

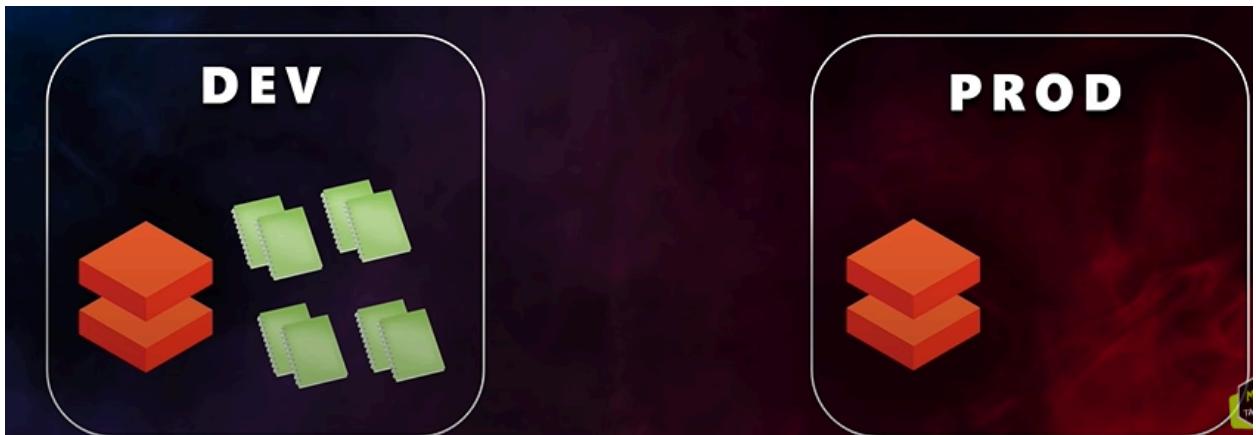
## What is CI/CD?

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

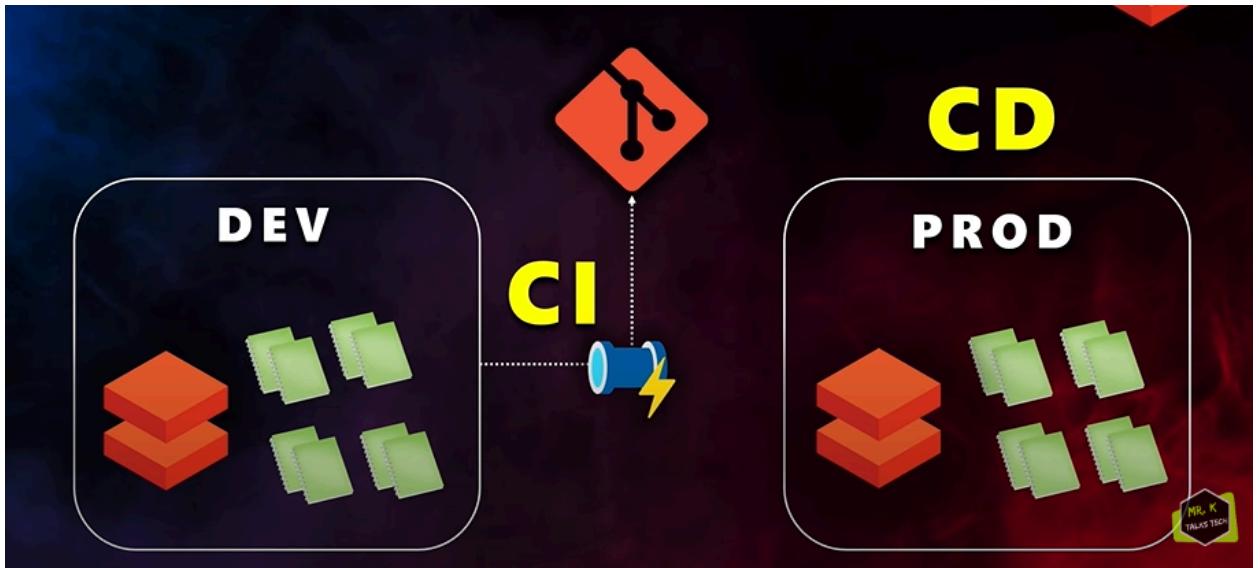


some notebooks in it.

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



# What is CI/CD ?

**- It is a set of practices used to automate and streamline the process of building, testing, and deploying code changes to different Environments**

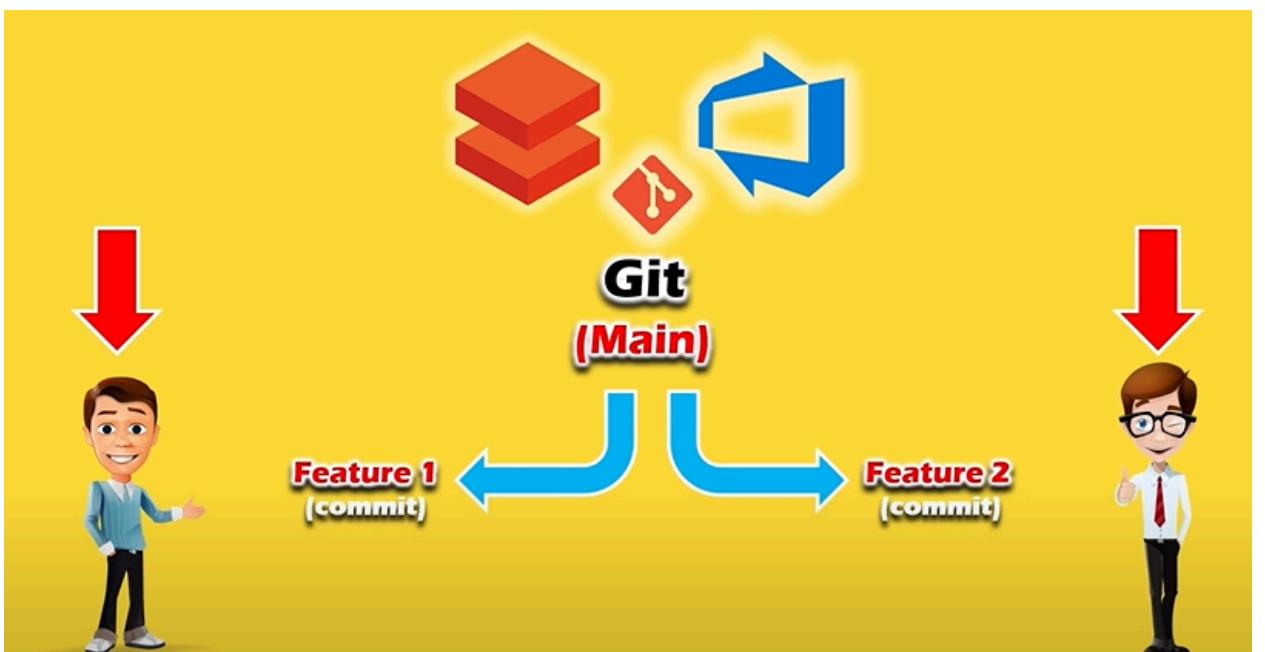
## Merging/Branching Techniques

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

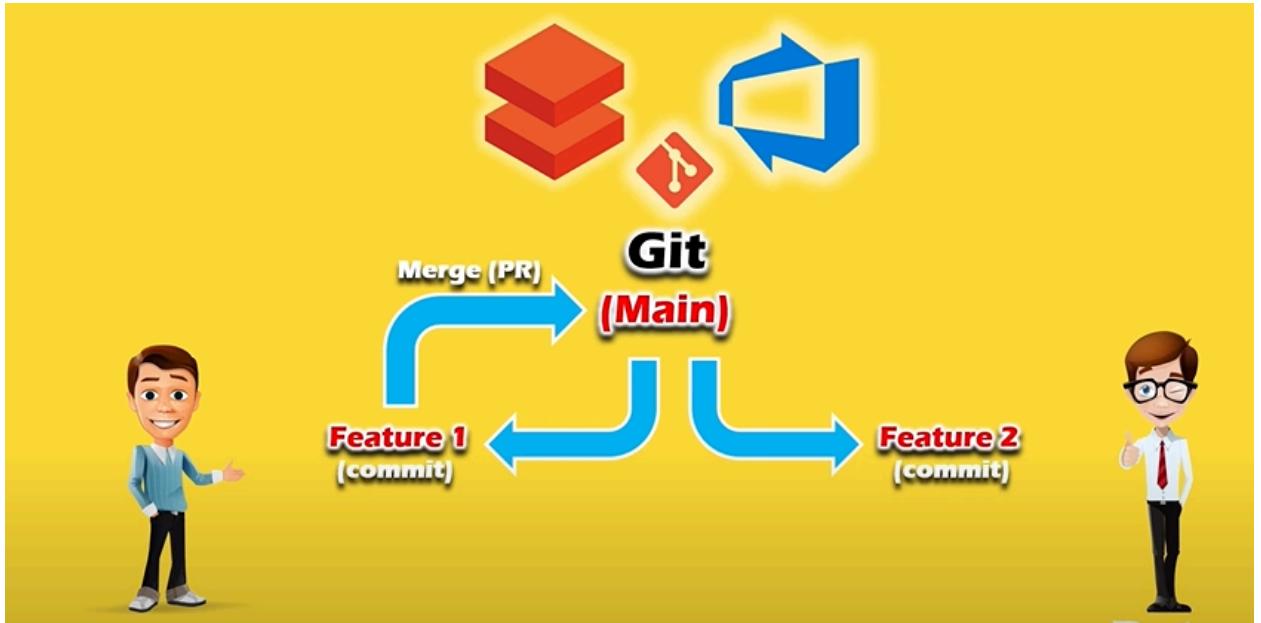
3. Important thing is to integrate azure devops repo to our dev env



