Data Engineering Project

Azure stack

Green Fox Academy

2023. 01. 09.

Table of contents

.

1. Business Concept

Our business goal is to get a better understanding of open-source technology trends.
We want to understand which are the "hot" projects and see how the monitored projects/companies are performing compared to others.
Our goal is to better understand the development of the open source tech market - we want to answer these questions using several data sources.

Datasets:
We will use the following freely available datasets provided by BigQuery. (Already extracted from BigQuery and stored in another location):

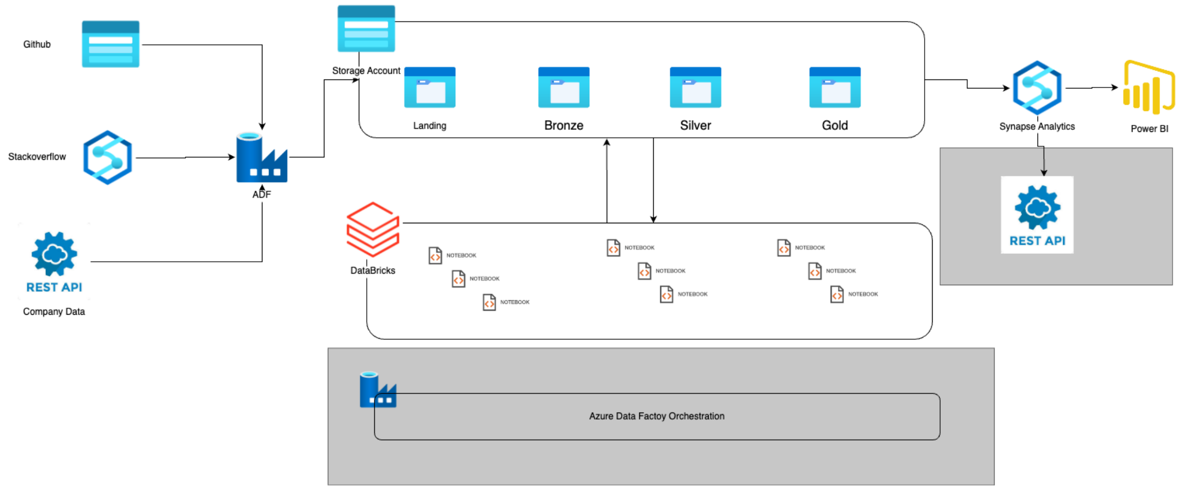
- Github
- StackOverflow

Configuration:
We will get the configuration data from an REST API

The target is to create a common data repository for these datasets, load them into a central database, apply the necessary transformations and create useful insights from the available data, that is presentable with Data Visualizations.

Example Question:
Check if there is a correlation between a trending Stackoverflow post about a tool and the Github stars statistic.

2. Architecture



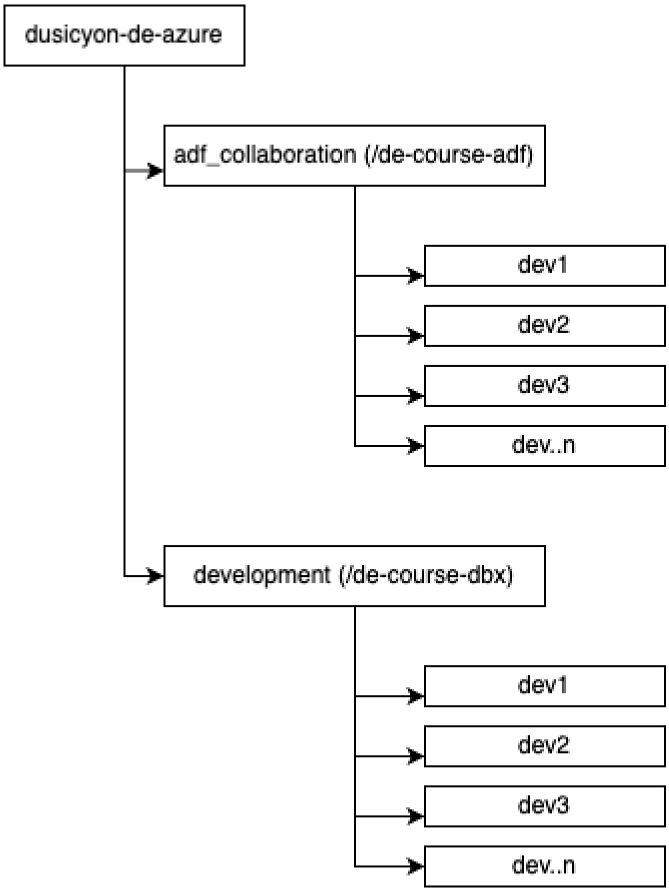
We use the medallion lakehouse architecture, with the following layers in it:

- **landing layer**: target of the ingestion pipelines, data is untyped, untransformed here, contains source data "as-is" (loaded with an ADF copy activity)
- **bronze/raw layer**: source data with minimal transformations but already in delta format (loaded with Databricks Notebooks called from an ADF pipeline)
- **silver/enriched layer**: cleaned and transformed data in delta format (loaded with Databricks Notebooks called from an ADF pipeline)
- **gold/curated layer**: presentation layer which contains aggregated data according to report needs (loaded with Databricks Notebooks called from an ADF pipeline)
- **serving layer**: a view layer in Synapse SQL Serverless Pool on top of the gold layer data which can be used as any other standard database by BI tools and analysts (loaded with Databricks Notebooks called from an ADF pipeline)

Technically the end goal is to create a Master Pipeline in Azure Data Factory with one "p_load_date" parameter which triggered with a specific date loads all of that date"s source data from their original location through all of the layers of our delta lake. We will serve the gold layer data with Synapse Views in SQL serverless Pool on top of it for BI tools and analysts.

3. Branching strategy

We use GitHub as source and version control service. The main idea is that there will be one root branch for the ADF and one for the Databricks. Everybody will have a separate branch under these branches for the development. The two root branches have been already created. This picture illustrates the branching strategy of the project.



4. Ingestion (Landing layer)

There are some suggestions to create and configure services for the project. Mainly you can use the services you already created for learning purposes, but some configuration (git, folders, etc.) is necessary. So you can ignore the create services part, but the configuration is essential!

The ingestion part of the project provides the data for the processing in the source format. This means in this case we pull data from the data sources and store it on the storage account. This part of the project helps to understand the following skills:

- Setup an architecture for a Delta Lake project
- Set security settings between Azure services
- Use Git integration
- Create Delta Lake folder structure
- Pull data from multiple type of datasources
- Store data to Landing layer
- Create flow pipeline for ingestion

We expect the following outcome of the ingestion:

- Have the architecture
- Create folder structure on Storage Account
- Pull data from the sources
- Store data in csv or in json format
- Create an ADF pipeline for orchestration

We will use data for three date:

- 20220731
- 20220831
- 20220930

If you call the API, or the parametrized flow you should use these dates. The first one is the initial load, the second and the third are the incrementals.

4.1 List of tasks

- Create an Azure Data Lake Storage Account
    - you can skip if you already have created

- Create an Azure Data Factory service

  - you can skip if you already have created

- Integrate your ADF with your git repo

- On the Azure portal add Storage Blob Contributor role to the ADF managed identity on the Storage Account you created

- Create Linked Services for sources in ADF and test if the connections are working:

  - Stackoverflow:
    - Synapse linked service
    - Fully qualified domain name: de-course-synw-ondemand.sql.azuresynapse.net
    - Database name: external_db
    - Authentication type: SQL Authentication
      - User name: destudent
      - Password: Stud123

  - GitHub:
    - Azure Data Lake Storage gen2 linked service
    - Authentication type: Account key
    - URL: https://decoursesacc.dfs.core.windows.net
    - Storage account key: VUFSrvMVgBg6u4uqwrBpN8qM9NpS9pwHMbKQCtsZl8bVlmHFBpx1jYir4Jp3wTEpmVudEYt+BalL+AStDN7uIw==

  - Company_detail:
    - REST linked service
    - API: https://de-course-ingest-api.azurewebsites.net/api/
    - Anonymous authentication

- Create Linked Service for target storage in ADF and test if the connection is working:
  - Azure Data Lake Storage gen2 linked service
  - Authentication type: System Assigned Managed Identity
  - From subscription chose your storage account

- Create a dataset for each of the source and target files/tables in ADF

  - it is recommended to organize the datasets in folder structure

  - you can import the schema of the sources but it is not necessary (the ingestion pipelines will be more dynamic if you do not specify the schema)

  - Stackoverflow datasets:
    - Source_StackoverflowPostQuestions:
      - Synapse dataset from Stackoverflow's synapse linked service
      - Table name: stackoverflow.stackoverflow_post_questions
    - Source_StackoverflowPostAnswers:
      - Synapse dataset from Stackoverflow's synapse linked service

            - Table name: stackoverflow.stackoverflow_post_answers
        - Landing_StackoverflowPostQuestions:
            - CSV dataset from the target storage linked service
            - file path: landing/stackoverflow/stackoverflow_post_questions.csv
            - this file will be overwritten with every ingestion pipeline run
        - Landing_StackoverflowPostAnswers:
            - CSV dataset from the target storage linked service
            - file path: landing/stackoverflow/stackoverflow_post_answers.csv
            - this file will be overwritten with every ingestion pipeline run

    - GitHub datasets:
        - Source_GitHubArchiveDay:
            - JSON dataset from the source storage linked service
            - file path: external/github/githubarchiveday_yyyymmdd.json
            - yyyymmdd should be a @p_load_date parameter in the dataset
        - Landing_GitHubArchiveDay:
            - JSON dataset from the target storage linked service
            - file path: landing/github/githubarchiveday_yyyymmdd.json
            - yyyymmdd should be a @p_load_date parameter in the dataset

