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):

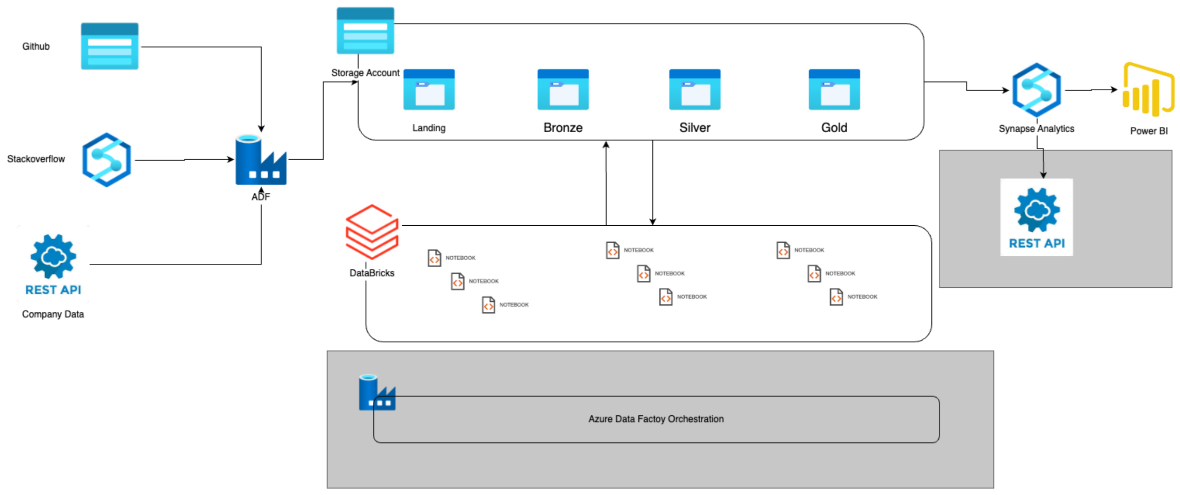
- Github
- StackOverflow

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.

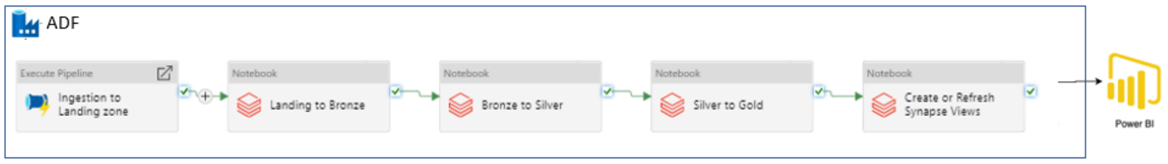
2. Architecture



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

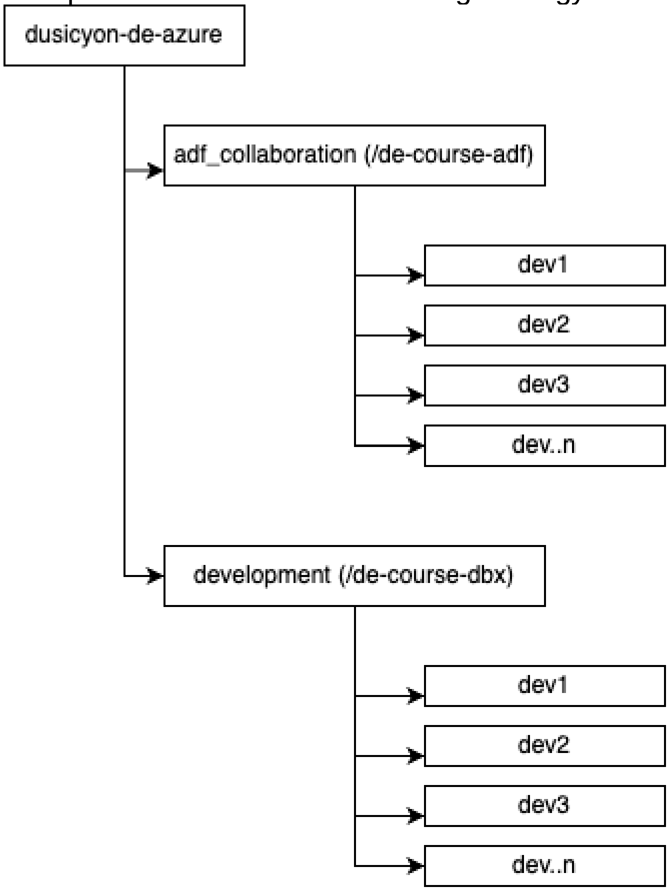
- **landing layer**: target of the ingestion pipelines, data is untyped, untransformed here, contains source data "as-is" (loaded with an ADF copy activity)
- **bronze/raw layer**: source data with minimal transformations but already in delta format (loaded with Databricks Notebooks called from an ADF 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 branch have been already created. This picture illustrates the branching strategy of the project.


dusicyon-de-azure
adf_collaboration (/de-course-adf)
dev1
dev2
dev3
dev..n
development (/de-course-dbx)
dev1
dev2
dev3
dev..n

4. Ingestion (Landing layer)

There are some suggestion 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 provide 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

Expected result
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+AStDN7ulw==
    - 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 this 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

4.3 Materials

4.3.1 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

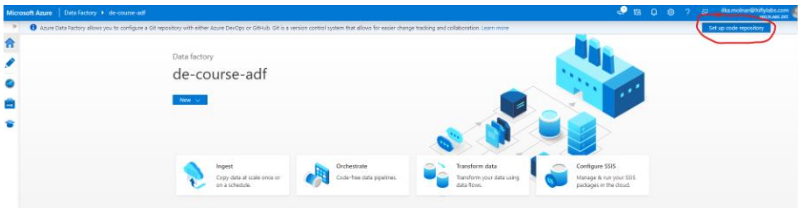
Future modification options:
- We create only one environment at start (for testing and production purpose we should create separate storage accounts)
- In the dev storage account it is also a possible approach to create separate containers for the developers (in that case the bronze/silver/gold layers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