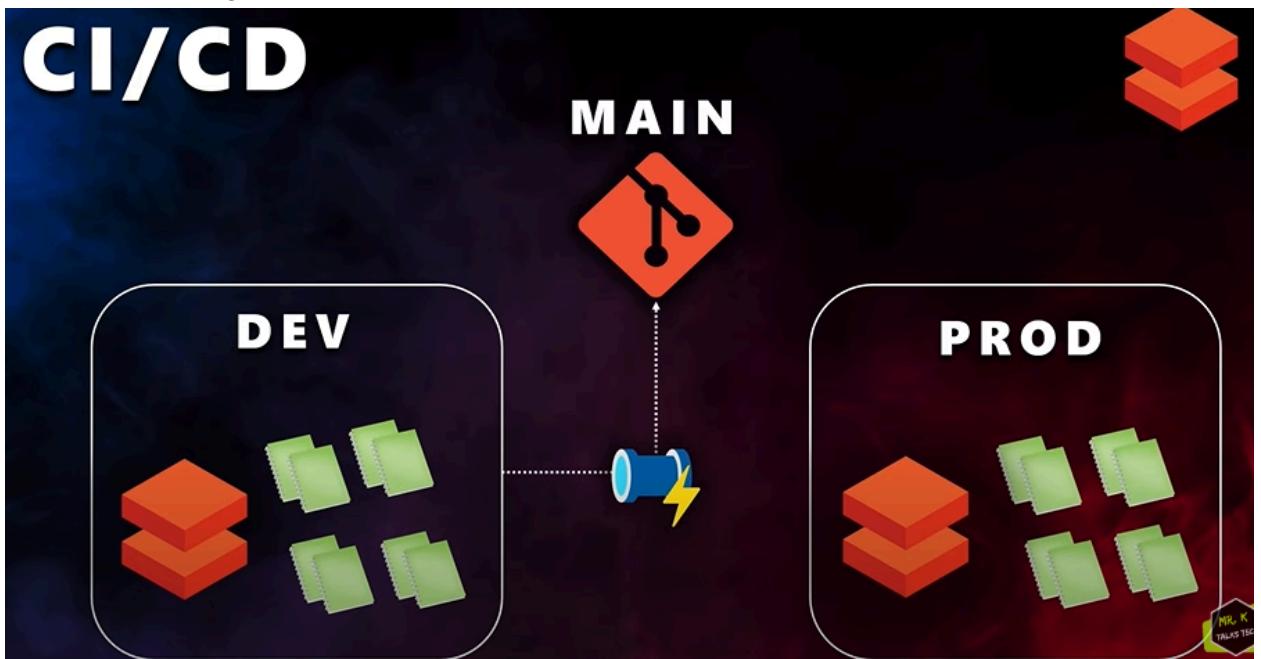
4. And while integrating ..we need to create a main branch in this repo ..this main branch is very imp..like whenever we commit anything in the main ..then CICD pipeline gets triggered
5. So we'll protect this main branch...no data engineers ..can directly commit to the main branch..and if incase they 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



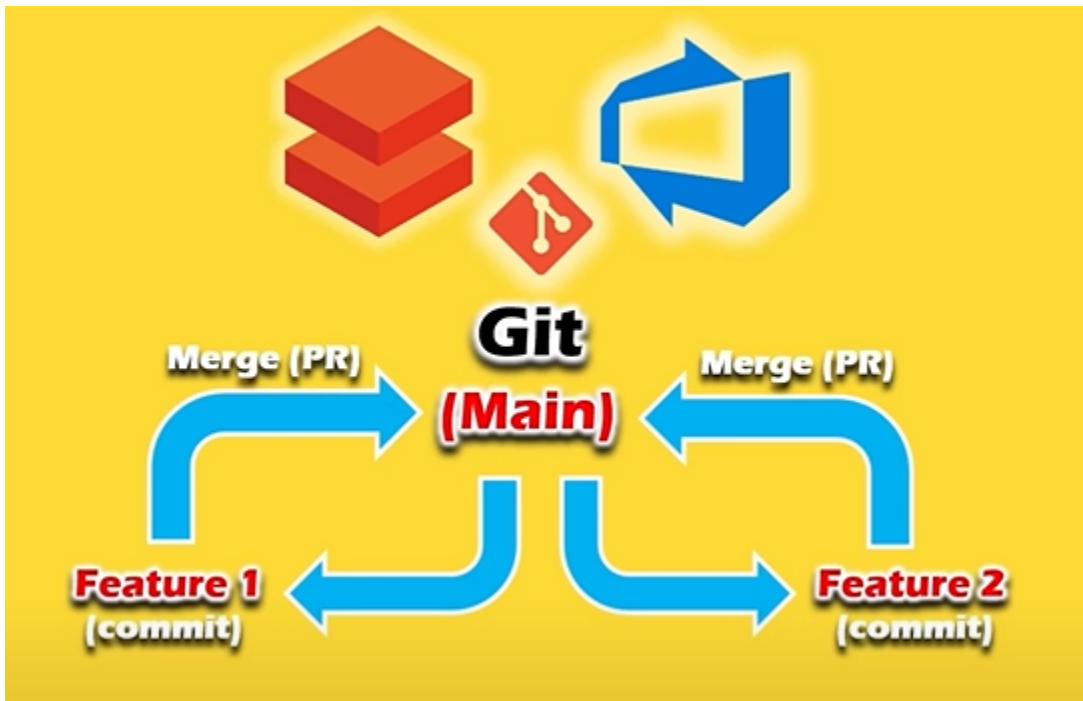
7. Now the question may arises..like which branch should be merged with main branch..then the team of data engineer's will discuss among themselves about the priorities and decide which feature to merge
8. Let's Suppose we have decided to go ahead with feature 1
9. Now using the pull request ..we will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

Screenshot of the Microsoft Azure portal showing the "rg-data-engineering" resource group. The left sidebar shows navigation options like Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Deployments, Security, Deployment stacks, Policies, Properties, Locks, Cost Management, Cost analysis, Cost alerts (preview), and Budgets. The main pane displays the "Essentials" section with a "Resources" tab selected. It lists five resources: adf-mrk-demo-01 (Data factory V2), dbw-mrk-demo-01 (Azure Databricks Service), kv-mrk-demo-001 (Key vault), mrkdatalakegen2 (Storage account), and synw-mrk-demo-01 (Synapse workspace). The resources are listed in a table with columns for Name, Type, Location, and three-dot ellipsis menu icons.

2. This are the notebooks we created for our project

The screenshot shows the Microsoft Azure Databricks interface. On the left, there's a sidebar with various navigation options like New, Workspace, Recents, Data, Workflows, Compute, SQL, and Machine Learning. The main area is titled 'Shared' and lists three notebooks:

Name	Type	Owner	Created
bronze to silver	Notebook	Kishor Kumar	4/16/2023
silver to gold	Notebook	Kishor Kumar	4/16/2023
storagemount	Notebook	Kishor Kumar	4/16/2023

3. We have another resource group for prod env...

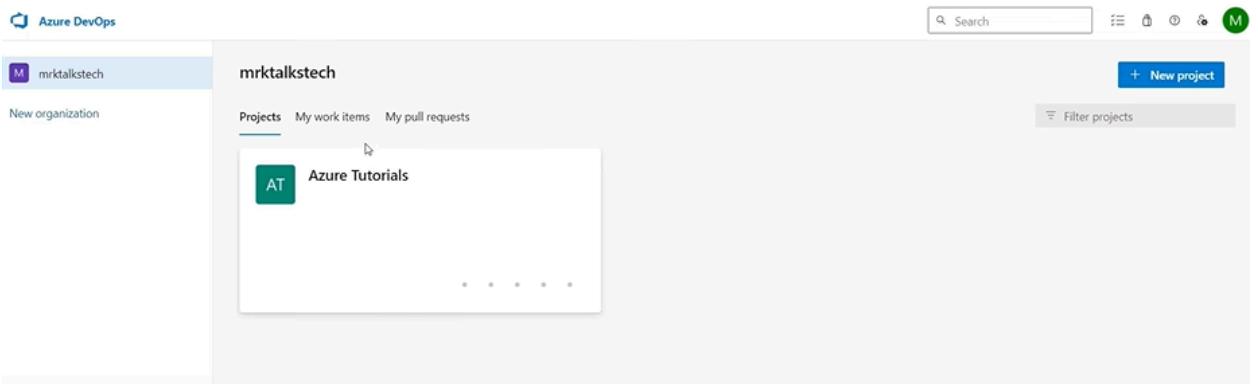
The screenshot shows the Microsoft Azure Resource Group overview for 'rg-data-engineering-project-prod'. The left sidebar includes options like Overview, Activity log, Access control (IAM), Tags, Resource visualizer, Events, Settings, Deployments, Security, Deployment stacks, Policies, Properties, Locks, Cost Management, Cost analysis, Cost alerts (preview), and Budgets. The main pane displays a list of resources with columns for Name, Type, and Location.

Name	Type	Location
adf-mrk-demo-01-prod	Data factory (V2)	Australia East
dbw-mrk-demo-01-prod	Azure Databricks Service	Australia East
kv-mrk-demo-001-prod	Key vault	Australia East
mrkdatalakegen2prod	Storage account	Australia East
synw-mrk-demo-01-prod	Synapse workspace	Australia East

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

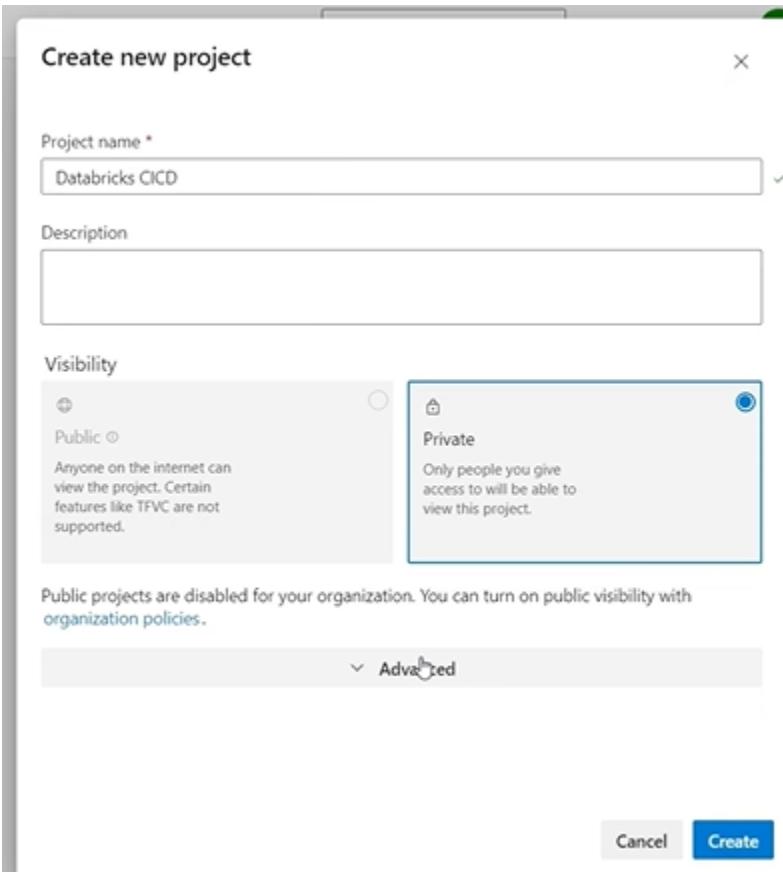
The screenshot shows the Microsoft Azure Databricks interface again. The sidebar is identical to the previous one. The main area is titled 'Shared' and shows a single folder icon with the message 'This folder is empty.'