    - CompanyDetail datasets:
        - Source_CompanyDetail:
            - REST dataset from the source REST linked service
            - relative URL: get_company_data_api?p_load_date=yyyymmdd
            - yyyymmdd should be a @p_load_date parameter in the dataset
        - Landing_CompanyDetail:
            - JSON dataset from the target storage linked service
            - file path: landing/company_detail/company_detail_yyyymmdd.json
            - yyyymmdd should be a @p_load_date parameter in the dataset

- Create 4 pipelines to copy all of the sources of a given date to the landing layer:
    - parameter: @p_load_date (in case of stackoverflow the date parameter is not needed)
    - variable: @v_load_date = @p_load_date (if missing, yesterday should be the default value)
    - the pipelines should contain the following two activities:
        - set variable
        - copy data activity: copy from source to landing dataset

- Create an “Ingestion” flow pipeline with one “p_load_date” parameter and the following activities:
    - set “v_load_date” variable
    - simply add 4 execute pipeline activity to run your previously created pipelines (connect all of them right after the set variable activity so ingestion can run in parallel mode)

4.2 Optional tasks

- Create dynamic stackoverflow datasets, where the table name/file name is a parameter

- Add an ingestion datetime column to the stackoverflow target datasets so you can see when was the data loaded from the source database

- Copy the github and company data sources to the following folder structure:
    - landing/github/yyyy/mm/dd/github_yyyymmdd.json
    - landing/company_detail/yyyy/mm/dd/company_detail_yyyymmdd.json
    - where the yyyy/mm/dd values should be set from the @p_load_date parameter

- Improve your "Ingestion" flow pipeline with the following logics:
    - delete the 4 direct execute pipeline activities
    - create an array which contains the 4 source file names or a config file with the file names and the source type (github/stackoverflow/company_detail) in it
    - create a for each activity which iterates through the items in parallel mode
    - inside the for each activity create a switch activity which executes the right ingestion pipeline for all of the file names (for example if the source type or the beginning of the file name is stackoverflow execute the stackoverflow ingestion pipeline)
    - probably this task in this form does not make sense for these 4 source files but in case of more source files this approach can significantly simplify your ingestion pipeline

- Create a log delta table, and insert one row into it at every pipeline run start and one in the end
    - possible logging informations: pipeline name, run ID, file name, start time, end time, status, error message, etc.
    - use ADF data flow to write the logs to a delta table without Databricks
        - source should be an empty dummy file
        - create the data for logging during runtime with derived columns defined from parameter values
        - sink should be a delta type inline dataset

- At the end of the "Ingestion" flow pipeline create an email notification which sends you an email after every run with the basic run results in it:
    - use Web Activity with Logic Apps to do it
    - a useful video on the subject: https://www.youtube.com/watch?v=zyqf8e-6u4w
    - this task can be time consuming so most likely there will be no time to do it now but it is present in the optional task list so you know that it is worth dealing with the topic in the future when you will have time for it

5. Bronze layer

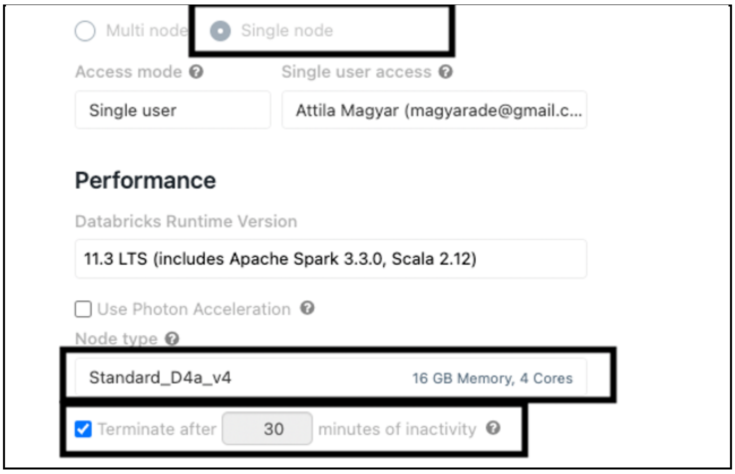
The objective of this phase is to create and load the bronze layer with all the data sources that we previously ingested. The bronze layer's data is almost identical to the original source data but the data is already stored in delta format here.

It's an important note that there is no one, generally accepted solution that can be considered good in all cases. There are plenty of possibilities and choices during the planning and development phases, and most of the time the final solution mode depends on the specific project needs. Throughout this project the general rule of thumb will be that you should achieve the actual task at least one way, it is completely up to you however, how you do it, as long as the final result is OK. We add one possible approach in the list of tasks and in the optionals tasks part we provide ideas for other alternative solutions.

5.1 List of tasks

- Create a Databricks workspace (if you have free subscription create the workspace in the region UK South)
    - you can skip if you already have created

- Create a single node cluster and set the automatic termination after 30 minutes of inactivity (if you have free subscription choose Standard_D4a_v4 node type for the cluster)



- [Integrate your Databricks workspace with your git repo]
    - generate access token with repo scope in github for databricks
    - set git integration in databricks workspace user settings
    - add repo (https://github.com/green-fox-academy/dusicyon-de-azure/) to your databricks workspace
    - create a new feature branch based on the "development" branch (dev_yourname) and checkout this branch

- Create a "setup" folder in the "de-course-dbx" folder in your feature branch and create a "mount_storage" notebook in it to [mount your Storage Account containers to your Databricks workspace]() (each container should be a different mount point, for example /mnt/landing)

    - register an Azure AD application
    - grant storage blob data contributor role to your databricks service principal on your storage account
    - create an Azure Key Vault and add the application secret, the client id and the tenant id to it
    - create an Azure Key Vault-backed Secret Scope in Azure Databricks
    - mount ADLS to Databricks using Secret Scope (secret scopes are only available in premium databricks tier, so if you have standard tier, you can skip this part and hard code the secrets in the notebook)
    - check if all your containers have been mounted

```
display(dbutils.fs.mounts())
```

(3) Spark Jobs

Table

	mountPoint	source
1	/databricks-datasets	databricks-datasets
2	/mnt/gold	abfss://gold@decoursesacc.dfs.core.windows.net
3	/mnt/silver	abfss://silver@decoursesacc.dfs.core.windows.net
4	/databricks/mlflow-tracking	databricks/mlflow-tracking
5	/databricks-results	databricks-results
6	/databricks/mlflow-registry	databricks/mlflow-registry
7	/mnt/bronze	abfss://bronze@decoursesacc.dfs.core.windows.net

Showing all 9 rows. | 8.41 seconds runtime

- Create a "bronze_db" database for the bronze layer tables and specify your bronze container as the location of the database (create a "create_databases" notebook in the setup folder and write the script in it)

- Create an "etl" folder in your feature branch and create 4 notebooks in it to load the previously ingested data sources to the bronze layer in delta format

    - General data modeling tips:
        - Create a primary key in every table, called *_pk*
            - in delta there is no primary key constraint like in SQL database so just create a new column called _pk based on the business key and in the optional task list we added some suggestions about how to check the primary key columns

        - there is no rule how to make the _pk column but one possible solution is:
            - when the business key is an id with numeric values the _pk column can be identical to the id column (id as _pk) but you should preserve the id column and the _pk column as well
            - when the business key is not an id create a hash value from the business key
    - Add *_datetime_utc* postfix to every datetime column name and make sure that every datetime is in UTC format
    - There is three different technical date column you should use when possible:
        - created_at_datetime_utc: when was the row created in the source system (in company detail json there is no such information)
        - loaded_at_datetime_utc: when was the row loaded to the target table
        - valid_date: the load date/file date parameter used at loading (business validity date, in stackoverflow dataset it is not mandatory)
    - Add *is_* prefix to every boolean type ('Yes'/'No', 1/0) column name and convert it to boolean (true/false).
    - prefix the table names with the first letter of the table's layer (for example "b_github" for bronze github table)

- General notebook development tips (a possible notebook structure):
    - import sql functions and types

```python
from pyspark.sql.types import ...
from pyspark.sql.functions import ...
```

    - create a widget for the file date parameter

```python
dbutils.widgets.text("p_file_date", "")
v_file_date = dbutils.widgets.get("p_file_date")
```

    - define the source data schema before reading the data (you can read the source data without defining schema first, explore the dataset and define the schema based on it)

```python
github_schema = StructType([
    StructField('id', StringType()),
    ...
])
```

    - read the data

```python
df = spark.read \
.schema...
.json...
```

    - transform the data

```python
transf_df = df.withColumn("_pk",col("id")) \
...
```

- finally write out the the date to a delta table in the previously created bronze db with overwrite mode

```
final_df = transf_df.select(
    col("_pk"),
...
```

```
final_df.write.mode("overwrite").format("delta").saveAsTable("bronze_db.b_github")
```

- check the loaded bronze table

```
%sql
select *
from bronze_db.b_github
limit 100
```

- in sql notebook fewer step is needed, you can define schema while reading for example or transform and write data in one step (it is also possible to do it in one big step but it will be a less readable code in the end)

- GitHub specific tips:
  - Create the _pk field based on the “id” field
  - Make sure that you read the nested fields correctly, and flatten its values to separate columns in the bronze table (for example from the nested repo field you should create 3 columns: repo_id, repo_name and repo_url)

- Stackoverflow specific tips:
  - it is not necessary to use file date parameter for these source data since there is only one source file for all of the dates but you can use the file date parameter and then filter the source data based on it (where creation_date <= valid_date)
  - Create the _pk field based on the “id” field

- Company detail tips:
  - Create the _pk field based on the “organization_name” field
  - Create a “tags_array” field from the source “tags” field

- Create a “landing_to_bronze” notebook which run all the bronze ETL notebooks sequentially

- Create a master pipeline in ADF which executes your ingestion flow pipeline and the “landing_to_bronze” databricks notebook sequentially

5.2 Optional tasks

- Create the notebooks with pyspark if you created it with sql first or vice versa

- Try to load the bronze tables with autoloader or COPY INTO command
  - instead of using file date parameter you will load the data that is newly arrived to the landing zone since the last loading
  - add the filename as a new column to the bronze tables because only from it can you determine the valid date information

  - with this loading strategy you will have to use append mode instead of overwrite mode (so in the tasks of the silver layer filter the bronze tables accordingly)

- create the bronze tables before running the etl notebooks in a separate “create_tables” notebook, and add constraints to the tables for schema enforcement (for example add NOT NULL constraints to the _pk fields)

- for primary key checking test the uniqueness of the _pk fields (there is not a built in solution to do that, you should write a separate code logic for uniqueness test)

- allow schema drifting and try out what happens if you modify the structure in one of the landing files

6. Silver layer

The objective of this phase is to create the silver layer for the project and understand the different time handling strategies we can use in a data platform.