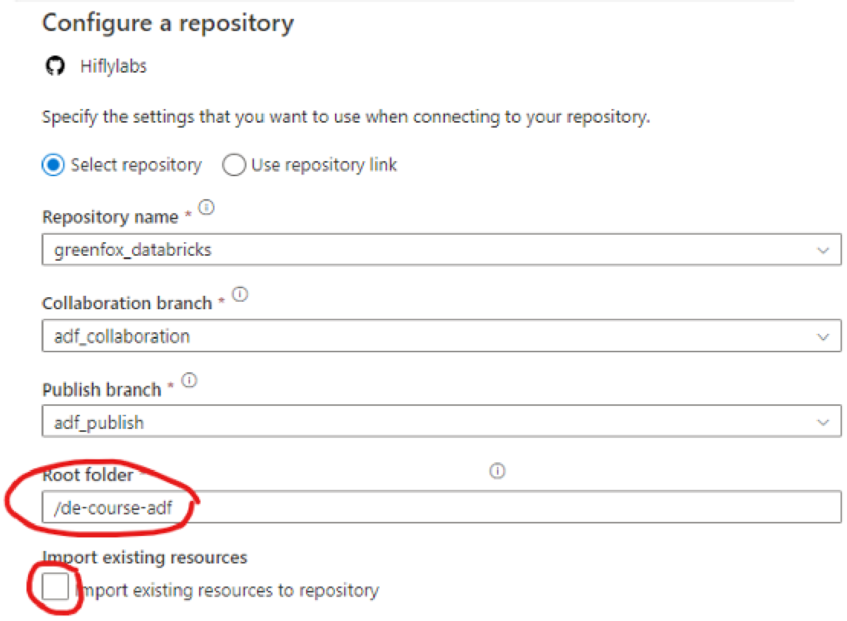
4.3.2 Azure Data Factory

No special configuration is needed.

4.3.2.1 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!





Configure a repository

Hiflylabs

Specify the settings that you want to use when connecting to your repository.

[✓ Select repository] Use repository link

Repository name *: greenfox_databricks

Collaboration branch *: adf_collaboration

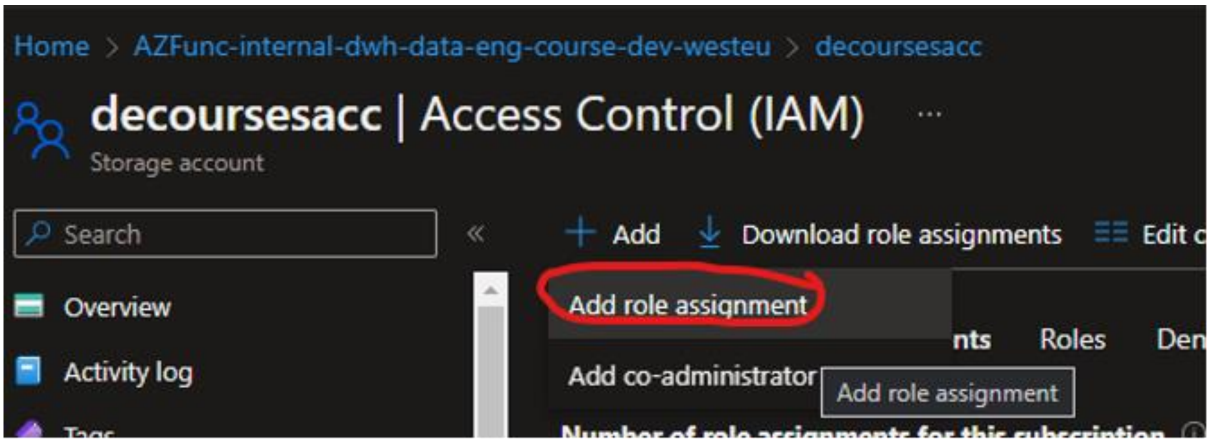
Publish branch *: adf_publish

Root folder: /de-course-adf

Import existing resources

☐ Import existing resources to repository

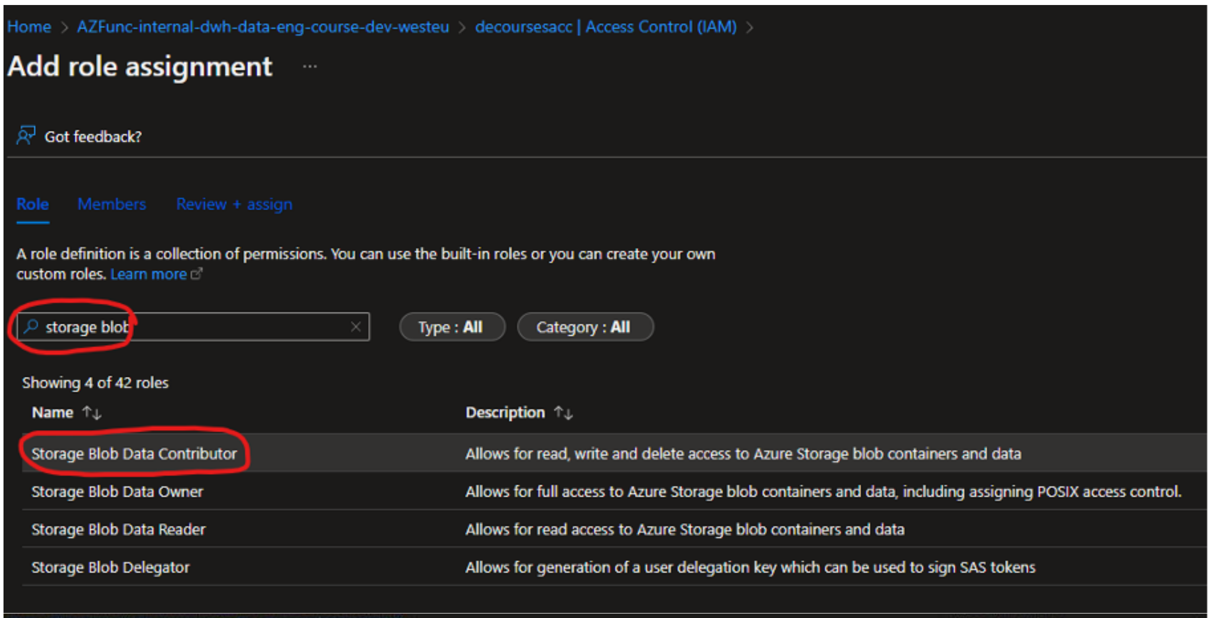
4.3.2.2 BlobStorage contributor access



Home > AZFunc-internal-dwh-data-eng-course-dev-westeu > decoursesacc

decoursesacc | Access Control (IAM) ...

Storage account



Home > AZFunc-internal-dwh-data-eng-course-dev-westeu > decoursesacc | Access Control (IAM) >

Add role assignment ...