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



The screenshot shows the Azure DevOps web interface. At the top, there's a header with a search bar and some navigation icons. Below the header, the user's organization is shown as 'mrktalkstech'. Under the 'Projects' tab, there is one project named 'Azure Tutorials'. A small icon labeled 'AT' is next to the project name.

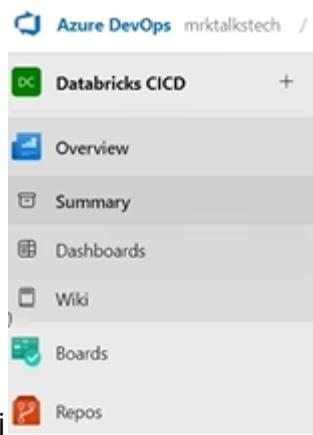
8. Now we'll create a brand new project..



The screenshot shows the 'Create new project' dialog box. The 'Project name \*' field contains 'Databricks CICD'. The 'Visibility' section shows two options: 'Public' (disabled) and 'Private' (selected). A note below states: 'Public projects are disabled for your organization. You can turn on public visibility with organization policies.' At the bottom, there is an 'Advanced' button and a 'Create' button.

now we have created a

project with the name CICD



9. Next we'll create a new repo inside the project go to repos and click in new repo

A screenshot of the 'Repos' page in the 'Databricks CICD' project. The page title is 'Databricks CICD is empty.' A dropdown menu is open over the 'New repository' button, which is highlighted with a cursor icon. Other options in the menu are 'Import repository' and 'Manage repositories'. The page also includes sections for cloning the repository via HTTPS or SSH, and a search bar at the top right.

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

A screenshot of the 'Create a repository' dialog box. It starts with a 'Repository type' dropdown set to 'Git'. Below it is a 'Repository name \*' input field containing 'Databricks CICD Tutorial'. There is a checked checkbox for 'Add a README'. At the bottom, there is a dropdown for '.gitignore' settings set to 'None', and a note stating 'Your repository will be initialized with a main branch.'

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

The screenshot shows the Azure DevOps interface for a project named 'Databricks CI/CD'. On the left, the navigation menu is open, showing options like Overview, Boards, Repos, Files, Commits, Pushes, Branches, Tags, Pull requests, Pipelines, Test Plans, and Artifacts. The 'Repos' option is selected. In the center, a repository named 'Databricks CI/CD Tutorial' is displayed. The repository page includes a search bar at the top right, a 'Set up build' button, and a 'Clone' button. Below the repository name, there's a table showing a single file: 'M1 README.md'. The file was added 'Just now' by 'f6583c26' with the commit message 'Added README.md'. Below the file list, there are sections for 'Introduction', 'Getting Started', 'Build and Test', and 'Contribute', each containing placeholder text for users to fill in.

## 12. Lets go back to dev workspace

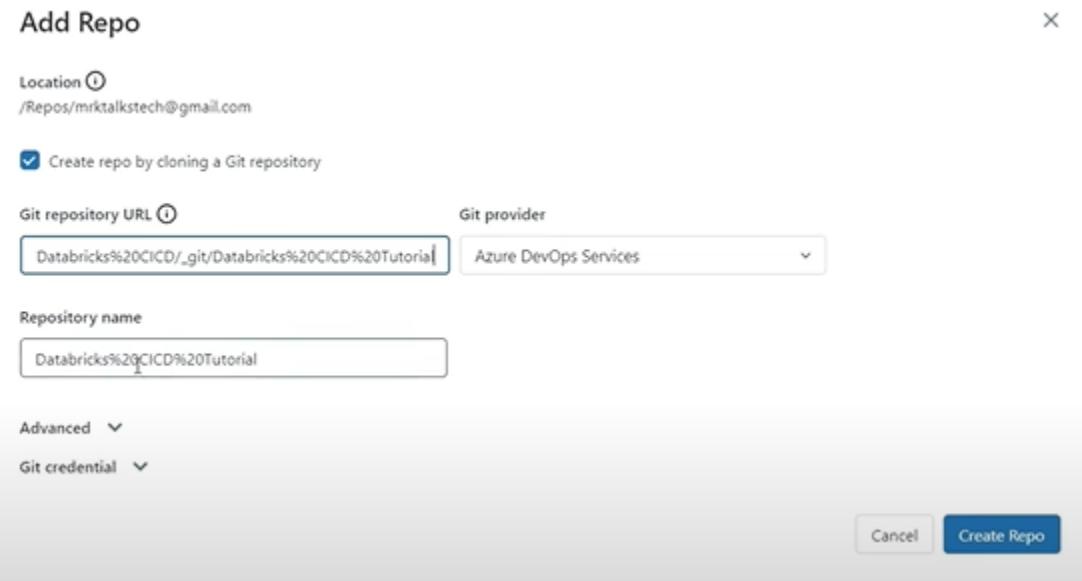
The screenshot shows the Databricks workspace interface. The left sidebar has sections for New, Workspace, Recents, Data, Workflows, Compute, SQL, Data Engineering, Machine Learning, and more. The 'Repos' section is currently selected. In the main area, under the 'Shared' tab, a table lists three repositories: 'bronze to silver', 'silver to gold', and 'storagemount', all owned by Kishor Kumar and created on 4/16/2023. There are 'Share' and 'Add' buttons at the top right of the table.

## 13. Here we have repos and click on add repo

The screenshot shows the Databricks workspace interface, similar to the previous one but with a different view. The left sidebar shows the 'Repos' section selected. In the main area, under the 'Repos' tab, there is a message: 'Your Repos user folder is currently not visible because you haven't created a repo yet. Your Repos user folder will be visible once you add a repo.' Below this message, there is a table with a single row for a folder icon, labeled 'This folder is empty'. At the top right of the table, there are 'Share' and 'Add' buttons.

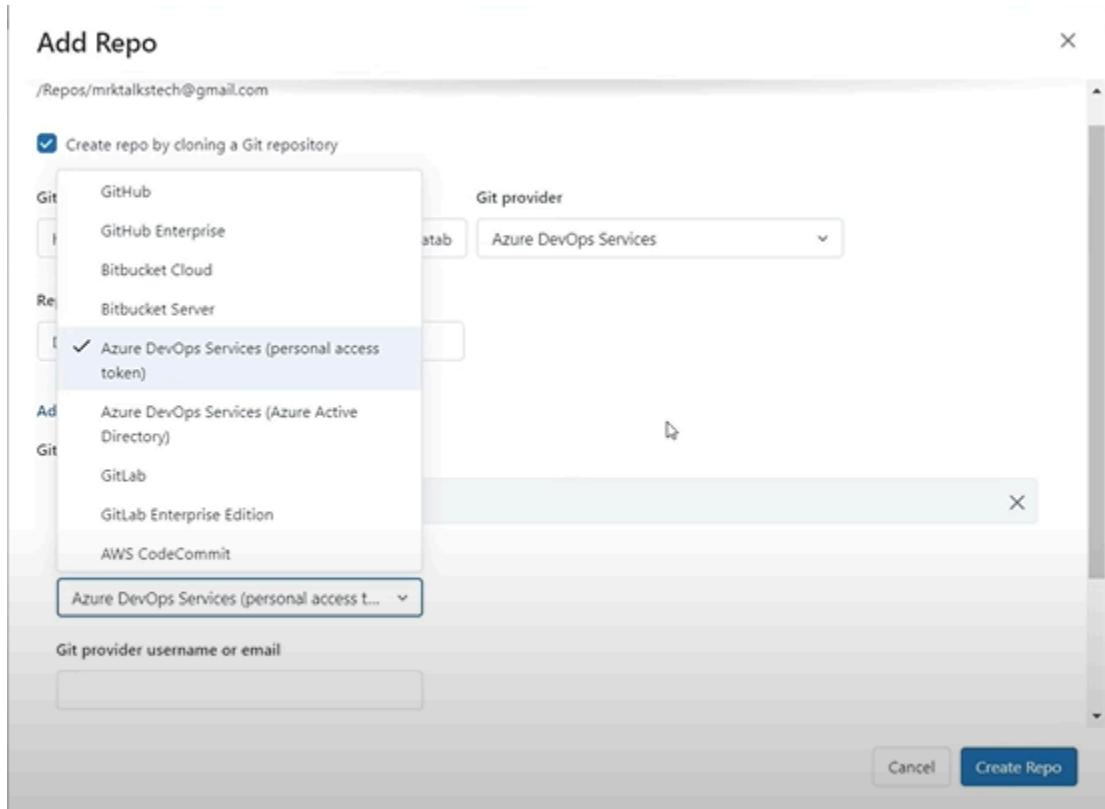
#### 14. Now we'll provide necessary details

---



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.  Create repo by cloning a Git repository 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

The screenshot shows the Microsoft Azure Databricks interface. On the left, the sidebar includes options like New, Workspace, Recents, Data, Workflows, Compute, SQL, SQL Editor, Queries, Dashboards, Alerts, Query History, SQL Warehouses, Data Engineering, Job Runs, Data Ingestion, Delta Live Tables, Machine Learning, Experiments, Features, Models, and Help. The main area displays a repository named "Databricks%20CICD%20Tutorial" under the path "Repos > mrktalkstech@gmail.com". The repository contains a single file named "README.md". The top right corner shows the user "dbw-mrk-demo-01" and the email "mrktalkstech@gmail.com". A small watermark "MR. K TALKS TECH" is visible in the bottom right.

