This tells the template where to find the notebooks relative to the repository root.

28. Here we have not specified the these parameters directly

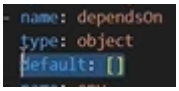
29. We'll define these variable with values in Azure devops



30. Now lets look at the template…here all the parameters of the dev stage…will be passed to this template

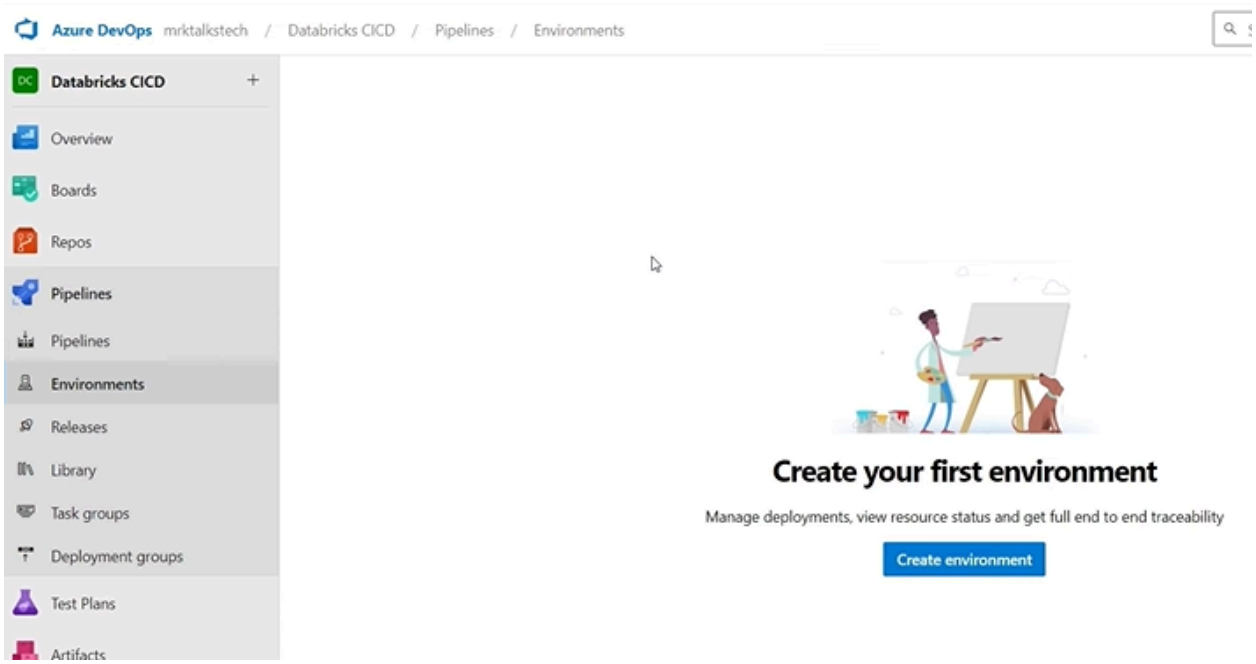