6.1 List of tasks

- Create a Databricks Notebook, which loads the Company Detail data from the bronze layer to the silver layer with SCD2 type operation

    - The Company Details configuration data can change over time. As mentioned earlier, this source is a business input and it is possible that business users may change its content from time to time. Add, update or remove organizations, change the repositories or just change the list of used tags. The task in this case is to ensure that all these changes are stored in the silver table, but it is possible to use only the latest, current version of the Company Details data.

    - For the project to work properly, it is necessary that an organization in the Company Details data that has already been deleted should not be included in the current version.

    - Add the following technical columns to the dataset:

        - _pk: a hash value created from the organization_name and valid_from_date column values (or try to use an identity column available in Databricks Runtime 10.4+)
        - is_current: boolean, True if the row is currently valid
        - valid_from_date: business validity start date (p_load_date/p_file_date parameter, for example: '2022-07-31')
        - valid_to_date: business validity end date, blank when a row is current (one day prior of the load date when the change/delete operation happened, for example: '2022-08-30')
        - dbx_created_at_datetiime_utc: current timestamp of databricks notebook run when the row is created
        - dbx_updated_at_datetime_utc: current timestamp of databricks notebook run when the row is updated

    - SCD2 hints:

        - try first the sql version of scd2 operation, it's easier to understand and write it in pyspark only if you have time for it

        - a possible code structure:

```
merge_query = f"""MERGE INTO silver_db.s_company_detail base_table
USING (

  --select new records for INSERT
  SELECT new_table.organization_name as mergeKey, new_table.*, ... as valid_from_date, ... as valid_to_date, ... as _pk
  FROM bronze_db.b_company_detail new_table

  UNION ALL

  --select old records for DELETE
  SELECT base_table.organization_name as mergeKey, new_table.*, ... as valid_from_date, ... as valid_to_date, ... as _pk
  FROM bronze_db.b_company_detail new_table
  FULL JOIN silver_db.s_company_detail base_table
  ON new_table.organization_name = base_table.organization_name
  WHERE base_table.is_current = true AND new_table.organization_name is null

  UNION ALL

  --select new records for UPDATE
  SELECT NULL as mergeKey, new_table.*, ... as valid_from_date, ... as valid_to_date, ... as _pk
  FROM bronze_db.b_company_detail new_table
  JOIN silver_db.s_company_detail base_table
  ON new_table.organization_name = base_table.organization_name
  WHERE base_table.is_current = true AND (
   new_table.repository_account <> base_table.repository_account or
   ... )

) staged_updates
ON base_table.organization_name = mergeKey
WHEN MATCHED AND base_table.is_current = true AND (
  base_table.repository_account <> staged_updates.repository_account or
  ... or
  staged_updates.organization_name is null)
THEN
  UPDATE SET is_current = ..., valid_to_date = ..., dbx_updated_at_datetime_utc = ...
WHEN NOT MATCHED THEN
  INSERT(_pk, organization_name, repository_account, ...)
  VALUES(staged_updates._pk, staged_updates.organization_name, staged_updates.repository_account, ...)"""

spark.sql(merge_query)
```

- Create a Databricks Notebook, which loads the GitHub data from the bronze layer to the silver layer incrementally
    - Make your code idempotent, pay attention not to duplicate data in case of a re-run:
        - if you use append mode, any previously loaded data for the same loading date must be deleted from the silver table
        - or you can use a selective overwrite logic instead of append mode
        - in both cases it is recommended to use partitions
    - From the GitHub dataset in the silver layer only a few columns are needed.

Column name	Description
_pk	Primary key for the table
repository_account	There is a delimiter in the repo.name field. It should be split into two columns. It is used as join fields with the Company Details
repository_name	There is a delimiter in the repo.name field. It should be split into two columns. It is used as join fields with the Company Details
user_id	It is the actor.id in the source

event_id	ID column in the source
type	Same in source
created_at_datetime_utc	Same in source

- Add the following technical columns to the dataset:
    - valid_date: validity date (p_load_date/p_file_date parameter, for example: '2022-07-31')
    - dbx_created_at_datetime_utc: current timestamp of databricks notebook run when the row is created

- Create two Databricks Notebooks, which loads the Stackoverflow data from the bronze layer to the silver layer with SCD1 upsert operation
    - Since we have only one source dataset for stackoverflow questions and answers as well, use the following modifications to the stackoverflow bronze layer's etls to check that when a stackoverflow question is updated the SCD1 operation is working correctly:
        - filter the datasets: where last_activity_datetime_utc <= p_load_date
        - pick a column and change its value on 1000 rows when it is running with the first two load date parameters (the last, '2022-09-30' dataset should contain the original data in it)
    - Load all of the source columns and add the following technical columns to it:
        - valid_date: validity date (p_load_date/p_file_date parameter used when the row is created or updated, for example: '2022-07-31')
        - dbx_created_at_datetiime_utc: current timestamp of databricks notebook run when the row is created
        - dbx_updated_at_datetime_utc: current timestamp of databricks notebook run when the row is updated

- Create a "bronze_to_silver" notebook which run all the silver ETL notebooks sequentially
- Modify your master pipeline in ADF to execute the "bronze_to_silver" flow after "landing_to_bronze" runs successfully

6.2 Optional tasks

- There is no silver layer specific optional task, but you can pick any task from the previous layer's optional task list, such as logging, table structure and data quality handling, testing, sql vs pyspark transformations, etc.

6.3 Test examples

- In order to make the sample tables clear, only the columns that are important for understanding are listed here, but of course the silver table must contain all of the columns included in the task description

<u>Company detail test</u>

- Test for Airbyte records

Source data:

p_load_date = '2022-07-31'

organization_name	l1_type	tags
Airbyte	modern_data_stack	airbyte

p_load_date = '2022-08-31'

organization_name	l1_type	tags
Airbyte	modern_data_stack	airbyte

p_load_date = '2022-09-30'

organization_name	l1_type	tags
Airbyte	Modern data stack	airbyte

Expected result in silver table:

organization_name	l1_type	tags	is_current	valid_from_date	valid_to_date
Airbyte	modern_data_stack	airbyte	FALSE	31/07/2022	29/09/2022
Airbyte	Modern data stack	airbyte	TRUE	30/09/2022	

- Test for Dremio records

Source data:

p_load_date = ‘2022-07-31’

organization_name	l1_type	tags
Dremio	modern_data_stack	dremio

p_load_date = ‘2022-08-31’ (there is no Dremio record in this dataset!)

organization_name	l1_type	tags

p_load_date = ‘2022-09-30’

organization_name	l1_type	tags
Dremio	Modern data stack	dremio

Expected result in silver table:

organization_name	l1_type	tags	is_current	valid_from_date	valid_to_date
Dremio	modern_data_stack	dremio	FALSE	31/07/2022	30/08/2022
Dremio	Modern data stack	dremio	TRUE	30/09/2022	

GitHub test

- Load the bronze and silver GitHub table with '2022-07-31' , '2022-08-31' and '2022-09-30' p_load_date_parameters, and after that check the row counts (creation month is the end of month of the created_at_datetime_utc field)

creation_month	valid_date	dbx_created_at_datetime_utc	row_cnt
31/01/2022	31/07/2022	timestamp1	83873
28/02/2022	31/07/2022	timestamp1	83724
31/03/2022	31/07/2022	timestamp1	91902
30/04/2022	31/07/2022	timestamp1	96740
31/05/2022	31/07/2022	timestamp1	96077
30/06/2022	31/07/2022	timestamp1	93951
31/07/2022	31/07/2022	timestamp1	95931
31/08/2022	31/08/2022	timestamp2	123072
30/09/2022	30/09/2022	timestamp3	123890

- After all of the data has been loaded, run again the silver table loader notebook with '2022-09-30' p_load_parameter and check the row counts again: you should have the same row count values but the dbx_creation_datetime_utc value should be different than in the previous check

creation_month	valid_date	dbx_created_at_datetime_utc	row_cnt
31/01/2022	31/07/2022	timestamp1	83873
28/02/2022	31/07/2022	timestamp1	83724
31/03/2022	31/07/2022	timestamp1	91902
30/04/2022	31/07/2022	timestamp1	96740
31/05/2022	31/07/2022	timestamp1	96077
30/06/2022	31/07/2022	timestamp1	93951
31/07/2022	31/07/2022	timestamp1	95931
31/08/2022	31/08/2022	timestamp2	123072
30/09/2022	30/09/2022	timestamp4	123890