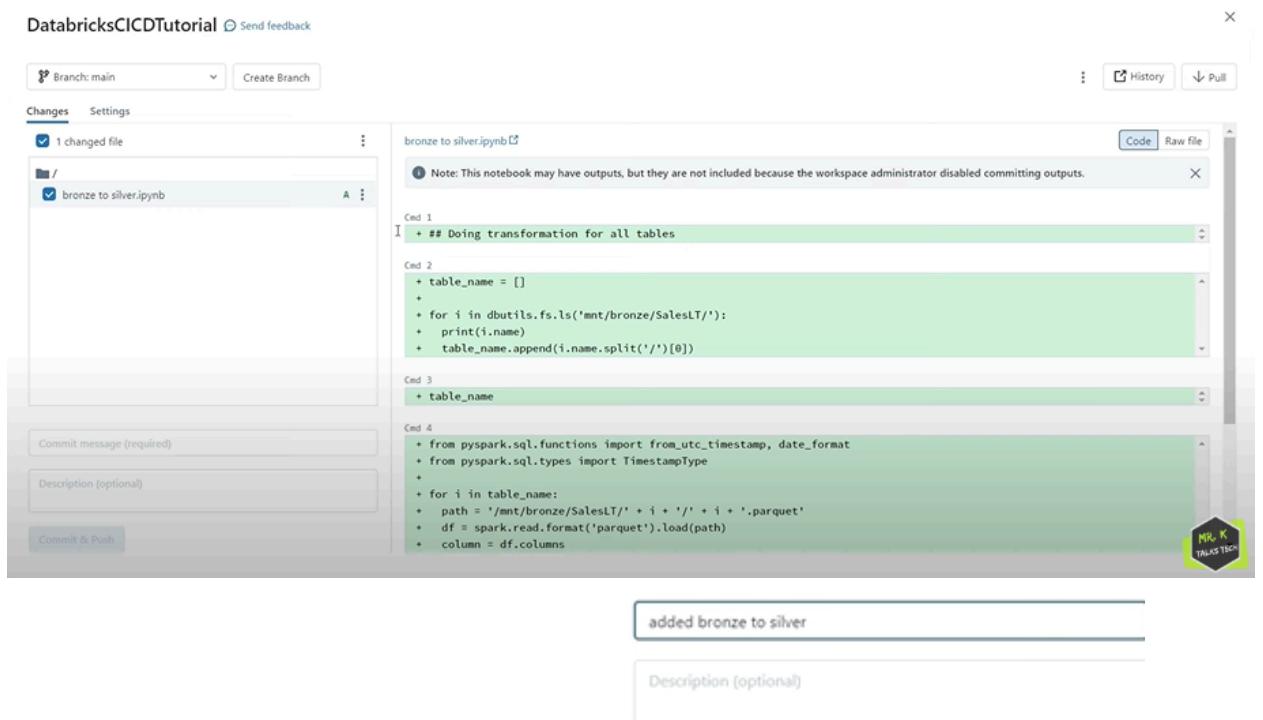
## Branch Protection

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

The screenshot shows the Microsoft Azure Databricks interface. The sidebar is identical to the previous one. The main area displays a repository named "DatabricksCICDTutorial" under the path "Repos > mrktalkstech@gmail.com". The repository contains two items: a notebook named "bronze to silver" and a file named "README.md". The top right corner shows the user "dbw-mrk-demo-01" and the email "mrktalkstech@gmail.com". A small watermark "MR. K TALKS TECH" is visible in the bottom right.

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



The screenshot shows the Databricks notebook editor interface. A file named "bronze to silver.ipynb" is open. The code cell contains Python code for transforming data from the bronze layer to the silver layer in a Databricks workspace. The code uses PySpark's SQL functions to read parquet files from the bronze layer and write them to the silver layer.

```
bronze to silver.ipynb
Note: This notebook may have outputs, but they are not included because the workspace administrator disabled committing outputs.

Cmd 1
+ ## Doing transformation for all tables

Cmd 2
+ table_name = []
+
+ for i in dbutils.fs.ls('mnt/bronze/SalesLT/'):
+   print(i.name)
+   table_name.append(i.name.split('/')[0])

Cmd 3
+ table_name

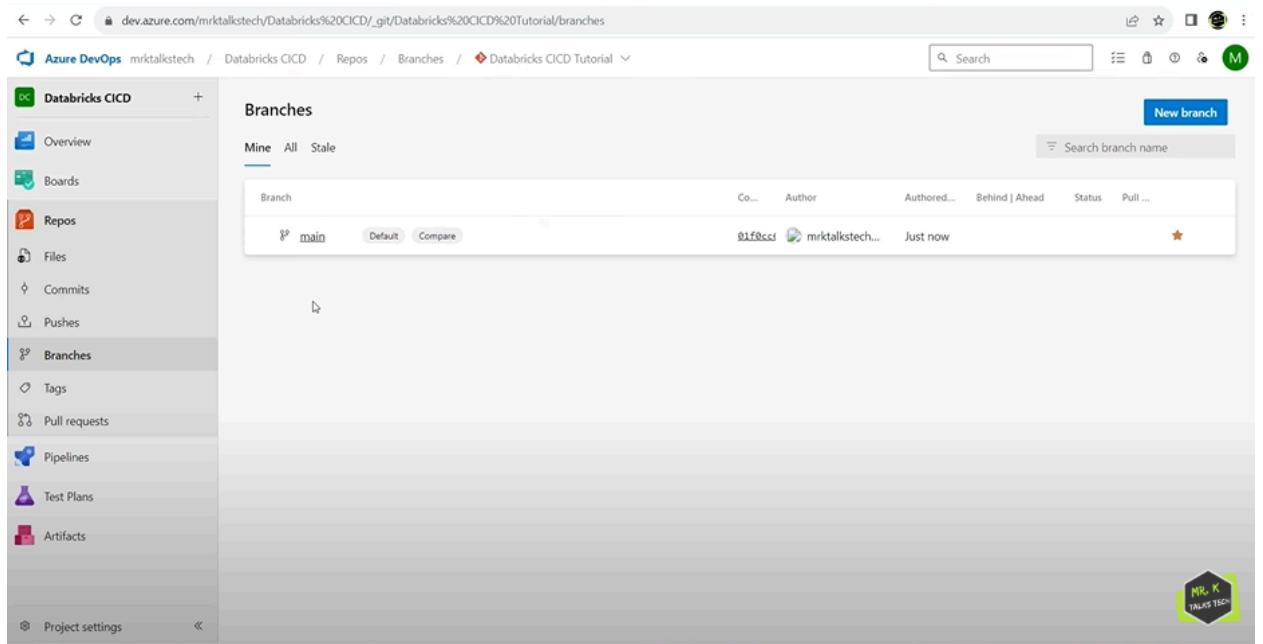
Cmd 4
+ from pyspark.sql.functions import from_utc_timestamp, date_format
+ from pyspark.sql.types import TimestampType
+
+ for i in table_name:
+   path = 'mnt/bronze/SalesLT/' + i + '/' + i + '.parquet'
+   df = spark.read.format('parquet').load(path)
+   column = df.columns
```

Commit message (required): added bronze to silver

Description (optional):

Commit & Push

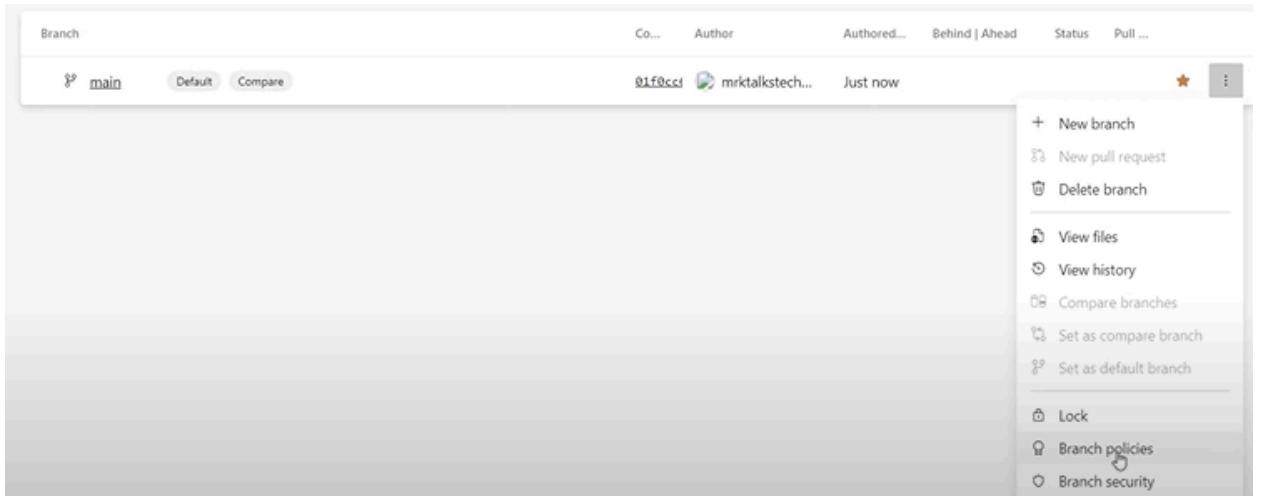
6. Now we'll give commit message and click push
7. Lets see how we can protect our main branch
8. For that go to branches



The screenshot shows the Azure DevOps Branches page for the "Databricks CICD" repository. The "main" branch is selected as the default branch. The page displays basic information about the branch, including the author (mrktalkstech) and the time it was authored (Just now).

Branch	Co...	Author	Authored...	Behind   Ahead	Status	Pull ...
main	mrktalkstech	mrktalkstech...	Just now			

click on branch policies



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

A screenshot of the Databricks CICD Project Settings interface. On the left, there is a sidebar with sections like General, Boards, Pipelines, and Pipelines. The main area shows a project named 'Databricks CICD T...'. Under 'main', the 'Policies' tab is selected. The 'Branch Policies' section is shown with a note: 'Note: If any required policy is enabled, this branch cannot be deleted and changes must be made via pull request.' There is a toggle switch set to 'On' for 'Require a minimum number of reviewers'. A text input field shows '2' as the minimum number of reviewers. Below this, there are four checkboxes: 'Allow requestors to approve their own changes', 'Prohibit the most recent pusher from approving their own changes', 'Allow completion even if some reviewers vote to wait or reject', and 'When new changes are pushed:'. There are also two other policy sections: 'Check for linked work items' (off) and 'Check for comment resolution' (off). A small watermark 'MR. K TALKS TECH' is in the bottom right corner.

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

The screenshot shows the Databricks workspace interface. On the left, there's a sidebar with 'Workspace' and 'Repos'. Under 'Repos', there's a section for 'mrktalktech@gmail.com' containing a 'DatabricksCICDTutorial' repository. The main area displays a table of contents for the 'main' branch:

Name	Type	Owner	Created
bronze to silver	Notebook	Kishor Kumar	4/16/2023
README.md	File	Kishor Kumar	11:16 PM
silver to gold	Notebook	Kishor Kumar	4/16/2023
storagemount	Notebook	Kishor Kumar	4/16/2023