Got feedback?

Role | Members | Review + assign

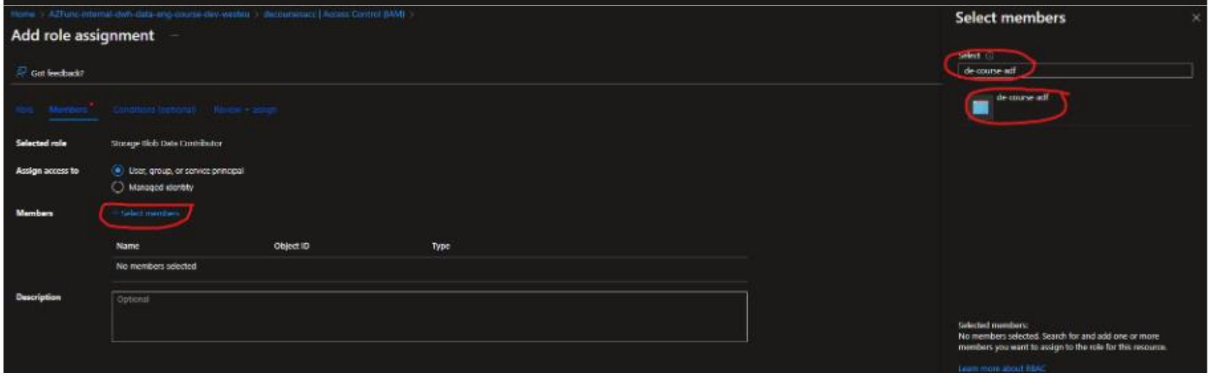
A role definition is a collection of permissions. You can use the built-in roles or you can create your own custom roles. Learn more

storage blob

Type : **All** | Category : **All**

Showing 4 of 42 roles

Name	**Description**
Storage Blob Data Contributor	Allows for read, write and delete access to Azure Storage blob containers and data
Storage Blob Data Owner	Allows for full access to Azure Storage blob containers and data, including assigning POSIX access control.
Storage Blob Data Reader	Allows for read access to Azure Storage blob containers and data
Storage Blob Delegator	Allows for generation of a user delegation key which can be used to sign SAS tokens



Home > AZFunc-internal-dwh-data-eng-course-dev-westeu > decoursesacc | Access Control (IAM) >

Add role assignment

Got feedback?

Role | Members * | Conditions (optional) | Review + assign

Selected role: Storage Blob Data Contributor

Assign access to: [✓ User, group, or service principal] Managed identity

Members: + Select members

Name	Object ID	Type
No members selected		

Description: Optional

Select members

Select: de-course-adf

de-course-adf

Selected members:
No members selected. Search for and add one or more members you want to assign to the role for this resource.

Learn more about RBAC

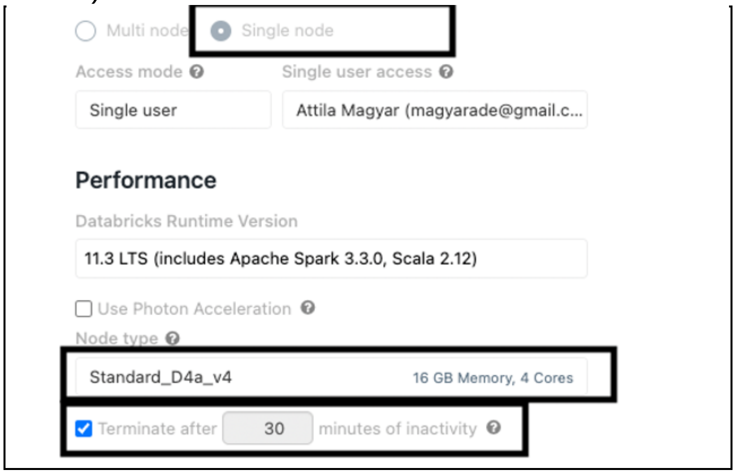
5. Bronze layer

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)



◯ Multi node ● Single node

Access mode: Single user
Single user access: Attila Magyar (magyarade@gmail.c...

Performance

Databricks Runtime Version: 11.3 LTS (includes Apache Spark 3.3.0, Scala 2.12)

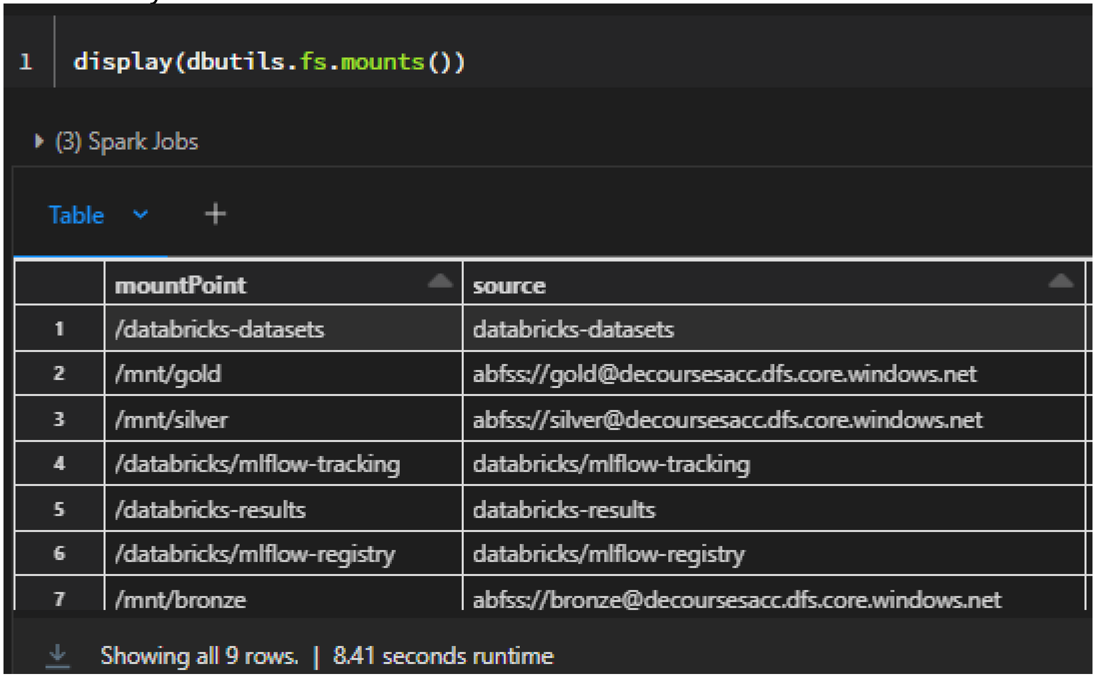
☐ Use Photon Acceleration

Node type: Standard_D4a_v4 16 GB Memory, 4 Cores

☑ Terminate after 30 minutes of inactivity

- 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_your-name) 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