Stackoverflow test

Source data:

p_load_date = '2022-07-31'
(activities after '2022-07-31' filtered out and random values modified for testing purpose)
(loaded to silver at timestamp1)

id	title	creation_datetime_utc	last_activity_datetime_utc
73182188	Dummy Title 2022.07	2022-07-31T09:43:19.837+0000	2022-07-31T09:53:20.257+0000

p_load_date = '2022-08-31'
(activities after '2022-08-31' filtered out and random values modified for testing purpose)
(loaded to silver at timestamp2)

id	title	creation_datetime_utc	last_activity_datetime_utc
73182188	Add delay to javascript after button click	2022-07-31T09:43:19.837+0000	2022-07-31T09:53:20.257+0000
73191008	Dummy Title 2022.08	2022-08-01T08:40:44.510+0000	2022-08-01T08:46:57.290+0000

p_load_date = '2022-09-30' (original source data)
(loaded to silver at timestamp3)

id	title	creation_datetime_utc	last_activity_datetime_utc
73182188	Add delay to javascript after button click	2022-07-31T09:43:19.837+0000	2022-07-31T09:53:20.257+0000
73191008	I have to refresh the page for the countdown timer to work	2022-08-01T08:40:44.510+0000	2022-08-01T08:46:57.290+0000
73575322	How to Fix an Encryption Error in Power BI Desktop	2022-09-01T20:55:23.973+0000	2022-09-01T20:55:23.973+0000

Expected result in silver table:

id	title	creation_datetime_utc	last_activity_datetime_utc	dbx_created_at_datetime_utc	dbx_updated_at_datetime_utc
73182188	Add delay to javascript after button click	2022-07-31T09:43:19.837+0000	2022-07-31T09:53:20.257+0000	timestamp1	timestamp2
73191008	I have to refresh the page for the countdown timer to work	2022-08-01T08:40:44.510+0000	2022-08-01T08:46:57.290+0000	timestamp2	timestamp3
73575322	How to Fix an Encryption Error in Power BI Desktop	2022-09-01T20:55:23.973+0000	2022-09-01T20:55:23.973+0000	timestamp3	timestamp3

7. Materials

7.1 Storage Account

Tips:

- Storage account names must be between 3 and 24 characters in length and may contain numbers and lowercase letters only.
- Your storage account name must be unique within Azure. No two storage accounts can have the same name.
- Enable hierarchical namespace to use this storage account for Azure Data Lake Storage Gen2 workloads
- For cost saving purpose:
    - Disable soft deletes
    - Set Local redundancy
- Download Azure Storage Explorer if you want to manage your Azure cloud storage resources from your desktop

Current project structure:

- **decoursesacc** storage account
    - landing container
    - bronze container
    - silver container
    - gold container
    - system container
    - sandbox container

Future modification options:

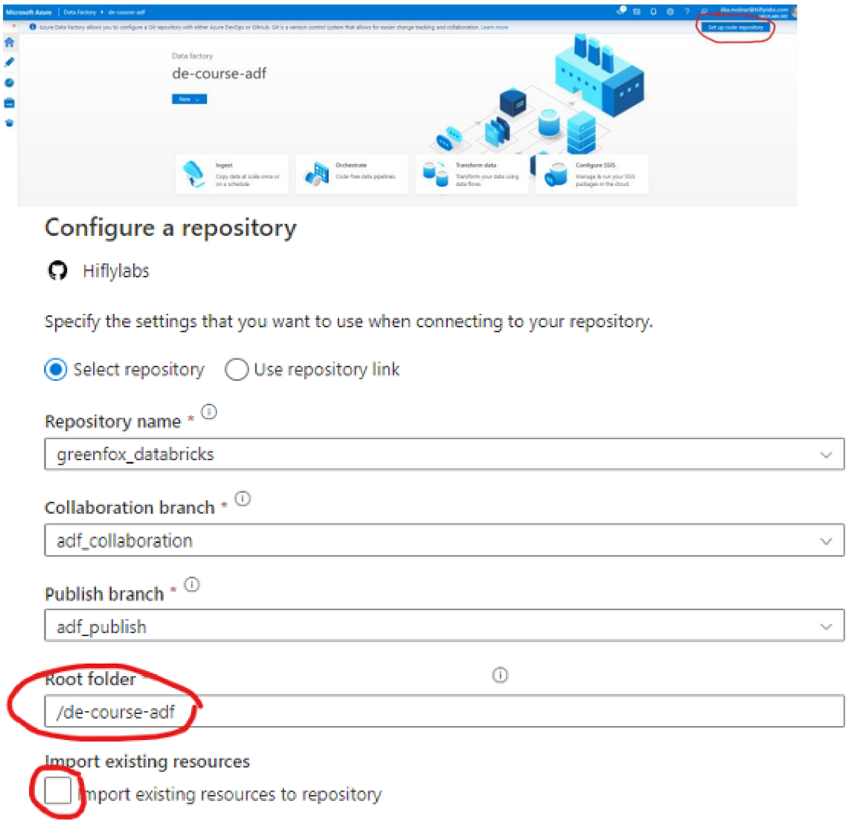
- We create only one environment at start (for testing and production purpose we should create separate storage accounts)
- In the dev storage account it is also a possible approach to create separate containers for the developers (in that case the bronze/silver/gold layers would be folders in dev environments and containers in test/production environments)
- Landing can be a separate, environment independent storage account

7.2 Azure Data Factory

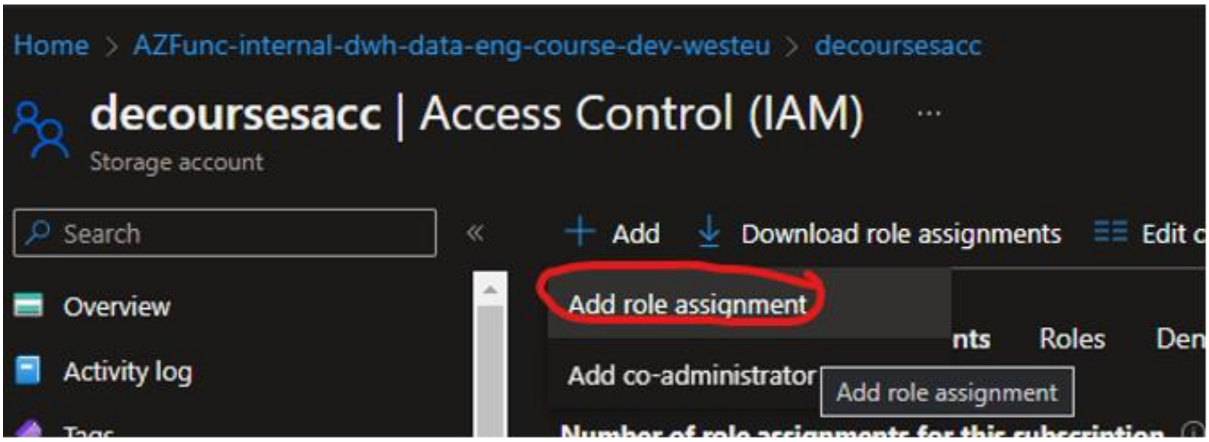
No special configuration is needed.

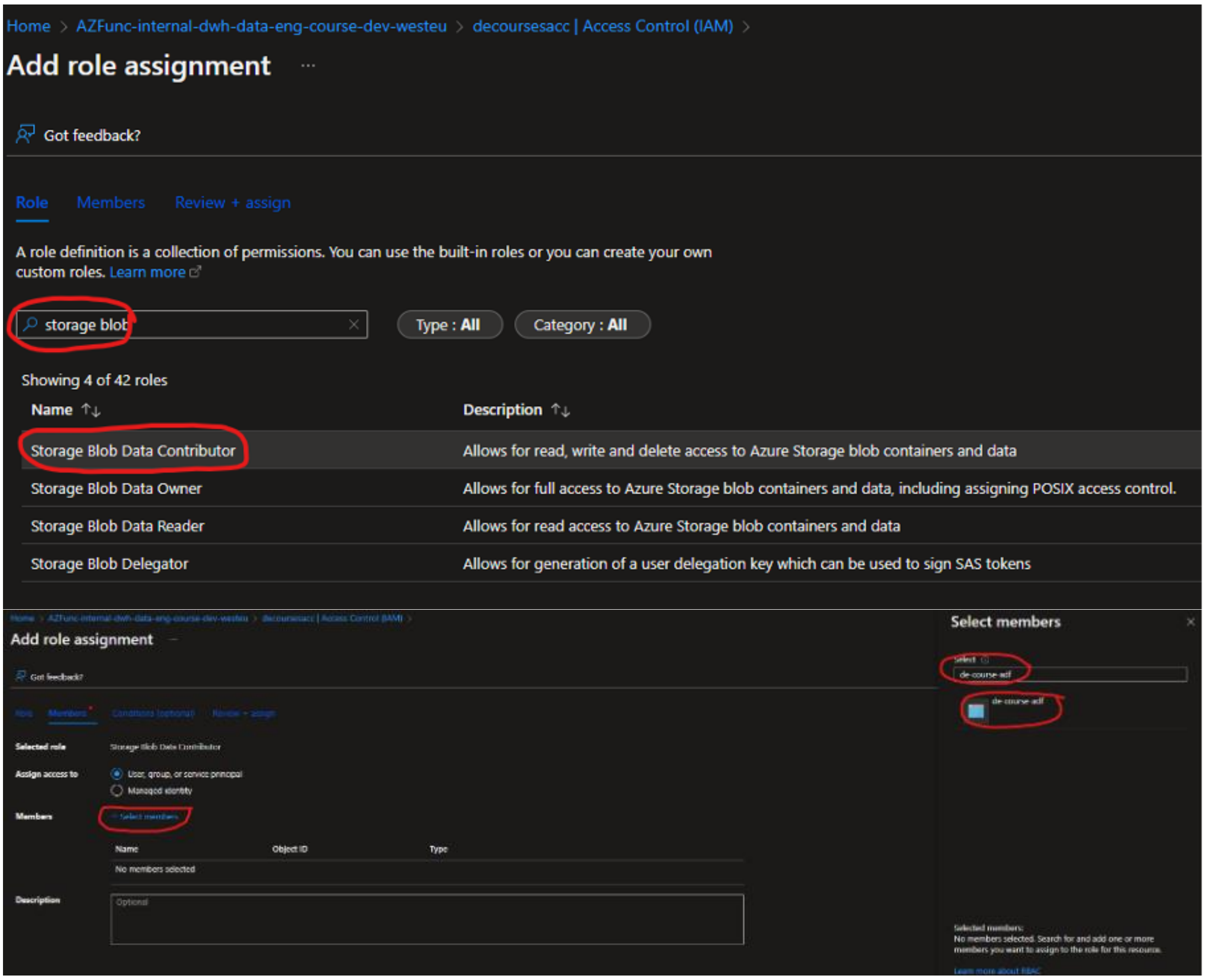
7.2.1 Azure Data Factory git integration

- Log into Github on ADF page
- Set adf_collaboration branch (everybody should use this)
- Set root_folder!
- Create your branch to work on. Give unique name such as **adf_ma.** You will need another branch for databricks!