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

The screenshot shows the Databricks commit interface for the 'main' branch. It displays an error message: 'Error pushing changes' with the sub-message 'Remote ref update was rejected. Make sure you have write access to this remote repository.' Below the message, there are buttons for 'Branch: main', 'Create Branch', 'History', and 'Pull'. The 'Changes' tab is selected, showing '2 changed files': 'silver to gold.ipynb' and 'storagemount.py'. The 'silver to gold.ipynb' notebook is expanded, showing its code:

```
silver to gold.ipynb
Note: This notebook may have outputs, but they are not included because the workspace administrator disabled committing outputs.

Cmd 1
+ ## Doing transformation for all tables (Changing column names)

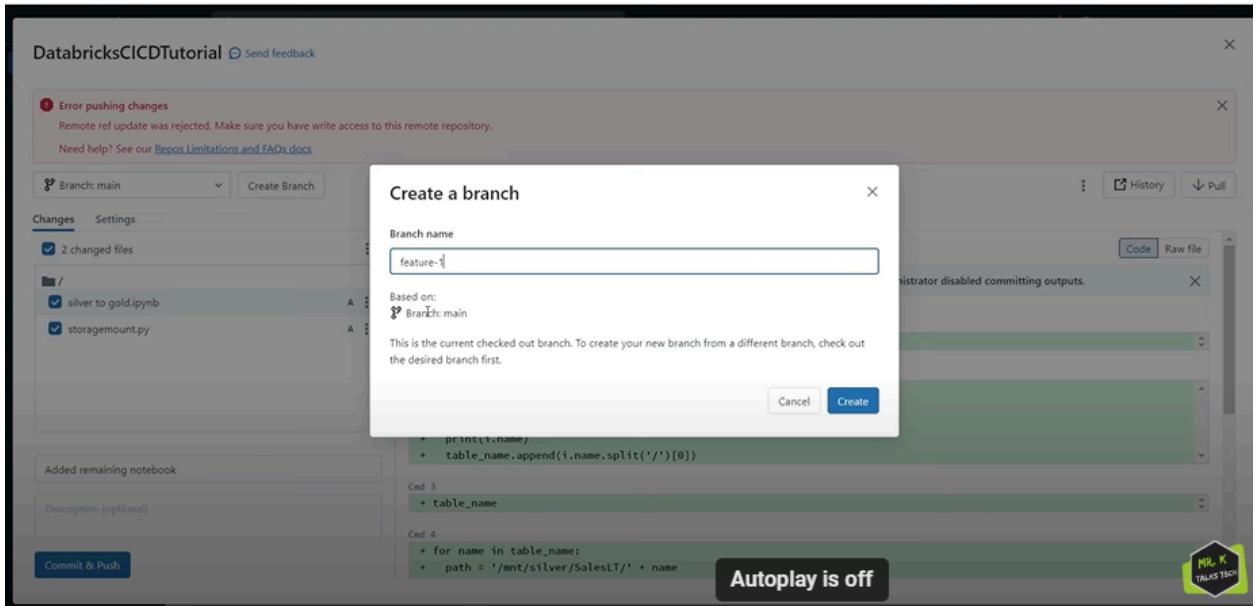
Cmd 2
+ table_name = []
+
+ for i in dbutils.fs.ls('mnt/silver/SalesLT'):
+   print(i.name)
+   table_name.append(i.name.split('/')[-1])

Cmd 3
+ table_name

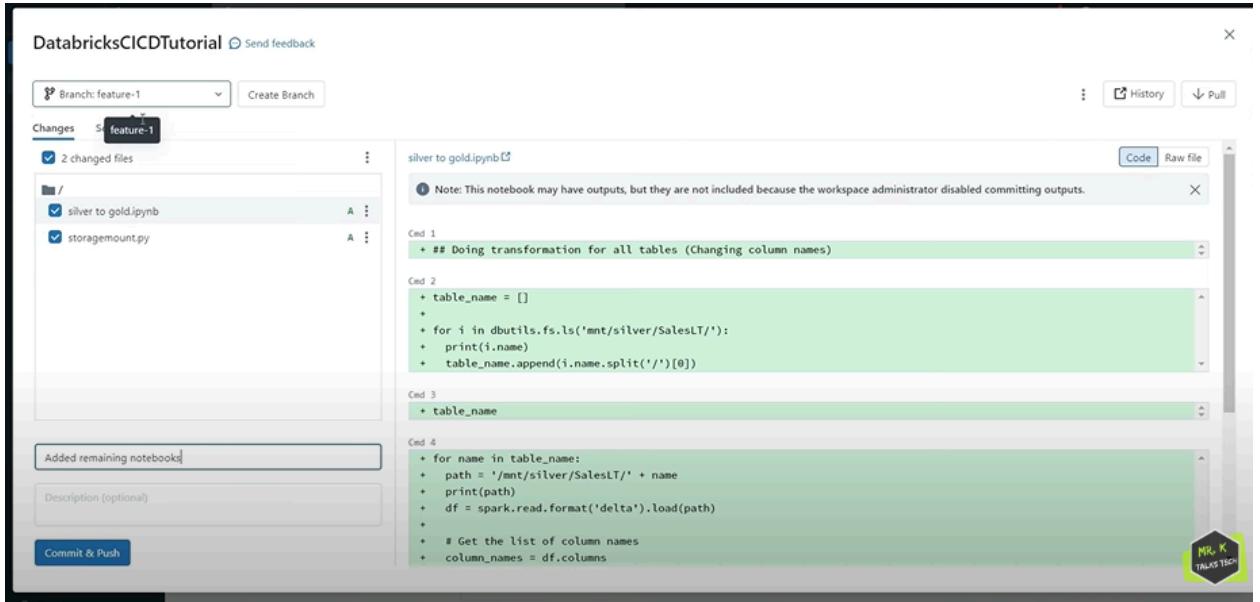
Cmd 4
+ for name in table_name:
+   path = '/mnt/silver/SalesLT/' + name
```

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

Azure DevOps repository interface for the 'Databricks CICD Tutorial' project. The left sidebar shows 'Files' selected. The main area displays two files: 'bronze to silver.ipynb' and 'README.md'. A message at the top says 'You updated feature-1 Just now'. Below it is a table of commits:

Name	Last change	Commits
bronze to silver.ipynb	15m ago	81f0cc6e added bronze to silver mrktalkstech@...
README.md	1h ago	f6583c26 Added README.md mrktalkstech

Sections like 'Introduction' and 'Getting Started' are present. A 'Create a pull request' button is visible.

click on create pull request

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

New pull request form for the 'Databricks CICD Tutorial' project. The source branch is 'feature-1' and the target branch is 'main'. The title is 'Added remaining notebooks' and the description is 'Added remaining notebooks'. The 'Title' field has a character count of 25/4000. The 'Description' field has a character count of 25/4000. The 'Reviewers' section has a placeholder 'Search users and groups to add as reviewers'. The 'Work items to link' section has a placeholder 'Search work items by ID or title'.

## 19. Created pull request

The screenshot shows the Azure DevOps interface for a project named 'Databricks CICD'. The left sidebar has a 'Pull requests' section selected. The main area displays a new pull request with the following details:

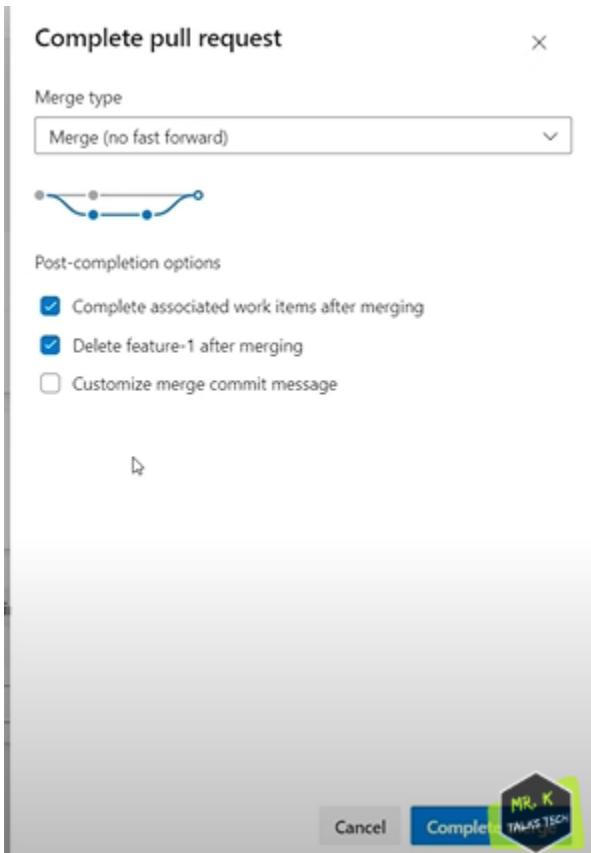
- Title:** Added remaining notebooks
- Description:** Added remaining notebooks
- Markdown supported:** Drag & drop, paste, or select files to insert.
- Reviewers:** Add required reviewers
- Work items to link:** Search work items by ID or title
- Tags:** (empty)

A preview window at the bottom shows the pull request summary: "Added remaining notebooks" with status "Active" and "J1 M". It also indicates "At least 1 reviewer must approve".

20. Now we can see at least one request

and we approve this

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



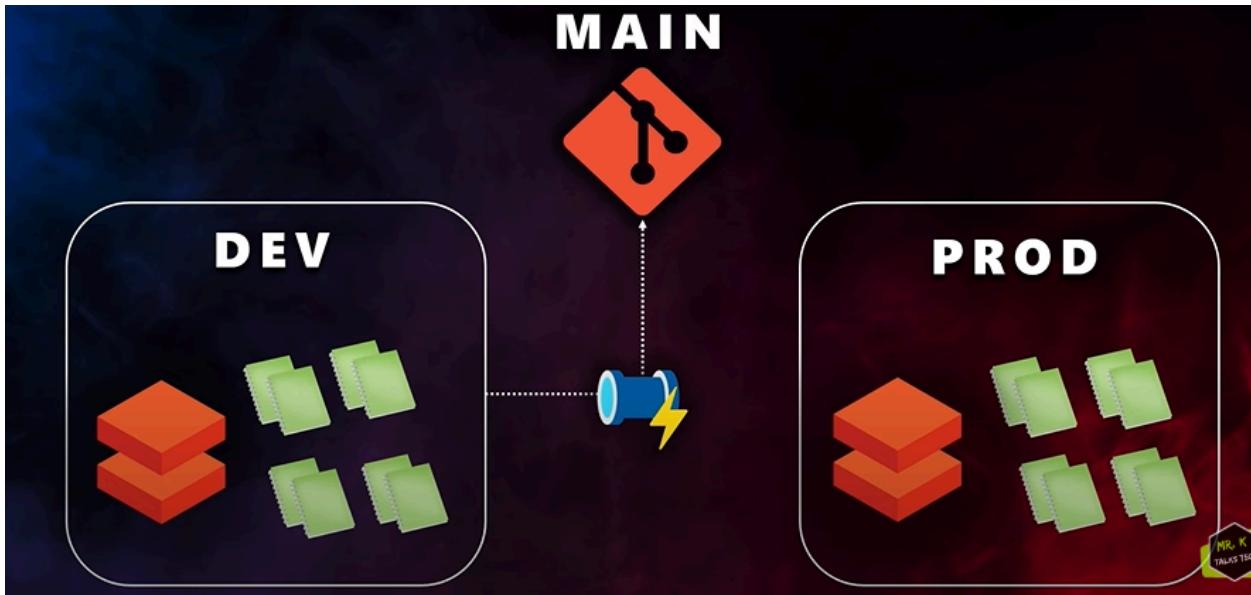
22. Successfully merged with main branch