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

  - create a widget for the file date parameter

```
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)

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

  - read the data

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

  - transform the data

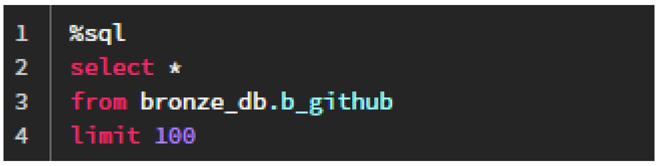
```
transf_df = df.withColumn("_pk",col("id")) \
...
```

  - 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

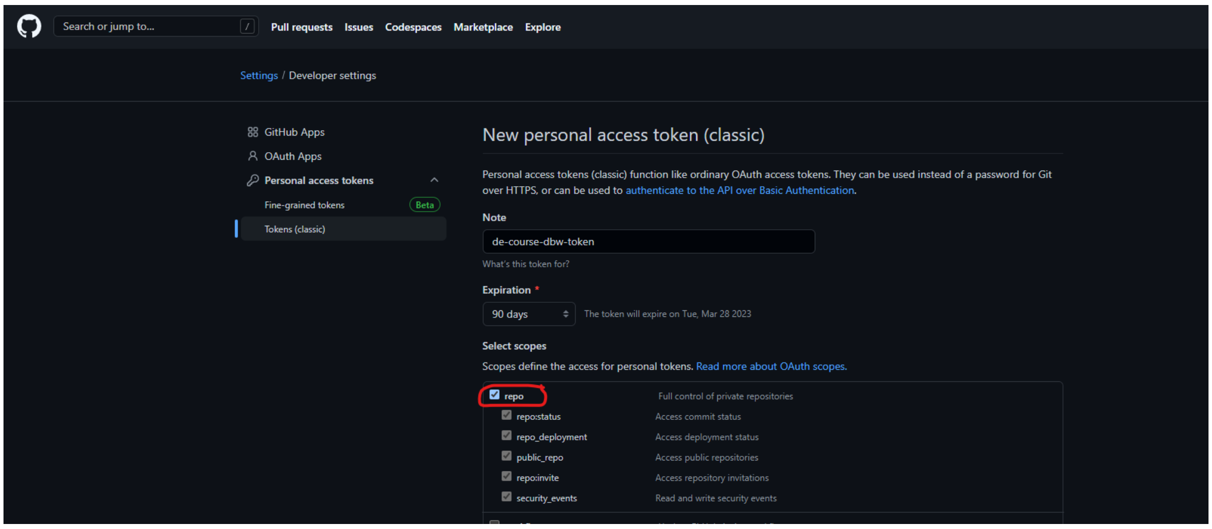
5.2 Optional tasks

- 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

5.3 Materials

5.3.1 Databricks git integration

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



Settings / Developer settings

New personal access token (classic)

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

Note: de-course-dbw-token

What's this token for?

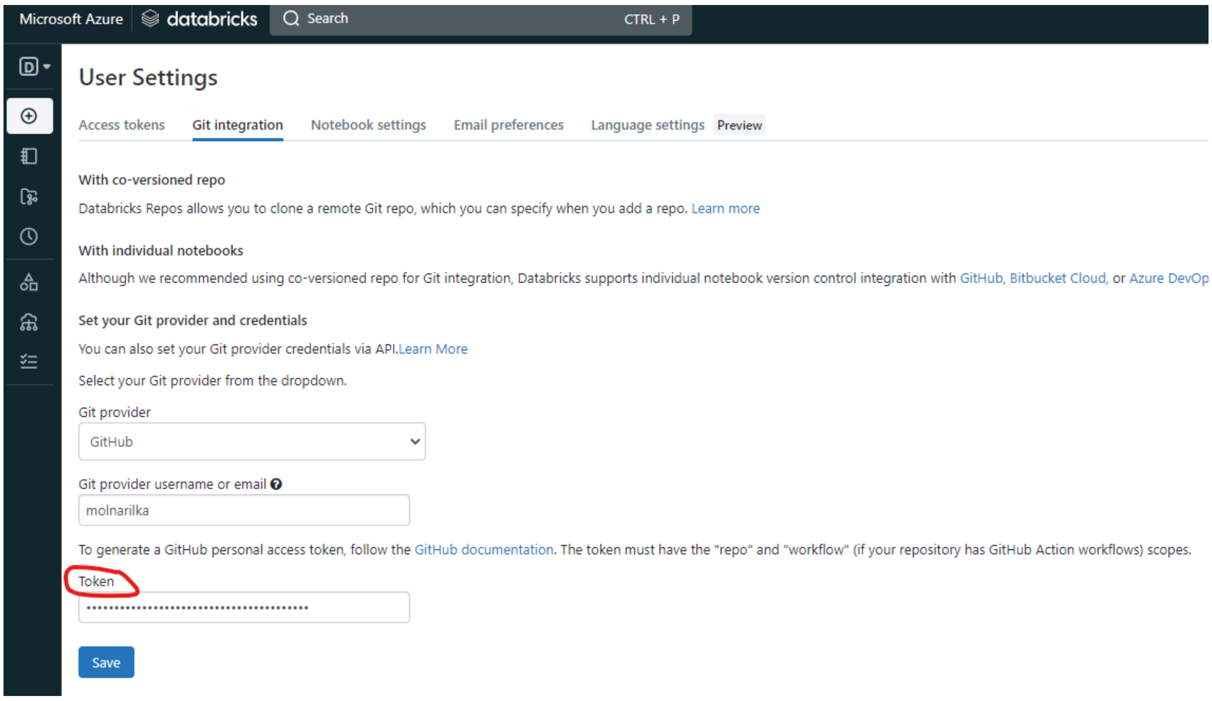
Expiration *: 90 days — The token will expire on Tue, Mar 28 2023

Select scopes

Scopes define the access for personal tokens. Read more about OAuth scopes.

✓ repo	Full control of private repositories
✓ repo:status	Access commit status