31. This parameter is defined in this template with default value as None



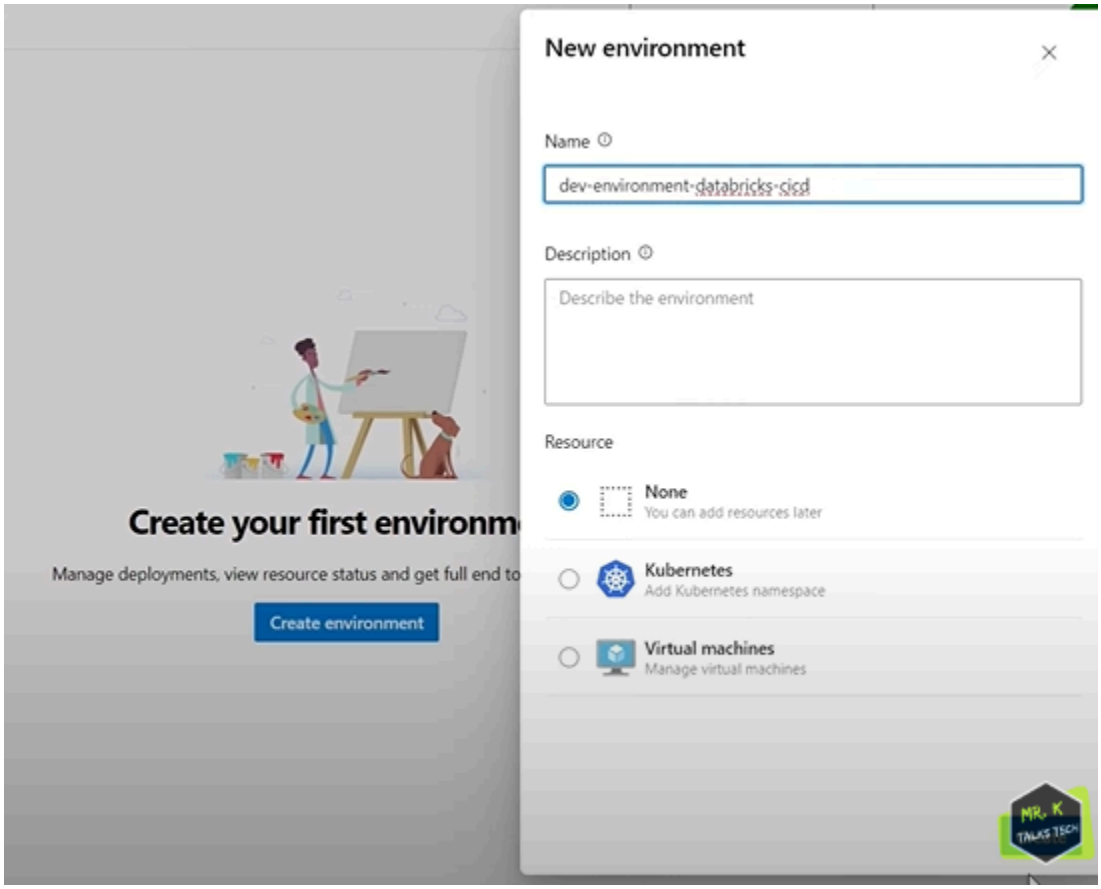
32. Once it gets the value of all the parameters..it proceed further to staging

Environments and Service connection

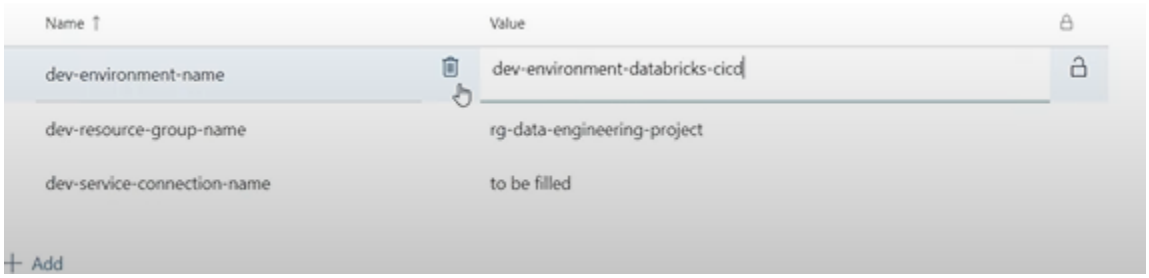
1. 

2. For example..if we created an env for dev then CICD pipeline runs and deploy the changes to the env..then we can come here to check the deployment status