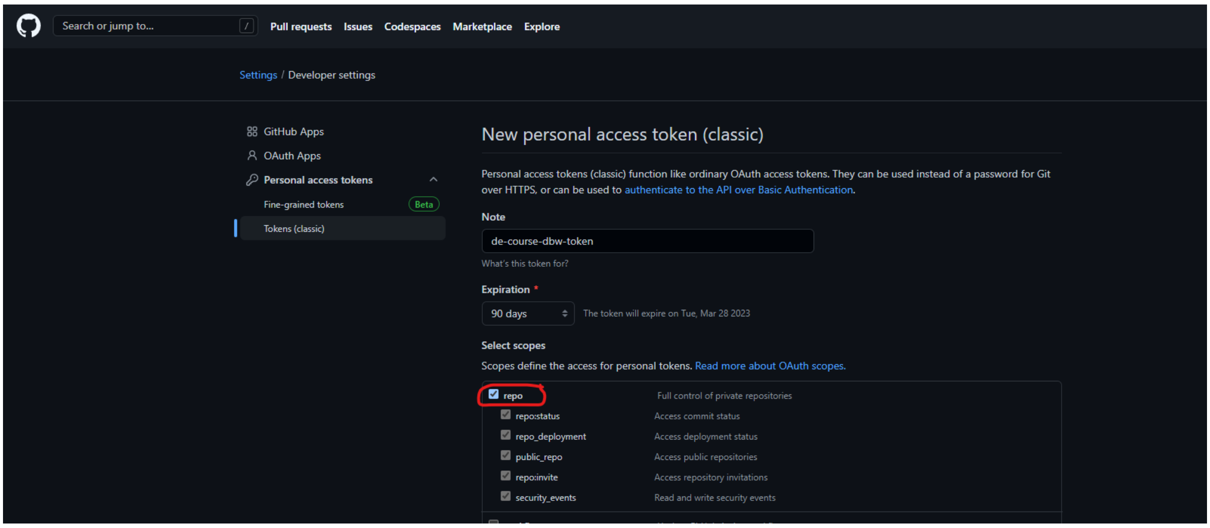
7.2.2 BlobStorage contributor access



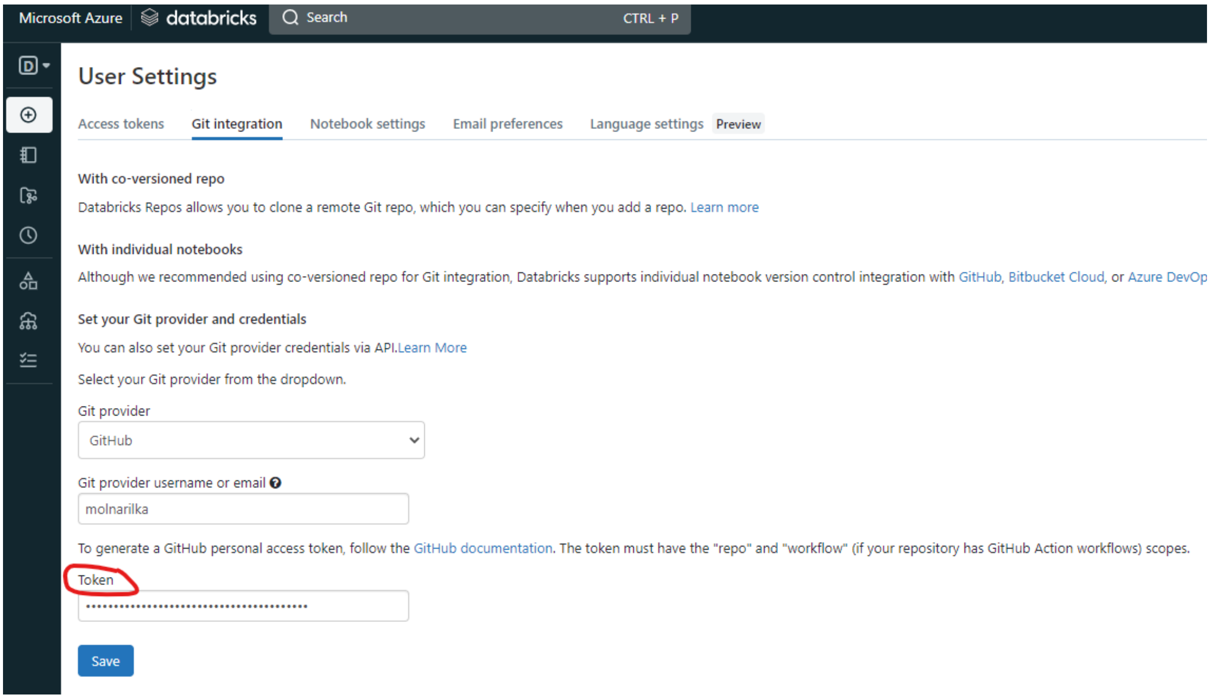


7.3 Databricks git integration

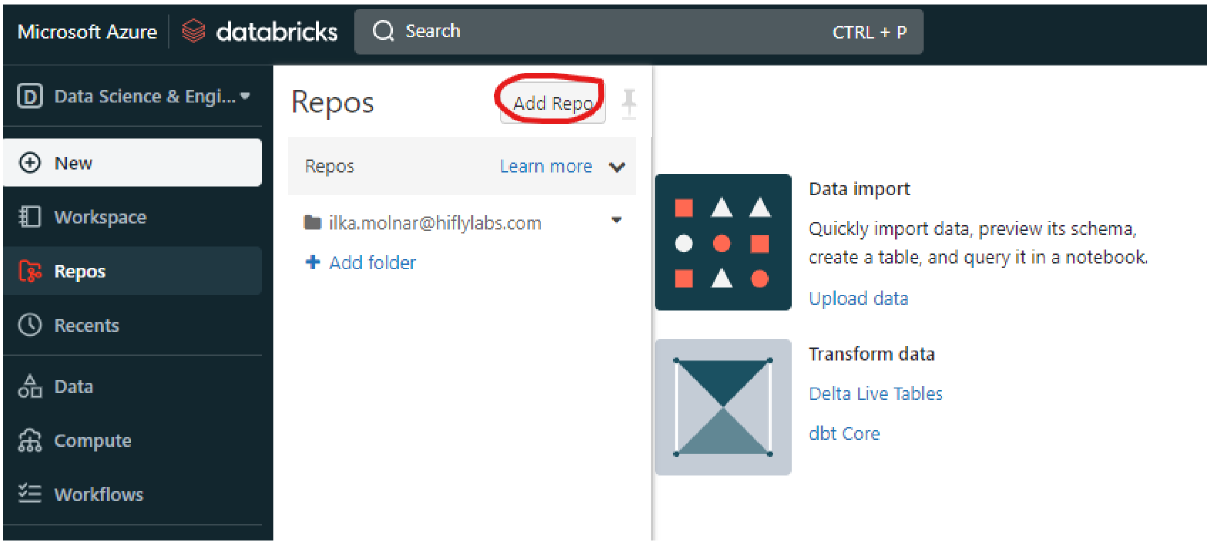
- Generate access token with repo scope in github for databricks (Settings/Developer settings/Public access tokens/Token(classic))

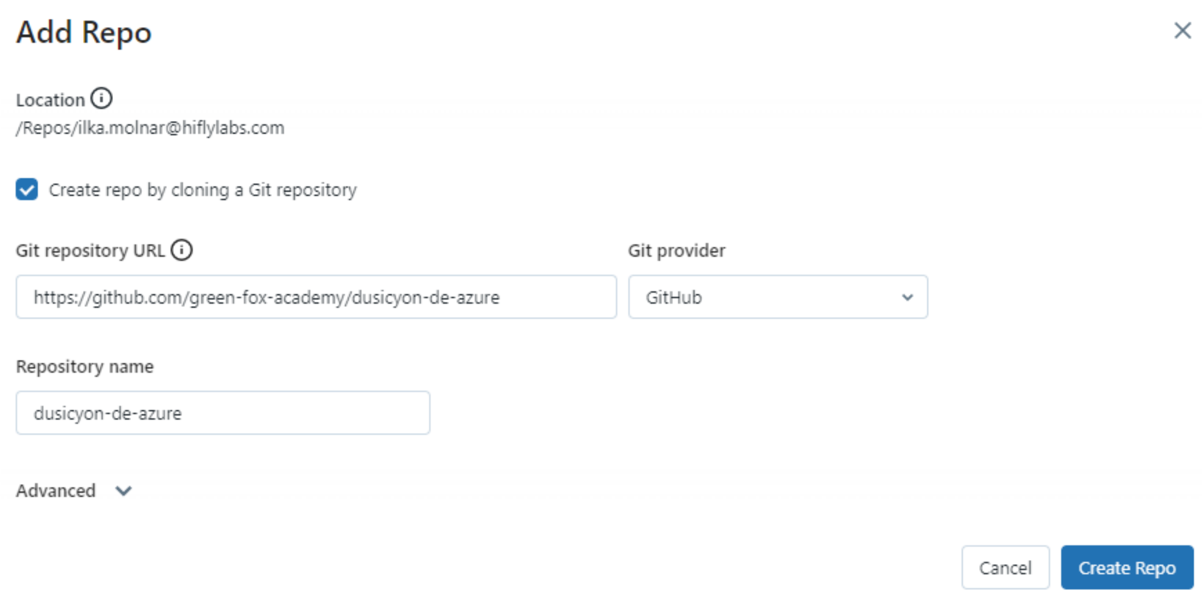


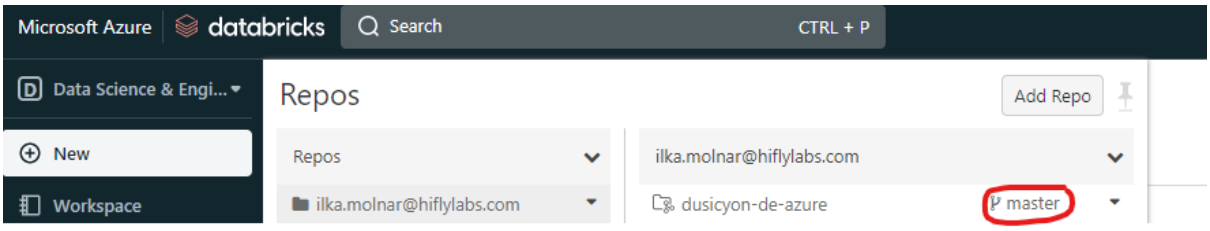
- set git integration in databricks workspace user settings