The screenshot shows the 'Files' page in the Azure DevOps interface for the 'Databricks CICD Tutorial' repository. The left sidebar has 'Repos' selected. The main area shows a list of files: 'bronze to silver.ipynb', 'README.md', 'silver to gold.ipynb', and 'storagemount.py'. The 'README.md' file has a recent commit from 'mrktalkstech'. The commit details are as follows:

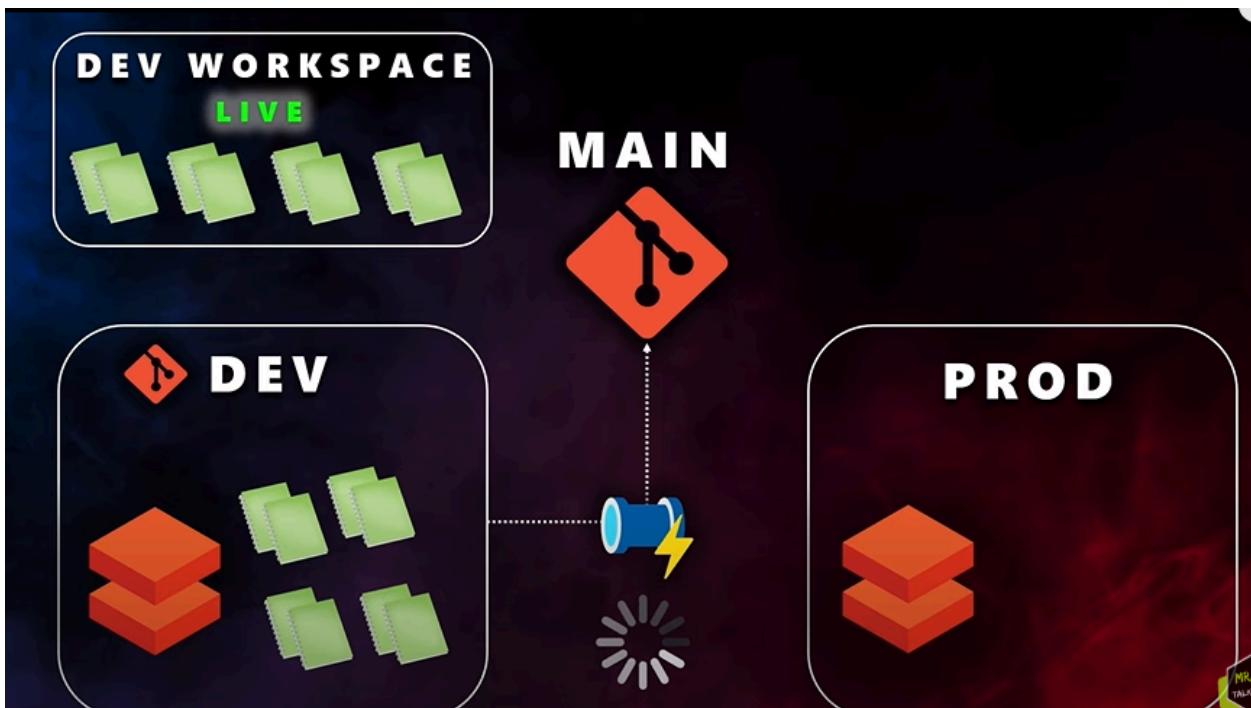
Name	Last change	Commits
bronze to silver.ipynb	22m ago	91f0cc6e added bronze to silver mrktalkstech@...
README.md	1h ago	f6583c26 Added README.md mrktalkstech
silver to gold.ipynb	8m ago	8a55c3af Added remaining notebooks mrktalks...
storagemount.py	8m ago	8a55c3af Added remaining notebooks mrktalks...

## Creating Continuous Integration Pipeline

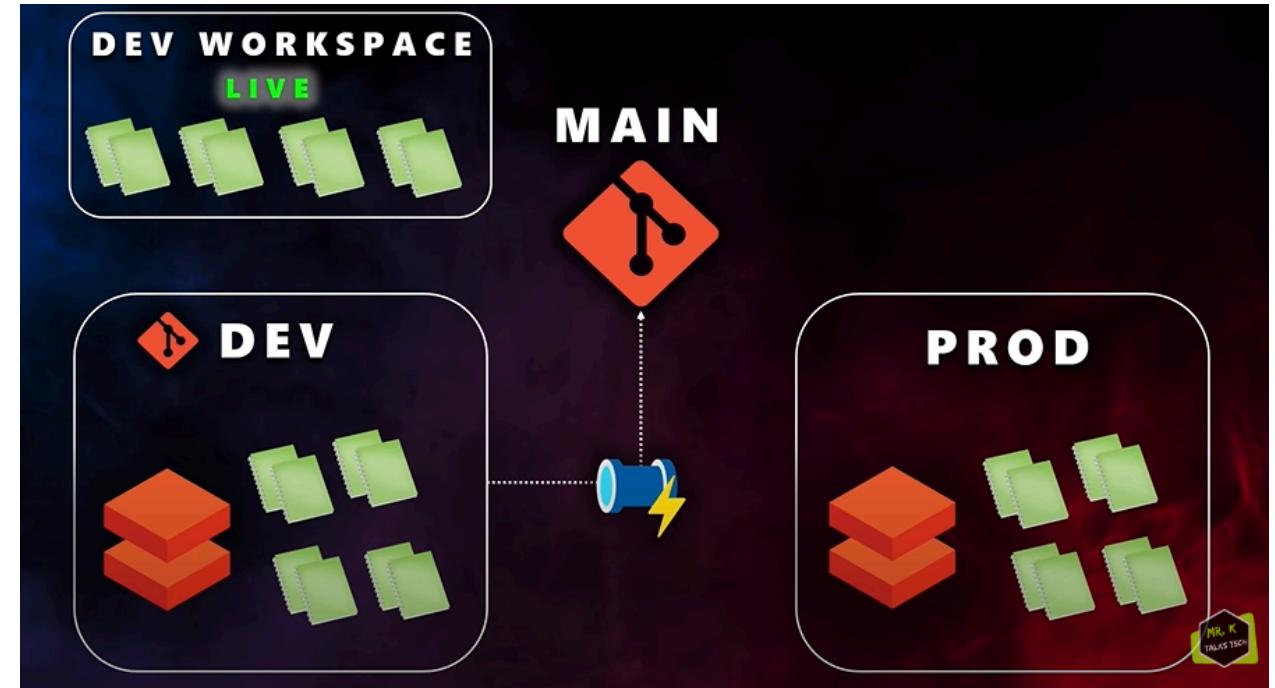
1. Here in our CICD pipeline...when ever we commit in the main branch..CICD will get triggered and commits the changes to prod env



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



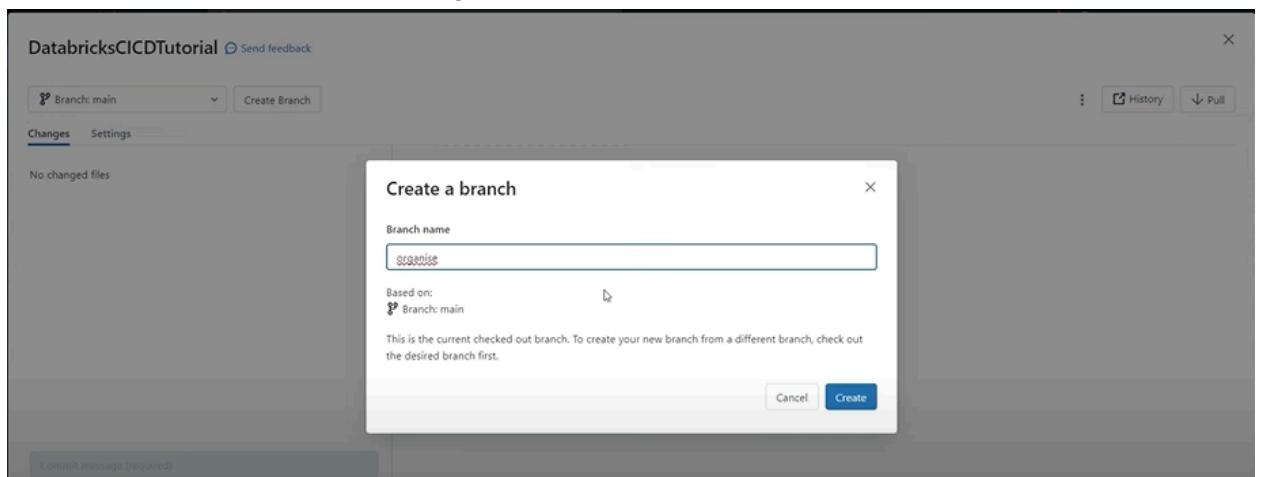
4. Lets see this practically
5. Actually whats the use of live Dev Workspace?..lets go to azure DF studio (refer vid)



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

The screenshot shows the Databricks workspace interface. On the left, there's a sidebar with options like New, Workspace, Recents, Catalog, Workflows, Compute, SQL, and more. The main area shows a file tree under 'Repos > mrktalkstech@gmail.com > DatabricksCICDTutorial'. Inside, there are two main folders: 'extra' and 'notebook'. Both contain three files: 'bronze to silver.ipynb', 'silver to gold.ipynb', and 'storagemount.py'. The 'notebook' folder has a highlighted status icon.

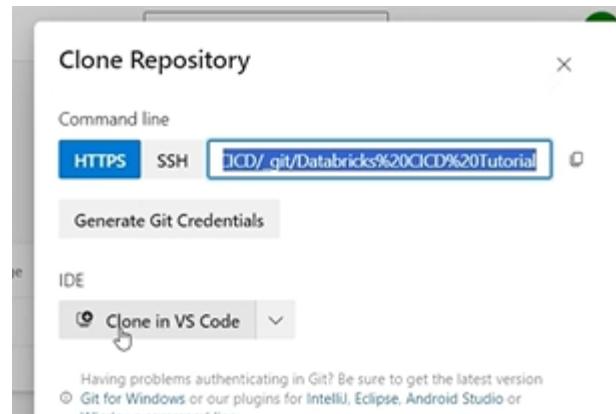
3. So we'll configure the CI pipeline in a way that only...Notebook folders get deployed in Prod env
4. Now we'll commit our changes in the Organize branch

This screenshot shows a GitHub pull request interface for the 'organize' branch of the 'DatabricksCICDTutorial' repository. The 'Changes' tab is selected, showing four files have been changed: README.md, bronze to silver.ipynb, silver to gold.ipynb, and storagemount.py. The README.md file content is displayed in a large text area, providing a template for a project's README.