3. So we create one env for our dev

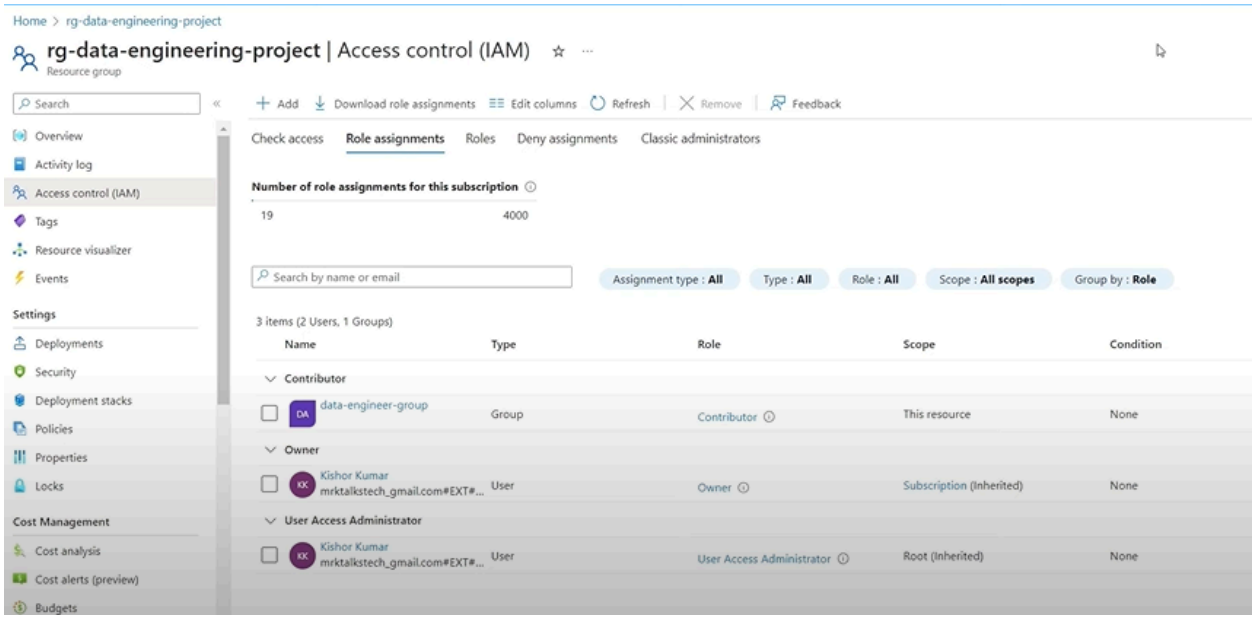


4. Now we will fill this  in our dev-env name variable



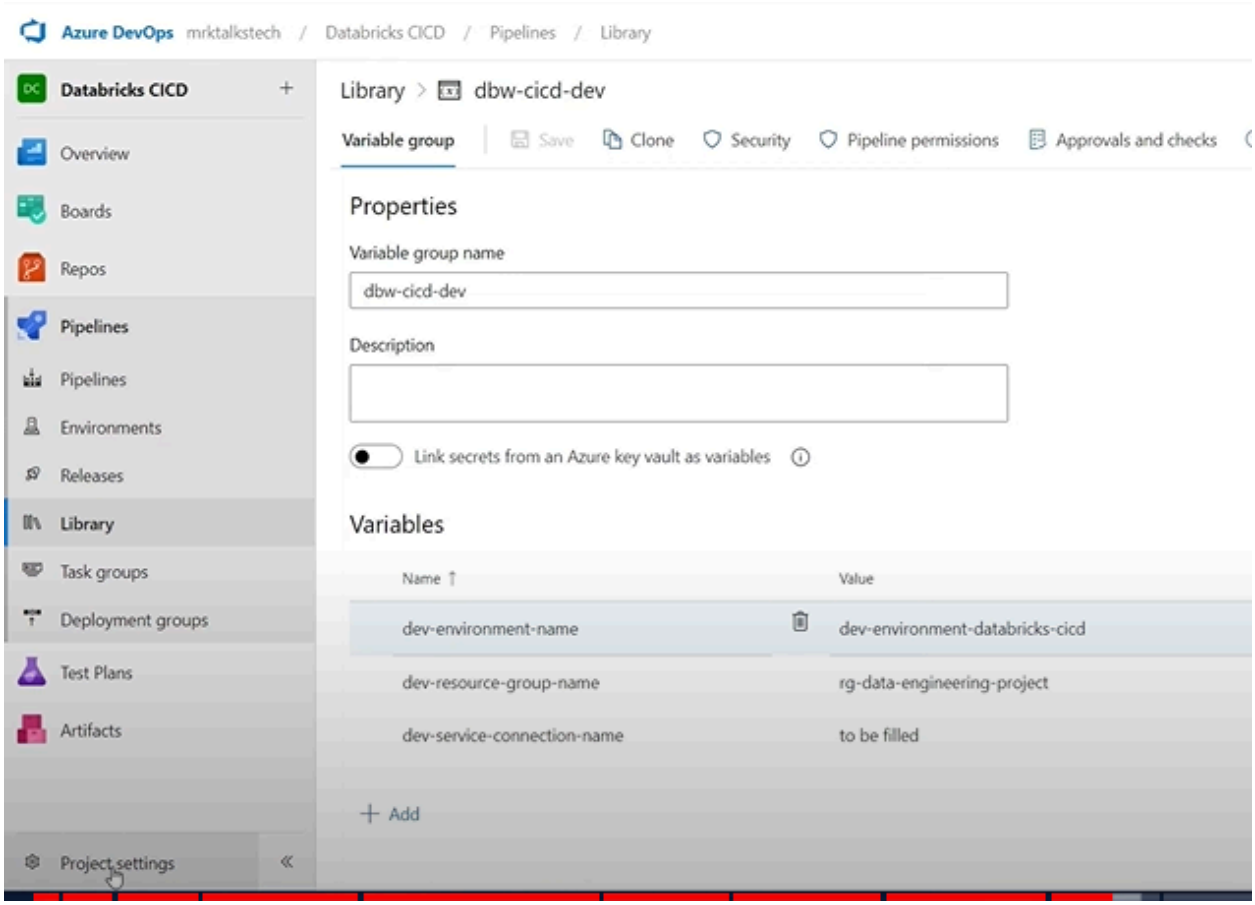
5. Lets discuss service connection `serviceConnection: $(dev-service-connection-r`
6. So we are creating CICD pipeline..and this needs some azure resources to use …so we need some kind of connection bw azure devops and azure resources..for this we'll use service connection
7. Before creating a service group… we'll go to our resource group access control