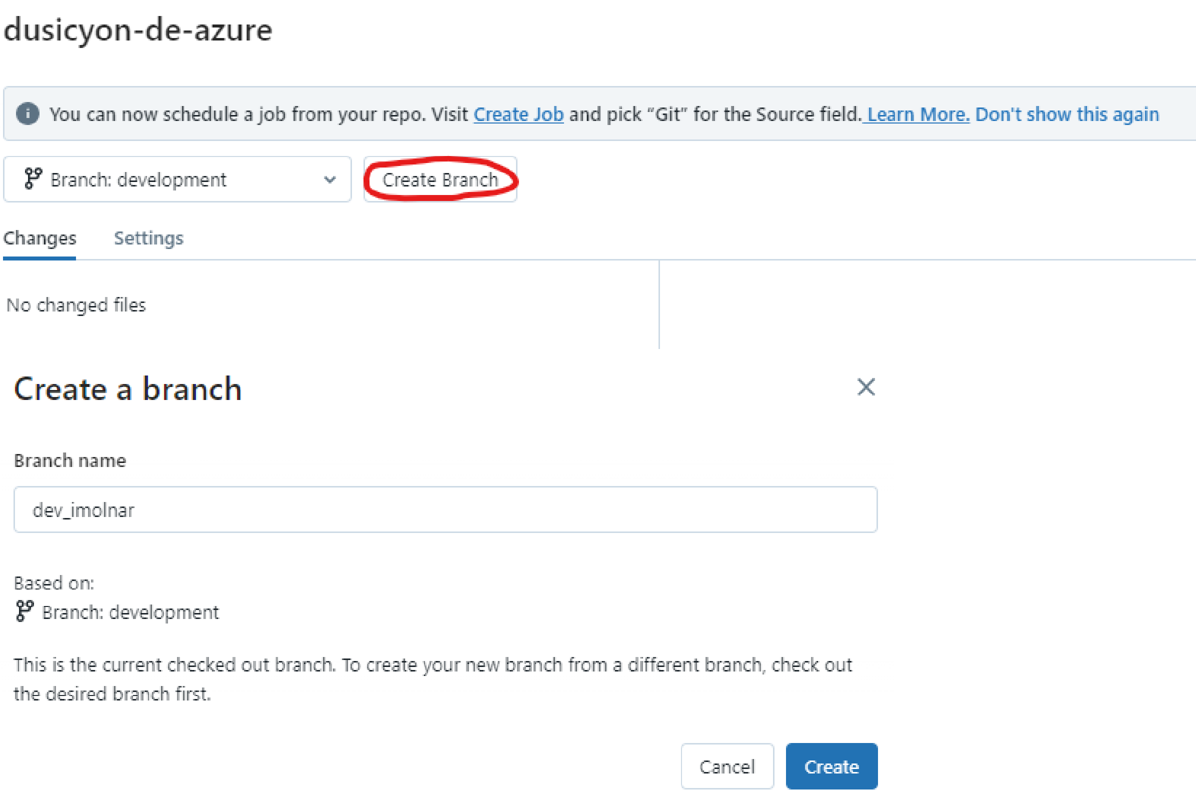
- add repo


Microsoft Azure
databricks
Search
CTRL + P
Data Science & Engi...
New
Workspace
Repos
Recents
Data
Compute
Workflows
Repos
Add Repo
Repos
Learn more
ilka.molnar@hiflylabs.com
Add folder
Data import
Quickly import data, preview its schema, create a table, and query it in a notebook.
Upload data
Transform data
Delta Live Tables
dbt Core


Add Repo
Location
/Repos/ilka.molnar@hiflylabs.com
Create repo by cloning a Git repository
Git repository URL
https://github.com/green-fox-academy/dusicyon-de-azure
Git provider
GitHub
Repository name
dusicyon-de-azure
Advanced
Cancel
Create Repo

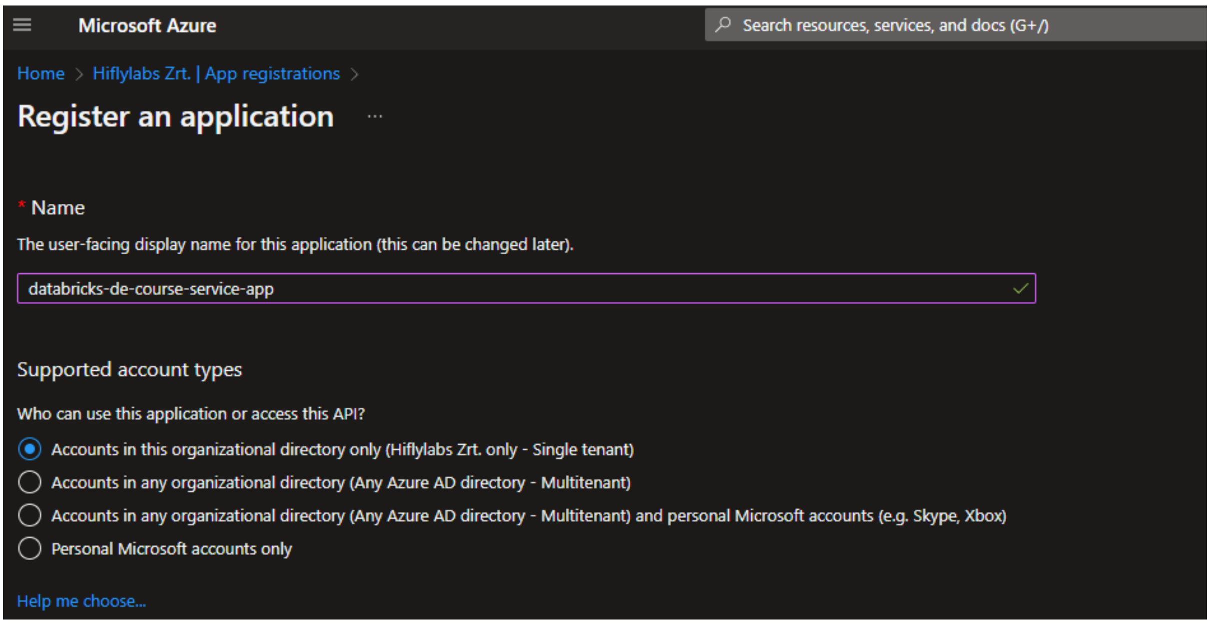

Microsoft Azure
databricks
Search
CTRL + P
Data Science & Engi...
New
Workspace
Repos
Add Repo
Repos
ilka.molnar@hiflylabs.com
ilka.molnar@hiflylabs.com
dusicyon-de-azure
master

- create your feature branch

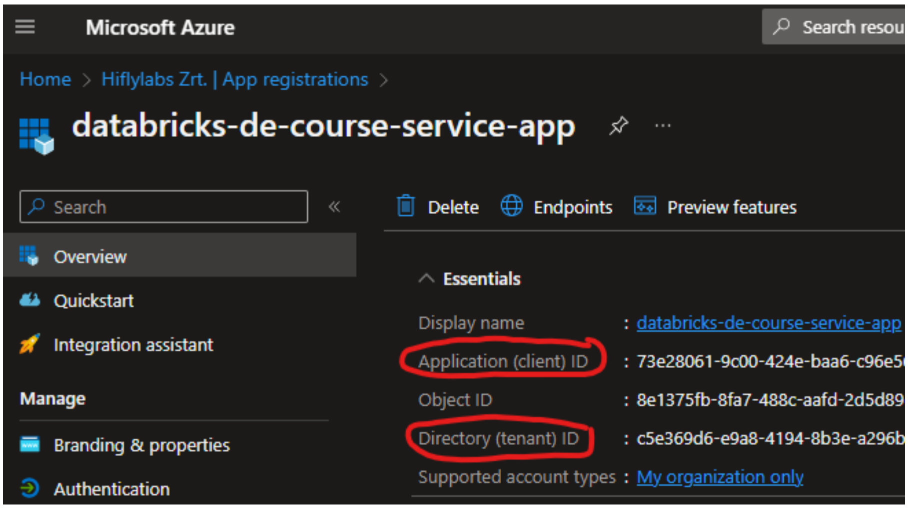


7.4 Storage account mounting

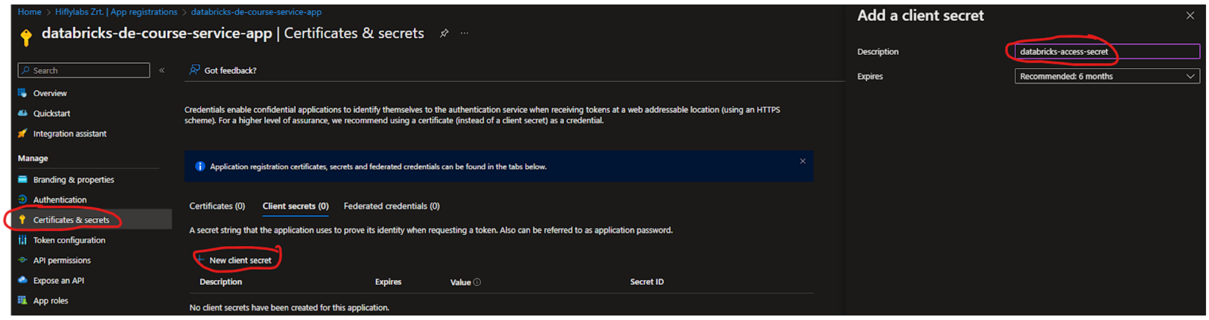
- Register an Azure AD application (Azure Active Directory/App registrations/New registration)