✓ repo_deployment	Access deployment status
✓ public_repo	Access public repositories
✓ repo:invite	Access repository invitations
✓ security_events	Read and write security events

- set git integration in databricks workspace user settings



User Settings

Access tokens | Git integration | Notebook settings | Email preferences | Language settings Preview

With co-versioned repo

Databricks Repos allows you to clone a remote Git repo, which you can specify when you add a repo. Learn more

With individual notebooks

Although we recommended using co-versioned repo for Git integration, Databricks supports individual notebook version control integration with GitHub, Bitbucket Cloud, or Azure DevOp

Set your Git provider and credentials

You can also set your Git provider credentials via API.Learn More

Select your Git provider from the dropdown.

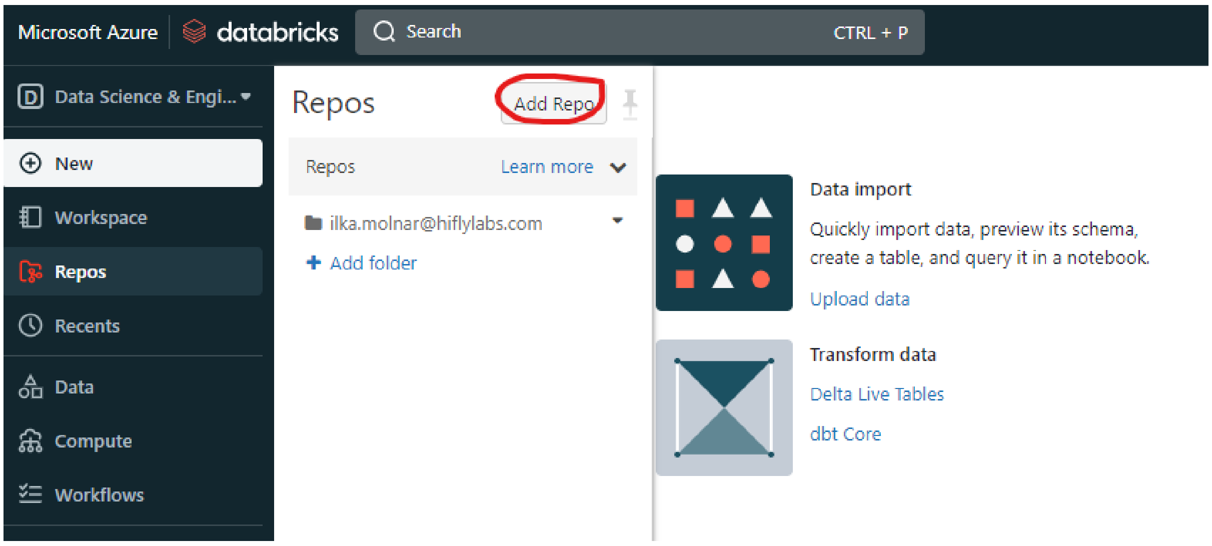
Git provider: GitHub

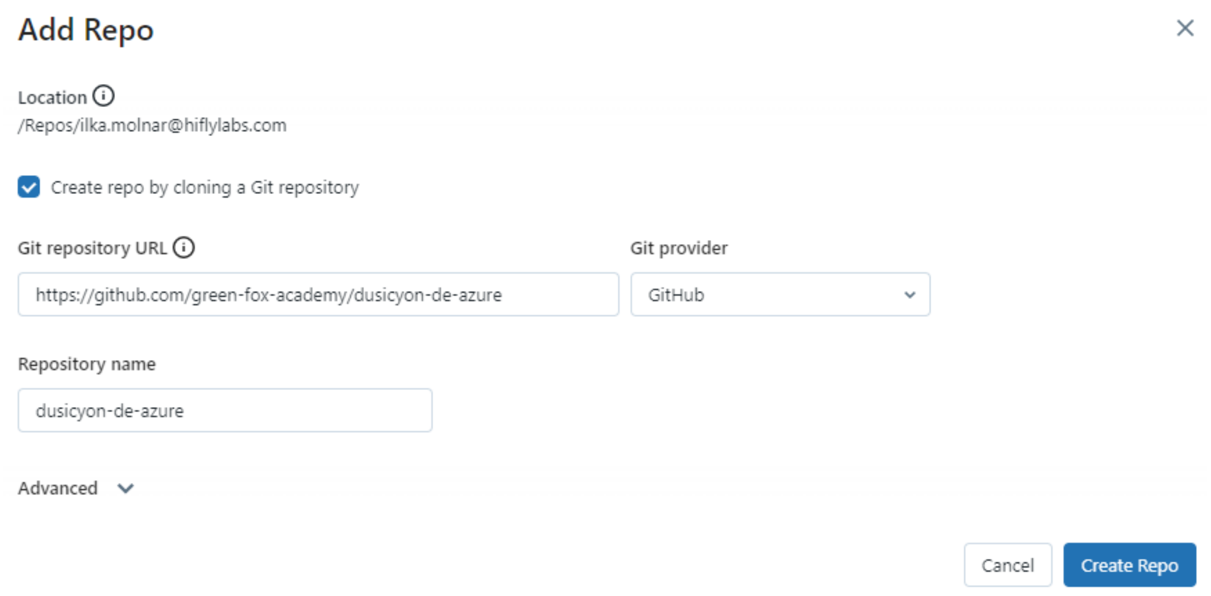
Git provider username or email: molnarilka

To generate a GitHub personal access token, follow the GitHub documentation. The token must have the "repo" and "workflow" (if your repository has GitHub Action workflows) scopes.