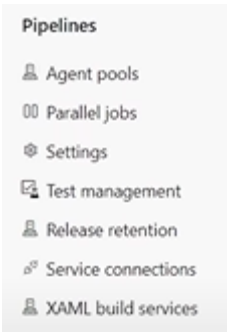
8. AT present only they have access to our res grp



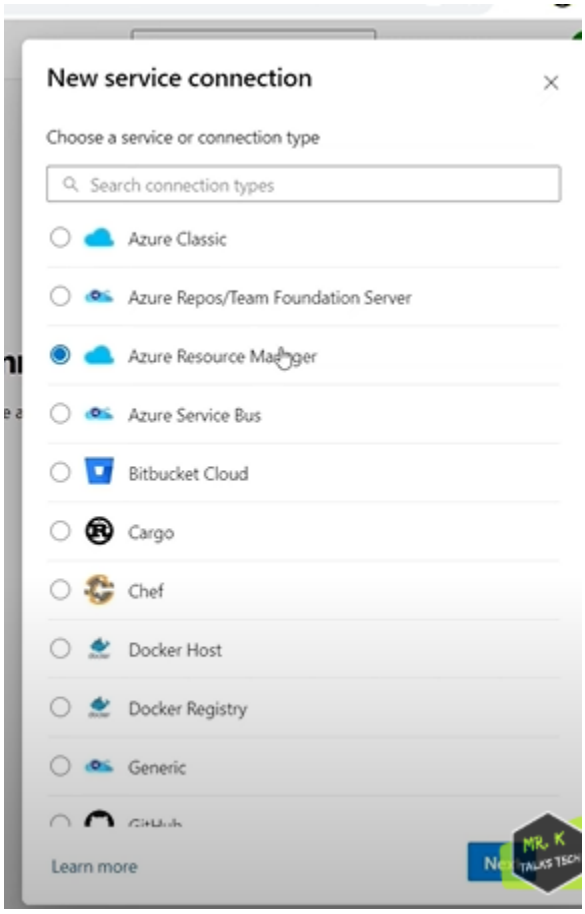
we cannot add azure devops directly here..so thats y we use service connection