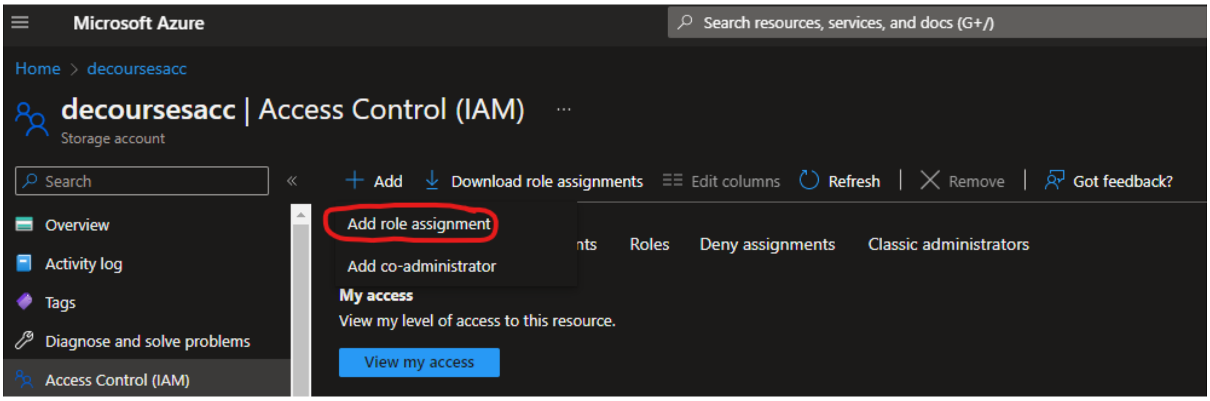
- copy the client id and the tenant id to notepad

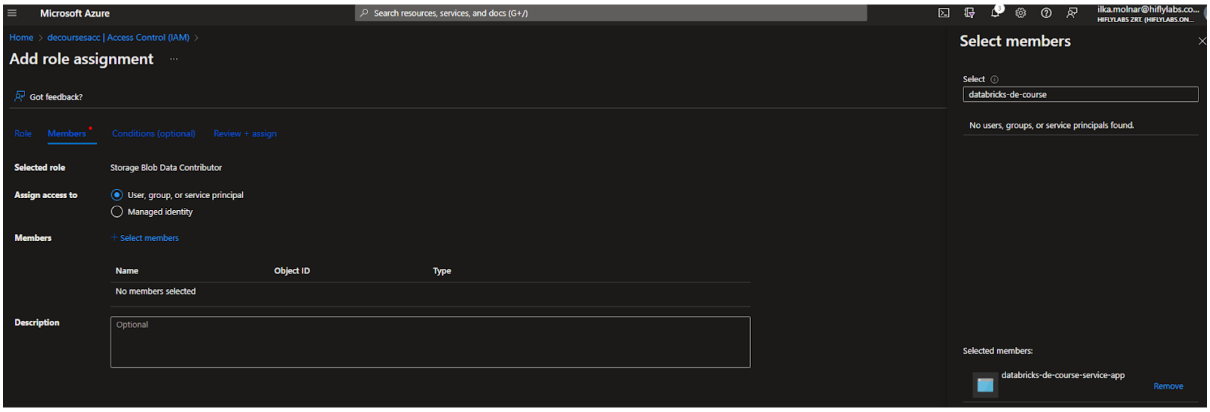


- generate an authentication key (new client secret) and copy the value to notepad

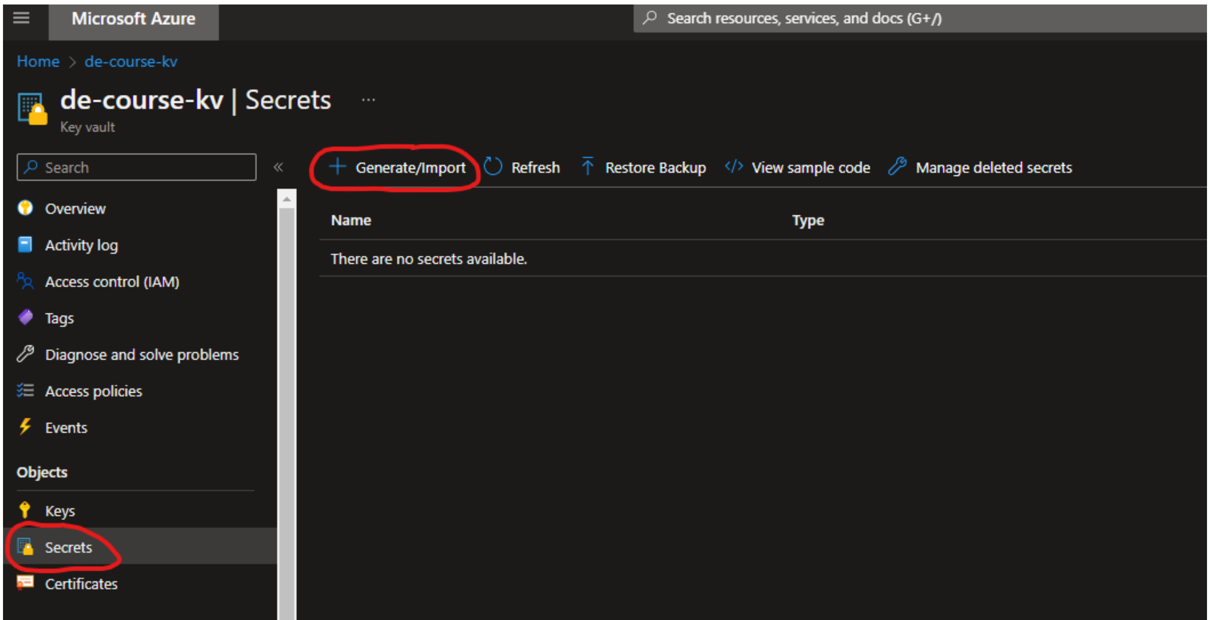


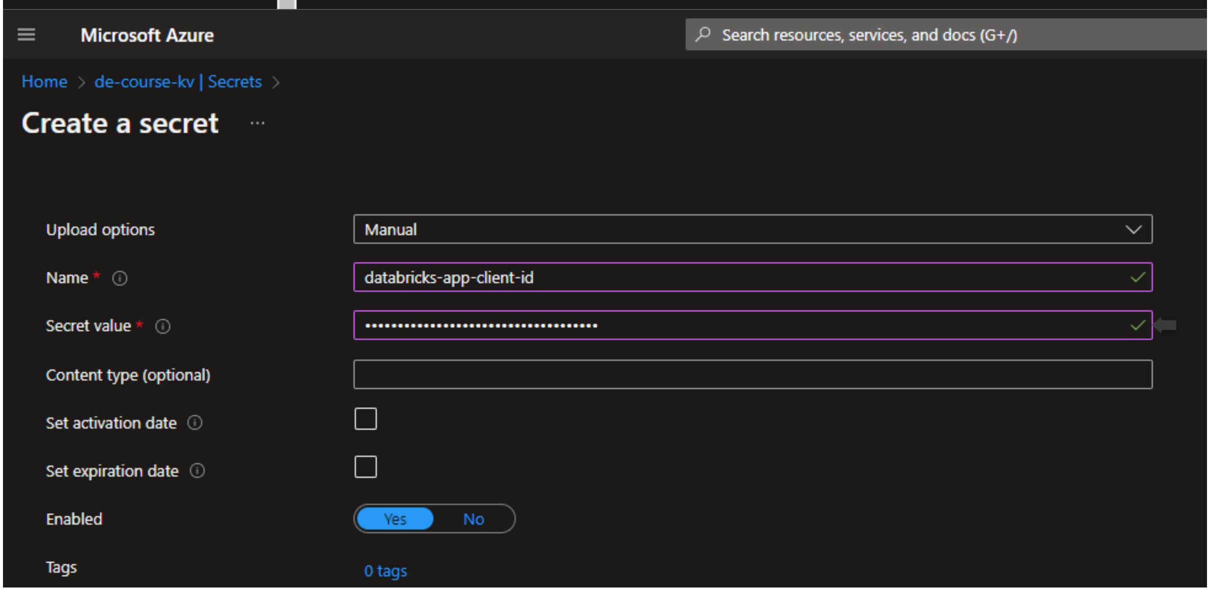
- Grant service principal access to ADLS account (grant storage blob data contributor role to your databricks service principal on your storage account)