Token: ••

- add repo




Add Repo

Location
/Repos/ilka.molnar@hiflylabs.com

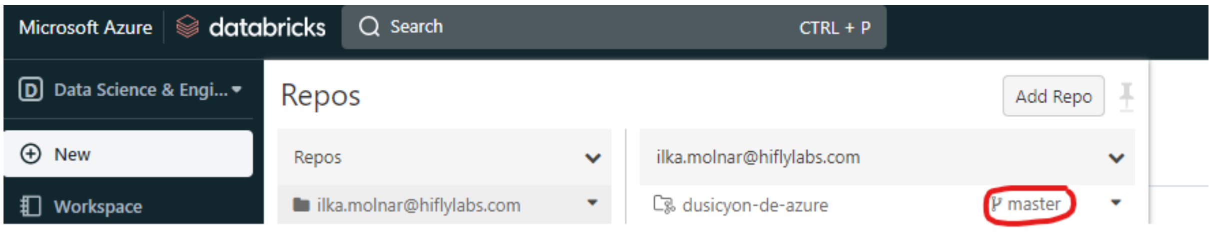
☑ Create repo by cloning a Git repository

Git repository URL	Git provider
https://github.com/green-fox-academy/dusicyon-de-azure	GitHub

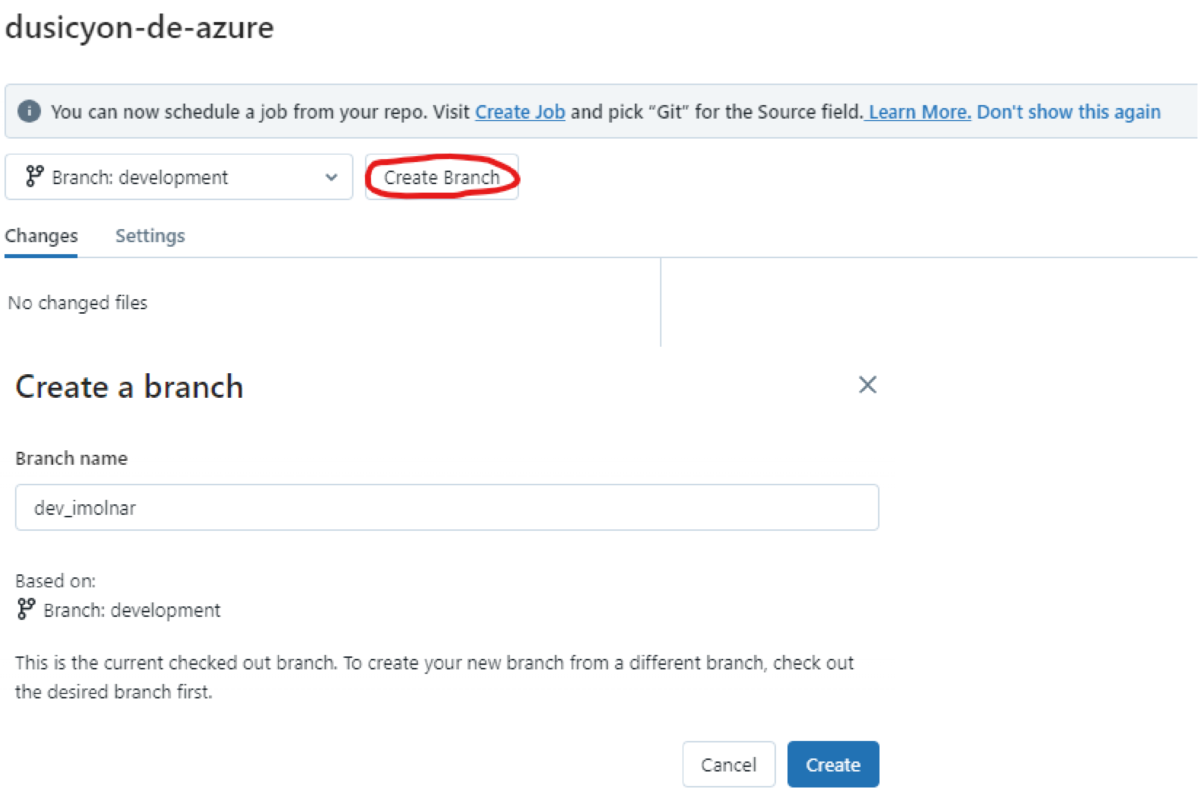
Repository name: dusicyon-de-azure

Advanced

Cancel   Create Repo



- create your feature branch


dusicyon-de-azure

You can now schedule a job from your repo. Visit Create Job and pick "Git" for the Source field. Learn More. Don't show this again

Branch: development | Create Branch

Changes | Settings

No changed files

Create a branch

Branch name

dev_imolnar

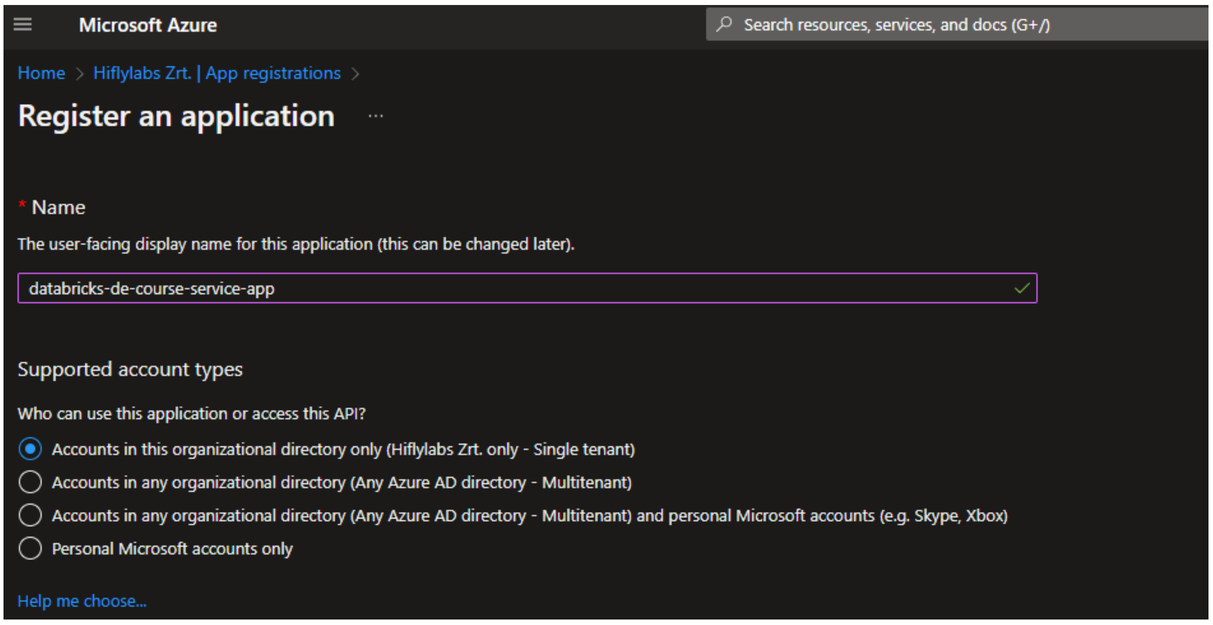
Based on:
Branch: development

This is the current checked out branch. To create your new branch from a different branch, check out the desired branch first.

Cancel | Create

5.3.2 Storage account mounting

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


Microsoft Azure | Search resources, services, and docs (G+/)

Home > Hiflylabs Zrt. | App registrations >

Register an application