9. For that Now go to project settings in the left bottom

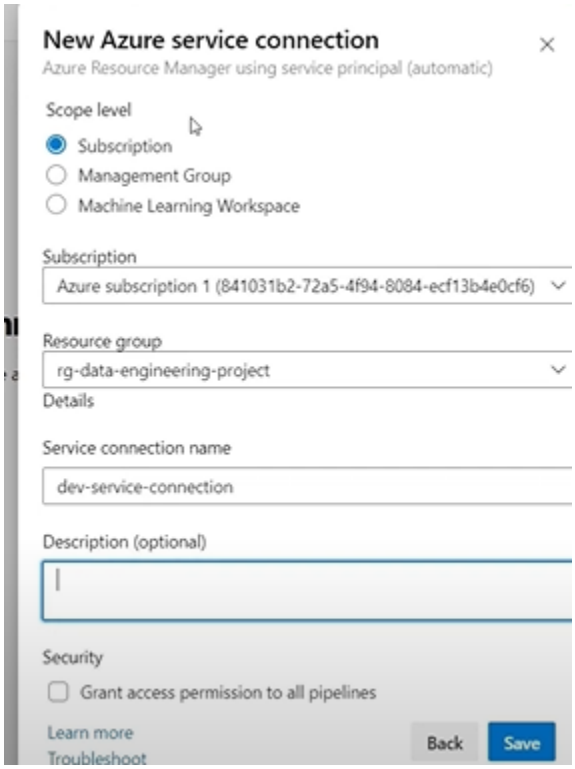




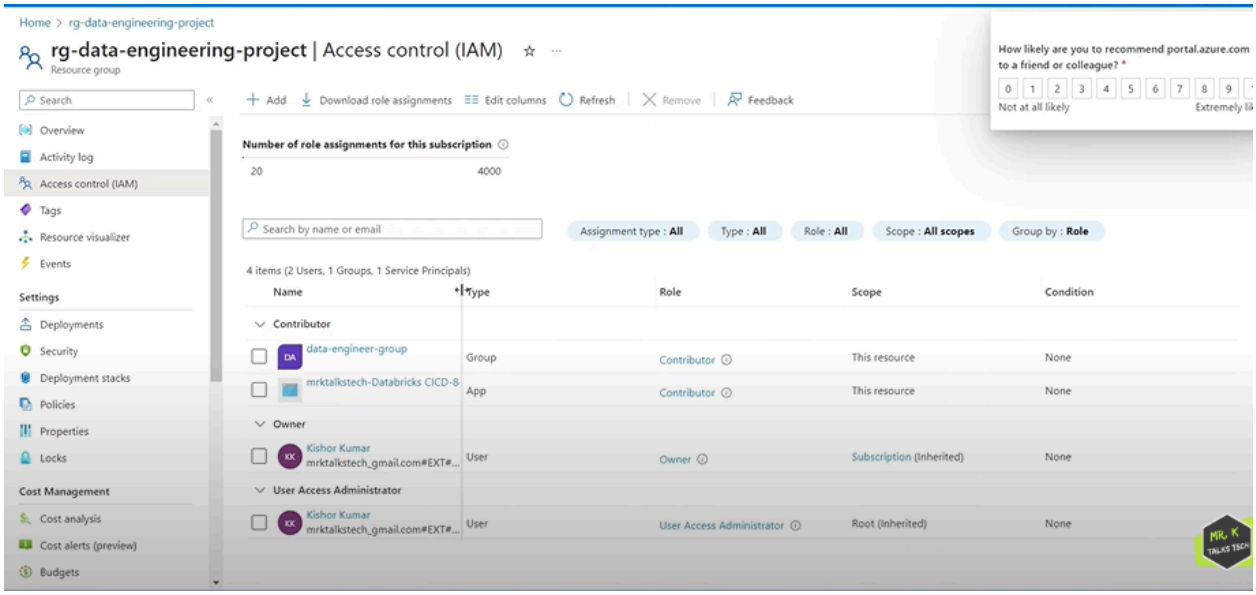
10. If we scroll down on left menu..we can see service coneciton



11. We'll use Service principle as authentication



12. Here the new service principle has created which has contributor access



13. Next we fill the parameter value

14. Refer video for deploying notebook functionality
15. Later we'll commit our changes in the branch that we have created in the vscode
16. And we'll merge it with main branch
17.