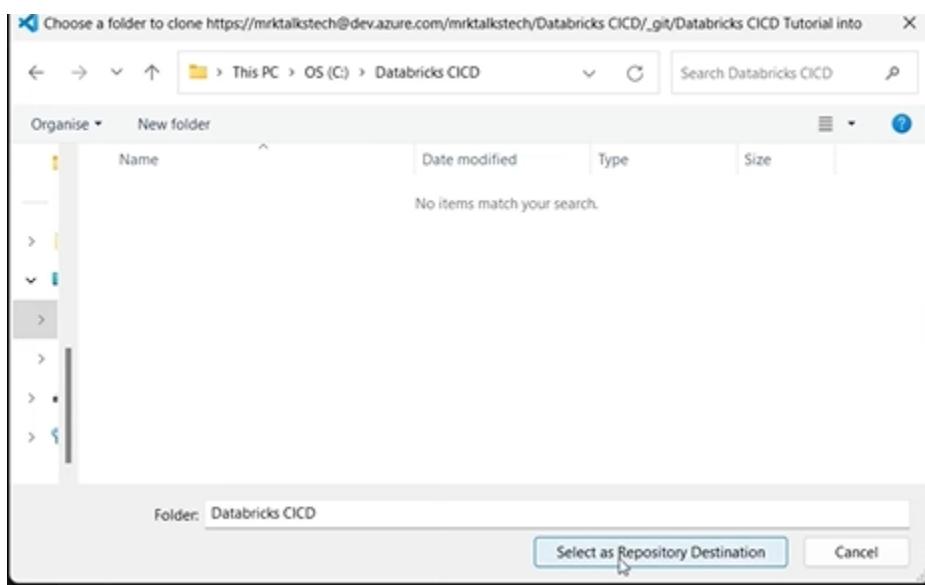
5. Now we'll create the pull request and approve it
6. We can see our main branch has folders now

The screenshot shows the Databricks workspace again. The sidebar is identical to the previous one. The main workspace shows the same file structure under 'Repos > mrktalkstech@gmail.com > DatabricksCICDTutorial'. The 'notebook' and 'extra' folders are now present in the main branch, indicating they have been committed and pushed.

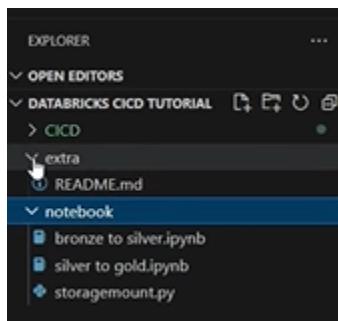
7. Now we have to code in YAML for configuration CI
8. We'll use VScode to write the YAML code..for that we need to clone this repo in our local setup

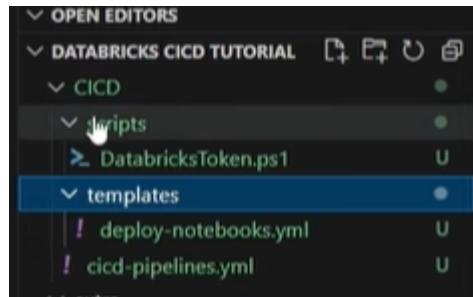


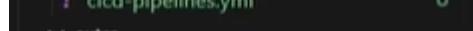
9. Click on Clone and click VScode
10. Now we'll select the dest and clone

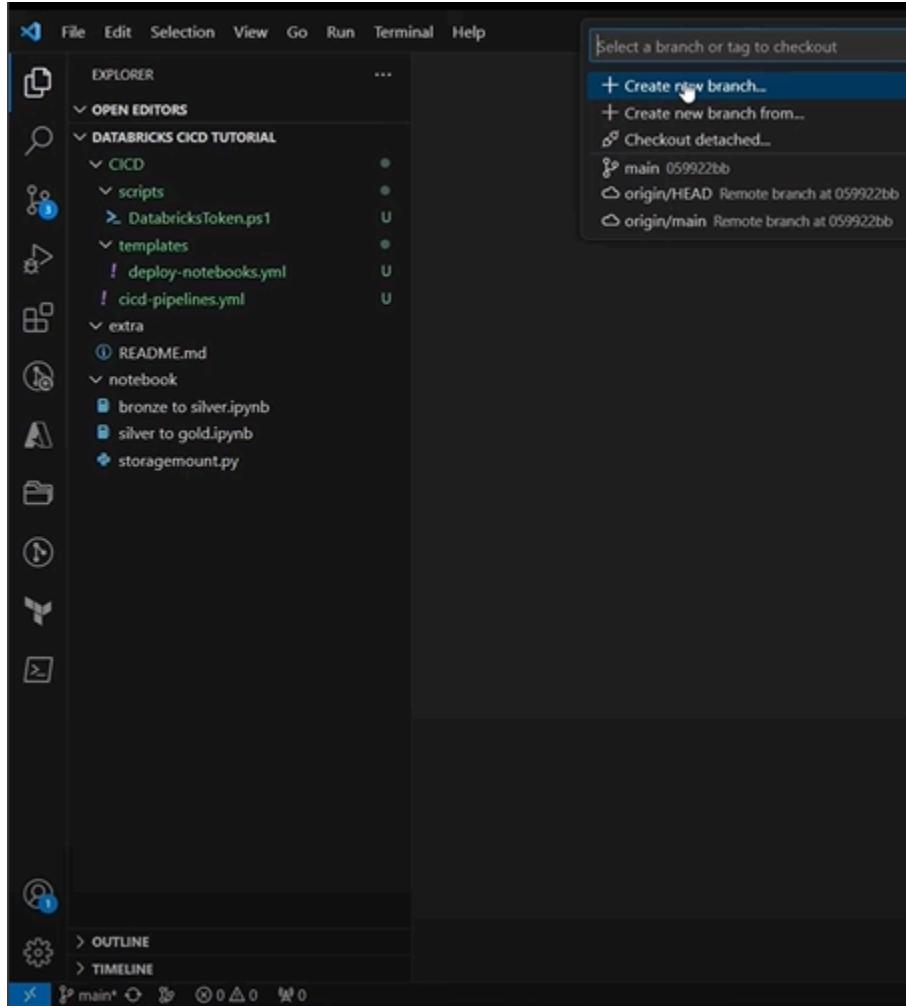


11. Now we will import the CICD folder which has YAML code to our local repo

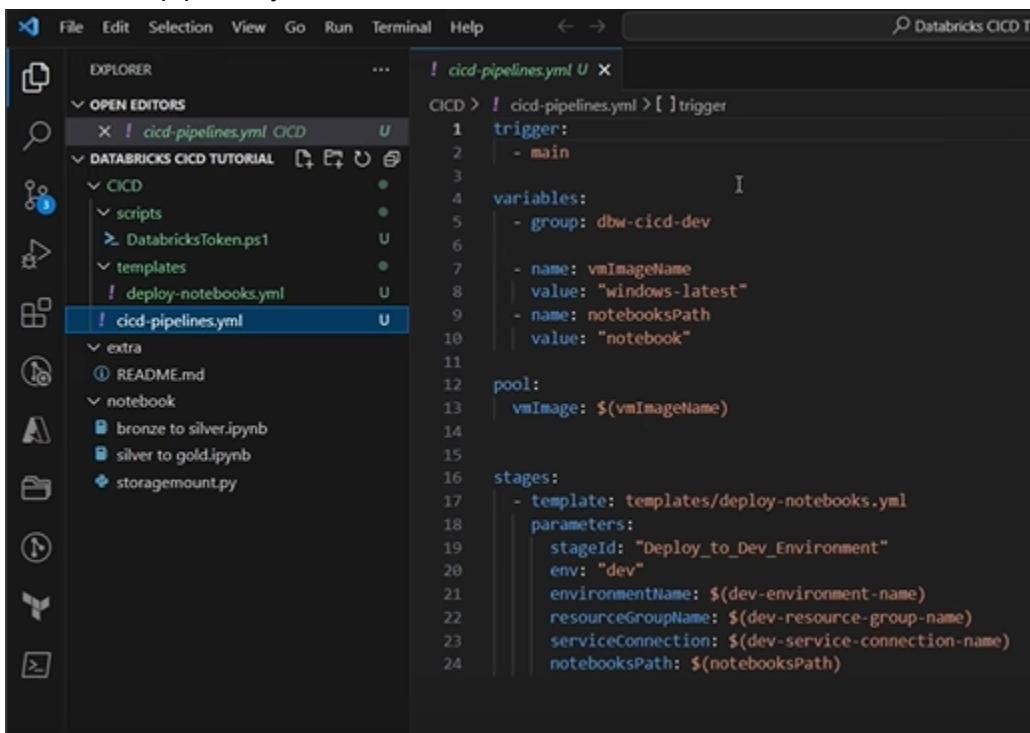




12. Here  deploy-notebooks.yml is the one which we use to implement CI
13. Now here we will create separate branch in vscode



14. Here CICD pipeline.yml is the master file



```
CICD > cicd-pipelines.yml > [ ] trigger
1 trigger:
2   - main
3
4 variables:
5   - group: dbw-cicd-dev
6
7   - name: vmImageName
8     value: "windows-latest"
9
10  - name: notebooksPath
11    value: "notebook"
12
13 pool:
14   vmImage: $(vmImageName)
15
16 stages:
17   - template: templates/deploy-notebooks.yml
18     parameters:
19       stageId: "Deploy_to_Dev_Environment"
20       env: "dev"
21       environmentName: $(dev-environment-name)
22       resourceGroupName: $(dev-resource-group-name)
23       serviceConnection: $(dev-service-connection-name)
24       notebooksPath: $(notebooksPath)
```

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

```
! cicd-pipelines.yml ✘

CICD > ! cicd-pipelines.yml > [ ] variables > {} o > group
  1 trigger:
  2   - main
  3
  4 variables:
  5   - group: dbw-cicd-dev
  6
  7   - name: vmImageName
  8   value: "windows-latest"
  9   - name: notebooksPath
10   value: "notebook"
11
12 pool:
13   vmImage: $(vmImageName)
14
15
16 stages:
17   - template: templates/deploy-notebooks.yml
18   parameters:
19     stageId: "Deploy_to_Dev_Environment"
20     env: "dev"
21     environmentName: $(dev-environment-name)
22     resourceGroupName: $(dev-resource-group-name)
23     serviceConnection: $(dev-service-connection-name)
24     notebooksPath: $(notebooksPath)
```

trigger:  
| - main

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

The screenshot shows the Azure DevOps interface for the 'Databricks CICD' project. The left sidebar has 'Pipelines' selected. The main area shows a 'Files' view with two items: 'extra' and 'notebook'. A tooltip for 'Pipelines' lists: Pipelines, Environments, Releases, Library, Task groups, and Deployment groups.