* Name

The user-facing display name for this application (this can be changed later).

databricks-de-course-service-app

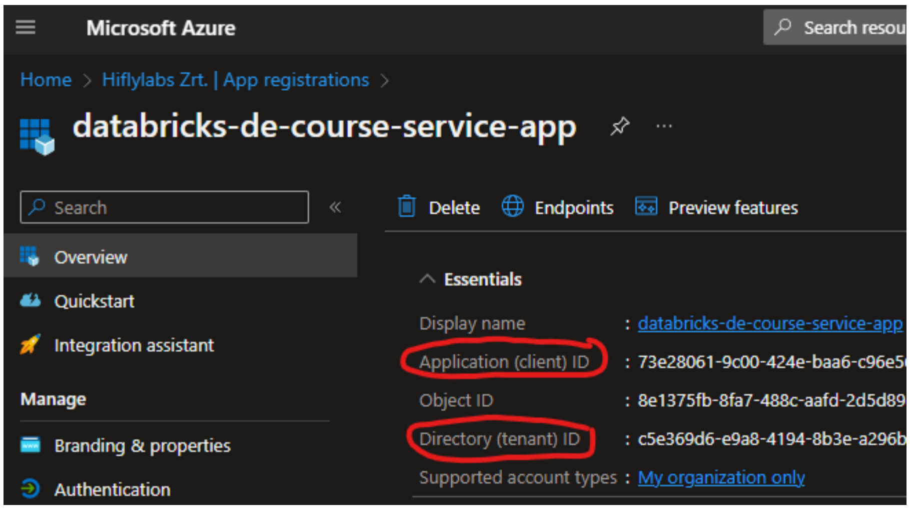
Supported account types

Who can use this application or access this API?

- [✓ Accounts in this organizational directory only (Hiflylabs Zrt. only - Single tenant)]
- Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- Personal Microsoft accounts only

Help me choose...

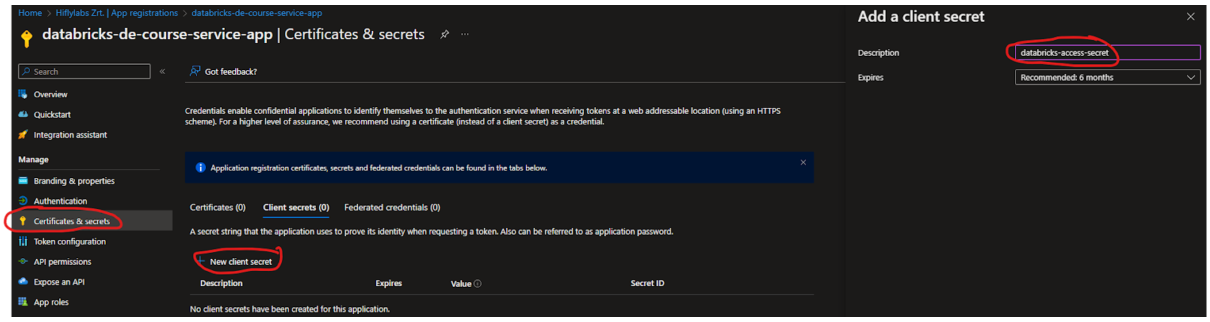
- copy the client id and the tenant id to notepad



databricks-de-course-service-app

Display name : databricks-de-course-service-app
Application (client) ID : 73e28061-9c00-424e-baa6-c96e5
Object ID : 8e1375fb-8fa7-488c-aafd-2d5d89
Directory (tenant) ID : c5e369d6-e9a8-4194-8b3e-a296b
Supported account types : My organization only

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



databricks-de-course-service-app | Certificates & secrets

Credentials enable confidential applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTPS scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Application registration certificates, secrets and federated credentials can be found in the tabs below.