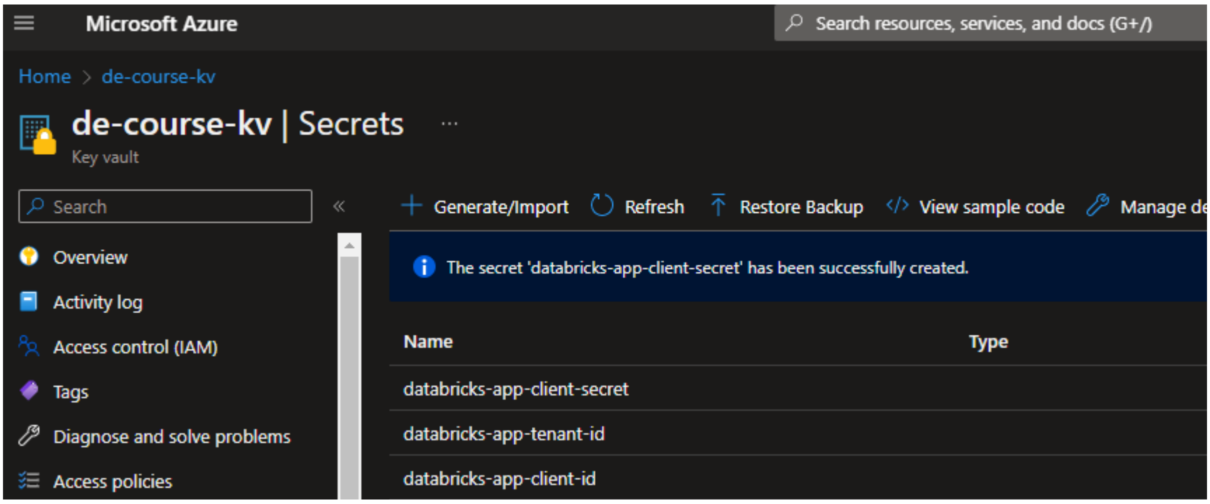




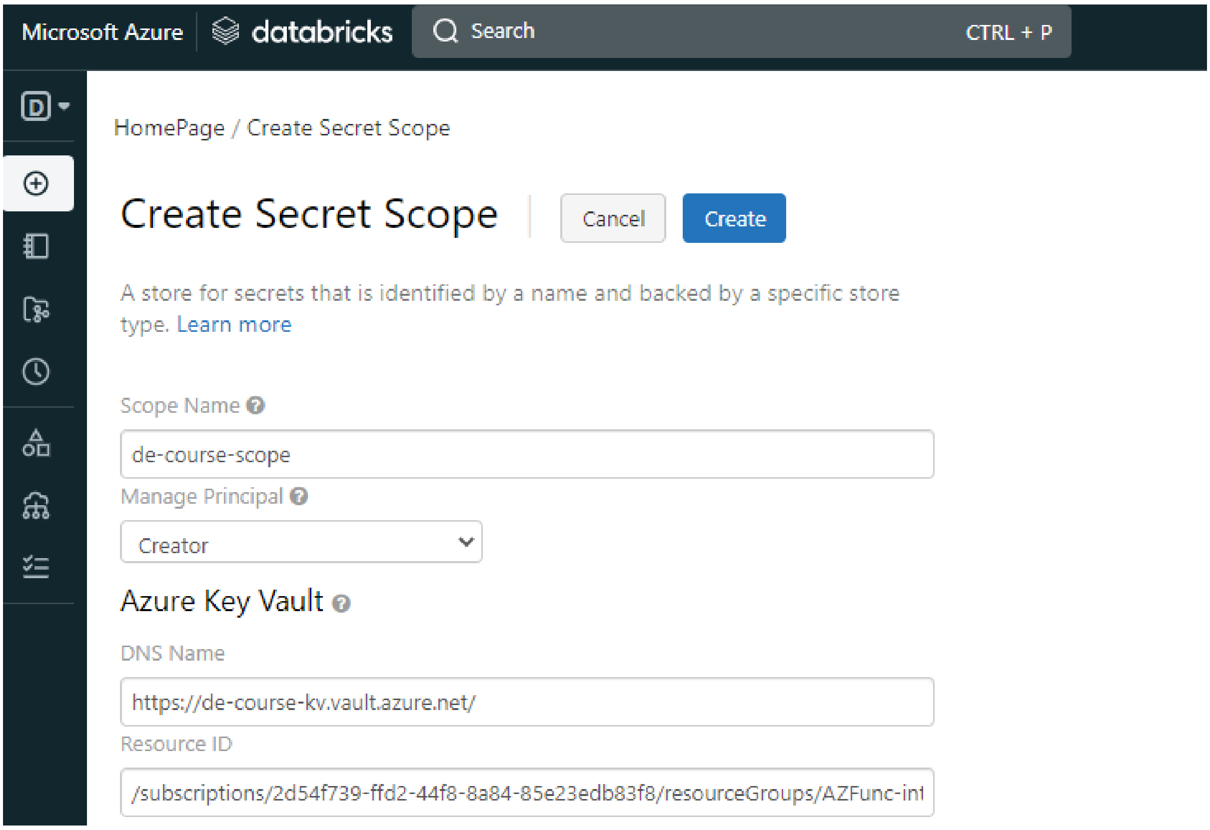
- Create an Azure Key Vault and add the application secret, the client id and the tenant id to it

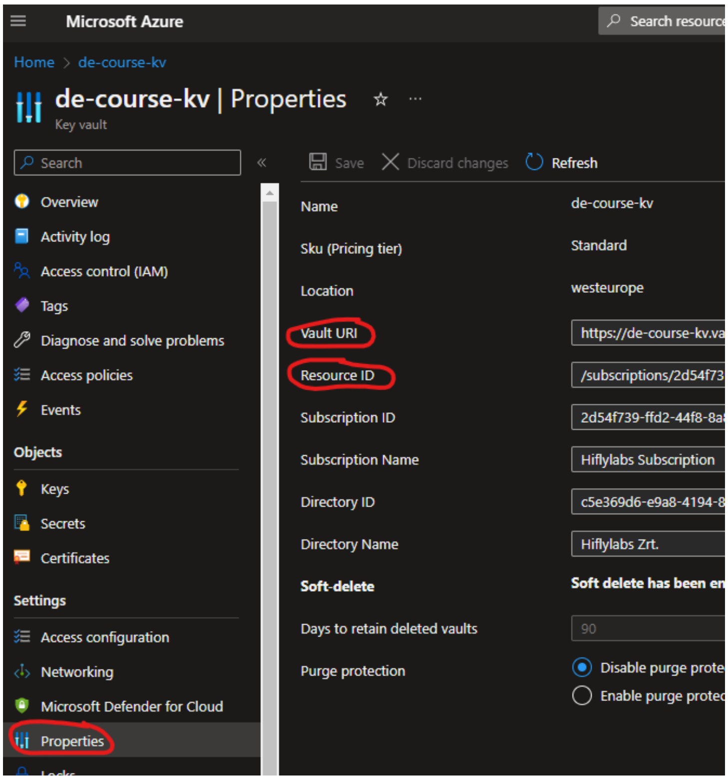




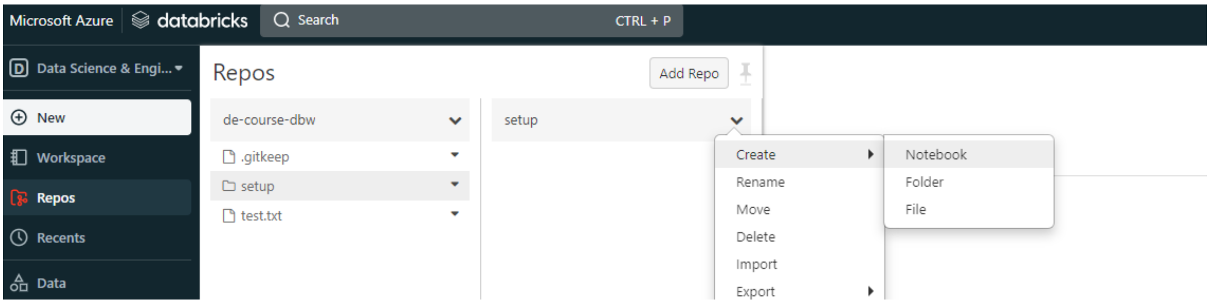


- Create an Azure Key Vault-backed Secret Scope in Azure Databricks (go to `https://<DATABRICKS-INSTANCE>#secrets/createScope` and replace <DATABRICKS-INSTANCE> with your actual Databricks instance URL)





- Mount ADLS to Databricks using Secret Scope



mount_adls_storage Python

File Edit View Run Help Last edit was 1 minute ago Give feedback

Cmd 1

```
storage_account_name = "decoursesacc"
client_id            = ...
tenant_id            = ...
client_secret        = ...
```

Command took 1.50 seconds -- by ilka.molnar@hiflylabs.com at 05/01/2023, 03:07:37 on unknown cluster

Cmd 2

```
configs = ...
```

Command took 0.05 seconds -- by ilka.molnar@hiflylabs.com at 05/01/2023, 03:07:40 on unknown cluster

Cmd 3

```
def mount_adls(container_name):
  dbutils.fs.mount(
    source = ...
    mount_point = ...
    extra_configs = ...)
```

Command took 0.10 seconds -- by ilka.molnar@hiflylabs.com at 05/01/2023, 03:07:41 on unknown cluster

Cmd 4

```
mount_adls("landing")
```

Command took 21.10 seconds -- by ilka.molnar@hiflylabs.com at 05/01/2023, 03:29:17 on unknown cluster

Cmd 5

```
...
```

Cmd 8

```
display(dbutils.fs.mounts())
```

(3) Spark Jobs

Table +

	mountPoint	source	encryptionType
3	/mnt/silver	abfss://silver@decoursesacc.dfs.core.windows.net	
4	/databricks/mlflow-tracking	databricks/mlflow-tracking	
5	/databricks-results	databricks-results	
6	/databricks/mlflow-registry	databricks/mlflow-registry	
7	/mnt/bronze	abfss://bronze@decoursesacc.dfs.core.windows.net	
8	/mnt/landing	abfss://landing@decoursesacc.dfs.core.windows.net	
9	/	DatabricksRoot	

Showing all 9 rows. | 8.41 seconds runtime

Command took 8.41 seconds -- by ilka.molnar@hiflylabs.com at 02/01/2023, 01:19:06 on singlenode

Cmd 9

```
dbutils.fs.unmount("/mnt/landing")
```