22. Go to [Azure DevOps](#) mrktalkstech / Databricks CICD / Pipelines / Library for creating variable groups

```
7   - name: vmImageName  
8     value: "windows-latest"  
9   - name: notebooksPath  
10    value: "notebook"
```

23. Here we have ...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 vmlImage 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

```
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

```
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

The screenshot shows the Azure DevOps interface for managing variable groups. On the left, there's a sidebar with options like Overview, Boards, Repos, Pipelines, Environments, Releases, Library, Task groups, Deployment groups, Test Plans, and Artifacts. The Library option is selected. In the main area, a variable group named "dbw-cicd-dev" is displayed. A note at the top says "Variable group must have at least one variable defined." Below that, the "Properties" section shows the group name and a description field. Under the "Variables" section, three variables are listed:

Name	Value
dev-environment-name	to be filled
dev-resource-group-name	rg-data-engineering-project
dev-service-connection-name	to be filled

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

```
! deploy-notebooks.yml U X
CICD > templates > ! deploy-notebooks.yml > [ ]parameters > {} 0 > type
1   parameters:
2     - name: stageId
3       type: string
4     - name: dependsOn
5       type: object
6       default: []
7     - name: env
8       type: string
9     - name: environmentName
10    type: string
11    - name: resourceGroupName
12      type: string
13    - name: serviceConnection
14      type: string
15    - name: notebooksPath
16      type: string
17
18  stages:
19    - stage: "${{ parameters.stageId }}"
20      displayName: "Deploying to ${{upper(parameters.env)}}] Environment"
21      dependsOn: ${{ parameters.dependsOn }}
22      jobs:
23        - deployment: Deploy
24          displayName: "Deploying Databricks Notebooks"
25          environment: ${{parameters.environmentName}}
26          strategy:
27            runOnce:
28              deploy:
```

to this template

```
- name: dependsOn
  type: object
  default: []
```

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

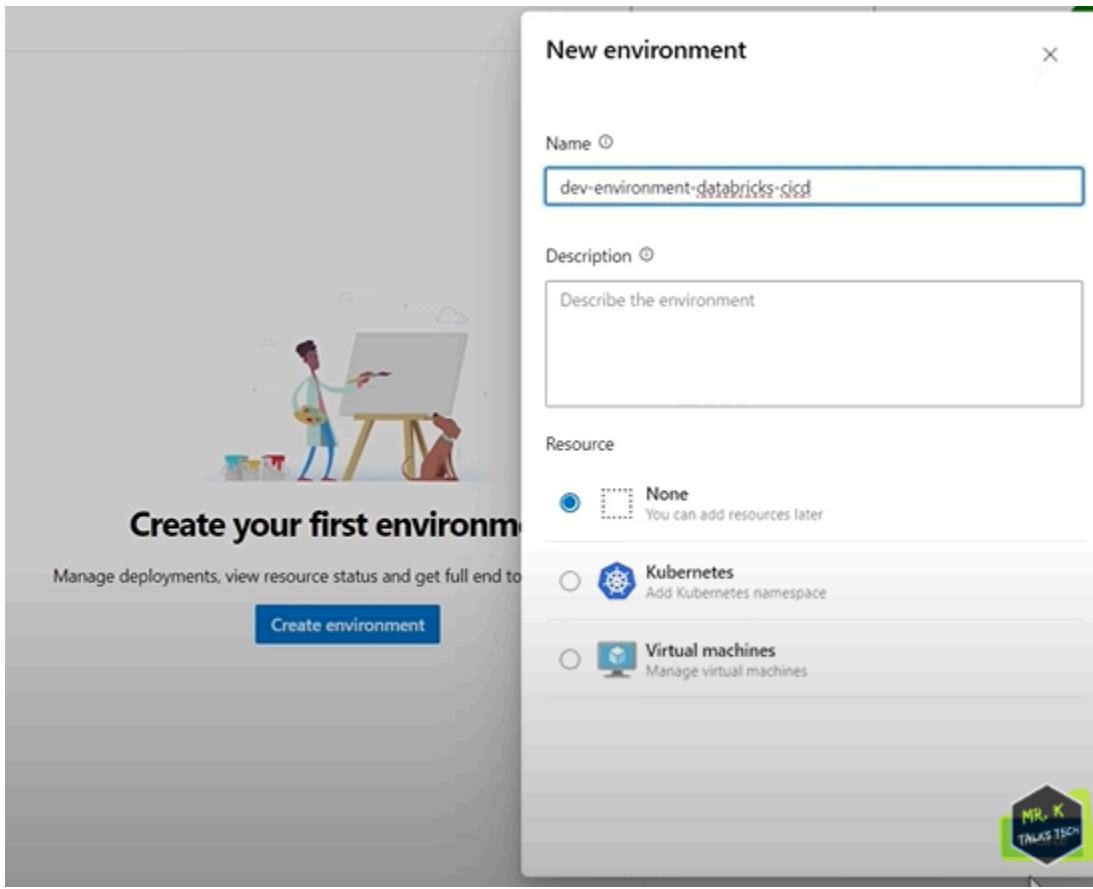
32. Once it gets the value of all the parameters..it proceed further to staging

## Environments and Service connection

The screenshot shows the Azure DevOps interface for a project named "Databricks CICD". The left sidebar has a list of options: Overview, Boards, Repos, Pipelines (which is selected), Pipelines, Environments (selected), Releases, Library, Task groups, Deployment groups, Test Plans, and Artifacts. On the right, there's a large callout for "Create your first environment" with the subtext "Manage deployments, view resource status and get full end to end traceability". A "Create environment" button is at the bottom of this callout. There's also a cartoon illustration of a person painting a canvas.

- 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

Name ↑	Value
dev-environment-name	dev-environment-databricks-cicd
dev-resource-group-name	rg-data-engineering-project
dev-service-connection-name	to be filled

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

The screenshot shows the 'Access control (IAM)' section of the Azure portal. The left sidebar lists various management categories like Overview, Activity log, and Resource visualizer. The main area displays 'Number of role assignments for this subscription' at 19, with a total limit of 4000. A search bar and filters for Assignment type (All), Type (All), Role (All), Scope (All scopes), and Group by (Role) are present. The table lists three items: a 'Contributor' role assigned to a 'data-engineer-group' (Group) with scope 'This resource' and condition 'None'; an 'Owner' role assigned to 'Kishor Kumar' (User) with scope 'Subscription (Inherited)' and condition 'None'; and a 'User Access Administrator' role assigned to 'Kishor Kumar' (User) with scope 'Root (Inherited)' and condition 'None'.

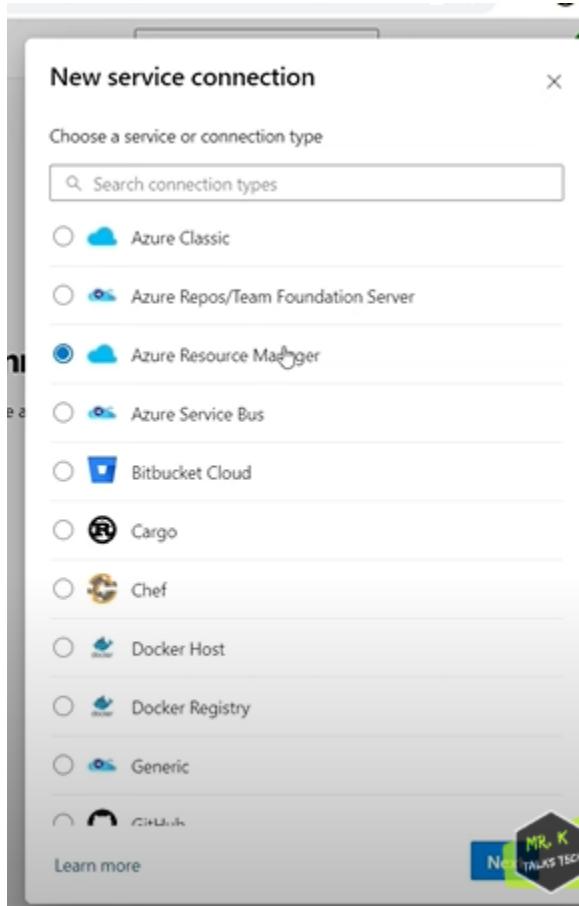
we cannot add azure devops directly here..so thats y we use service connection

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

The screenshot shows the 'Pipelines' library in Azure DevOps. The left sidebar has sections for Databricks CICD, Overview, Boards, Repos, Pipelines, Pipelines, Environments, Releases, Library, Task groups, Deployment groups, Test Plans, Artifacts, and Project settings. The 'Library' section is currently selected. On the right, a variable group named 'dbw-cicd-dev' is displayed. It has tabs for Variable group, Save, Clone, Security, Pipeline permissions, Approvals and checks, and a back arrow. The 'Properties' tab shows the variable group name 'dbw-cicd-dev' and a description field. The 'Variables' tab lists three variables: 'dev-environment-name' with value 'dev-environment-databricks-cicd', 'dev-resource-group-name' with value 'rg-data-engineering-project', and 'dev-service-connection-name' with value 'to be filled'. A '+ Add' button is at the bottom of the variable list.

- Pipelines
- Agent pools
  - Parallel jobs
  - Settings
  - Test management
  - Release retention
  - Service connections
  - XAML build services

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



## 11. We'll use Service principle as authentication

New Azure service connection

Azure Resource Manager using service principal (automatic)

Scope level

Subscription

Management Group

Machine Learning Workspace

Subscription

Azure subscription 1 (841031b2-72a5-4f94-8084-ecf13b4e0cf6) ▾

Resource group

rg-data-engineering-project ▾

Details

Service connection name

dev-service-connection

Description (optional)

Security

Grant access permission to all pipelines

Learn more

Troubleshoot

Back Save

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

Home > rg-data-engineering-project

rg-data-engineering-project | Access control (IAM) ⚡ ...

Resource group

Search

Add Download role assignments Edit columns Refresh Remove Feedback

Number of role assignments for this subscription

20 4000

How likely are you to recommend portal.azure.com to a friend or colleague? \*

0 1 2 3 4 5 6 7 8 9 Extremely likely

Not at all likely

Overview Activity log Access control (IAM) Tags Resource visualizer Events

Deployment Security Deployment stacks Policies Properties Locks

Cost Management Cost analysis Cost alerts (preview) Budgets

4 items (2 Users, 1 Groups, 1 Service Principals)

Name	Type	Role	Scope	Condition
Contributor				
DA	Group	Contributor	This resource	None
mirkalktech-Databricks CICD-8	App	Contributor	This resource	None
Owner				
Kishor Kumar	User	Owner	Subscription (Inherited)	None
User Access Administrator				
Kishor Kumar	User	User Access Administrator	Root (Inherited)	None

13. Next we fill the parameter value

Name ↑	Value	
dev-environment-name	dev-environment-databricks-cicd	🔗
dev-resource-group-name	rg-data-engineering-project	🔗
dev-service-connection-name	dev-service-connection	🔗

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.