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

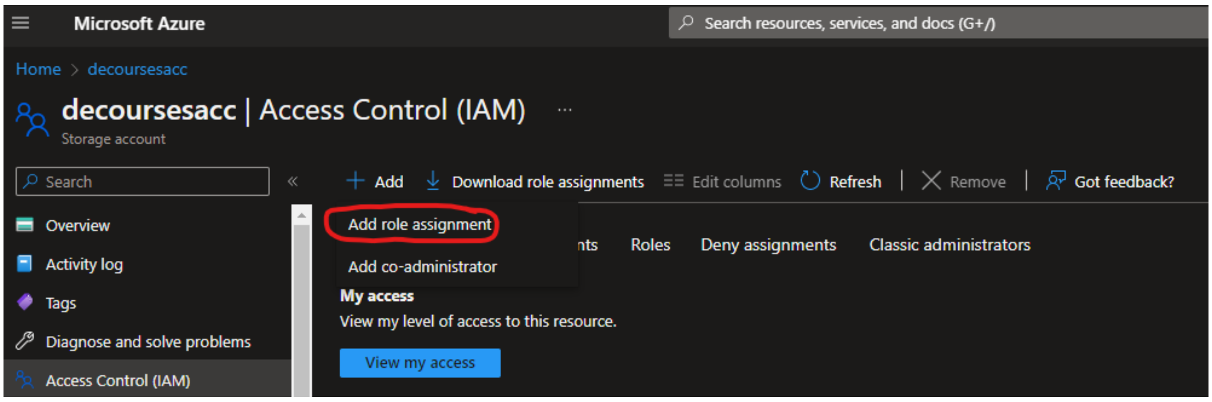
Description	Expires	Value	Secret ID

No client secrets have been created for this application.

Add a client secret

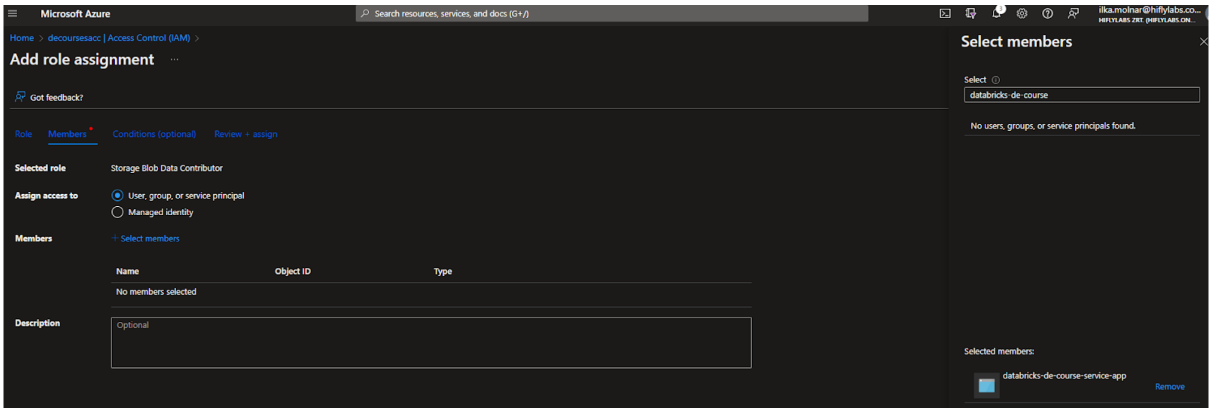
Description: databricks-access-secret
Expires: Recommended: 6 months

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



decoursesacc | Access Control (IAM)
Storage account

My access
View my level of access to this resource.



Home > decoursesacc | Access Control (IAM) >

Add role assignment

Selected role: Storage Blob Data Contributor

Assign access to: (●) User, group, or service principal ( ) Managed identity

Members: + Select members

Name	Object ID	Type
No members selected		

Description: Optional

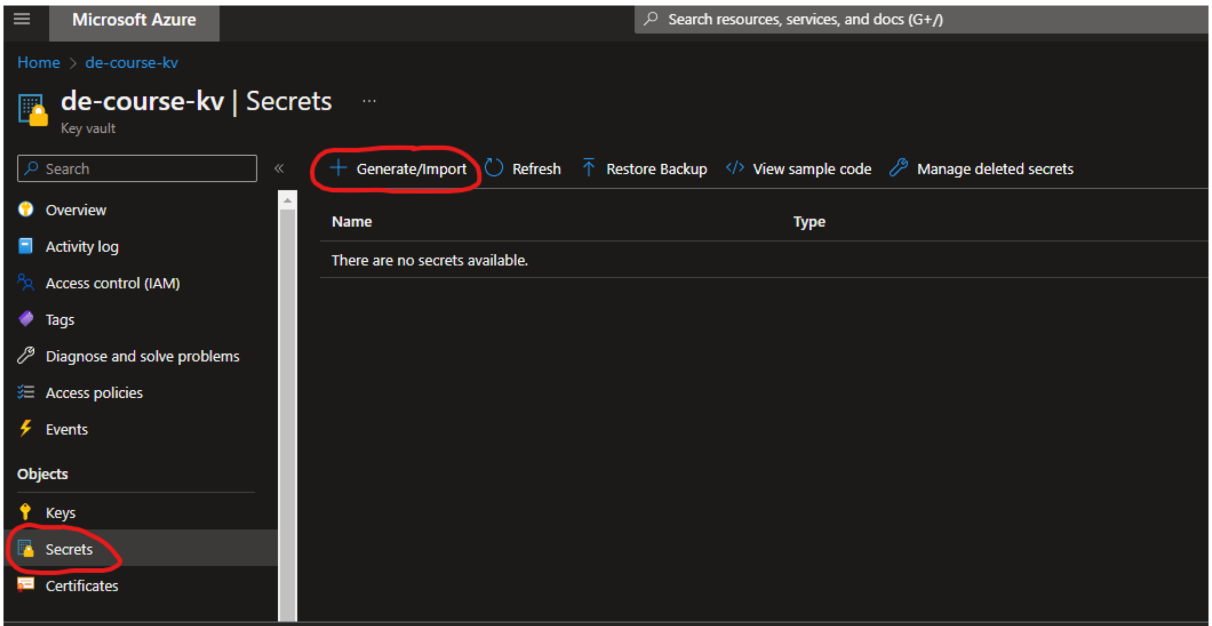
Select members

Select: databricks-de-course

No users, groups, or service principals found.

Selected members: databricks-de-course-service-app

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

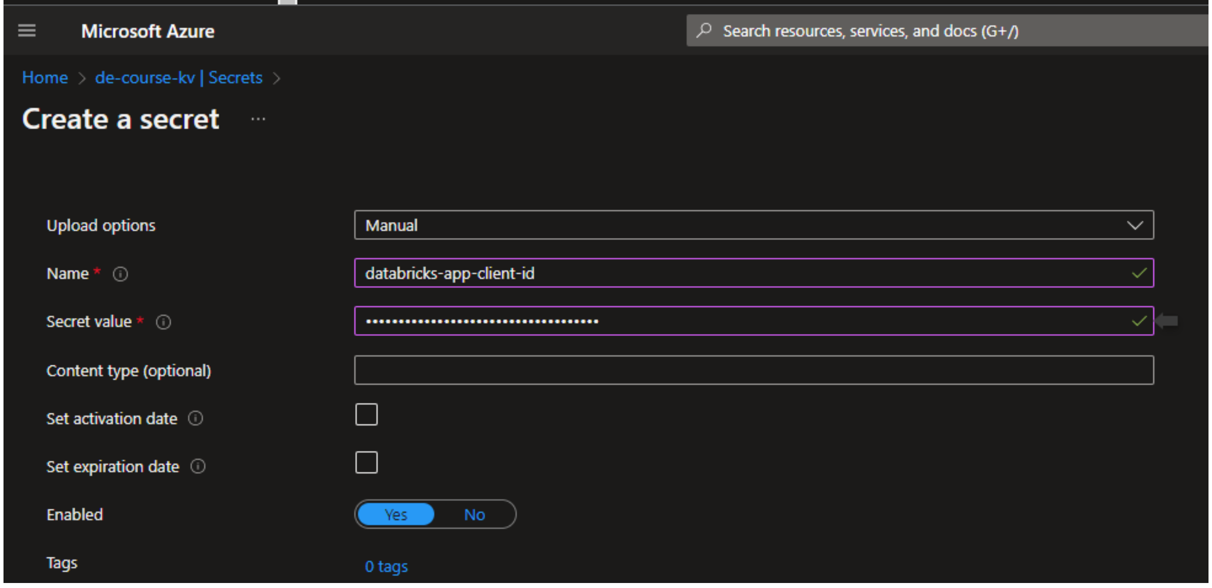


Home > de-course-kv

de-course-kv | Secrets

Key vault

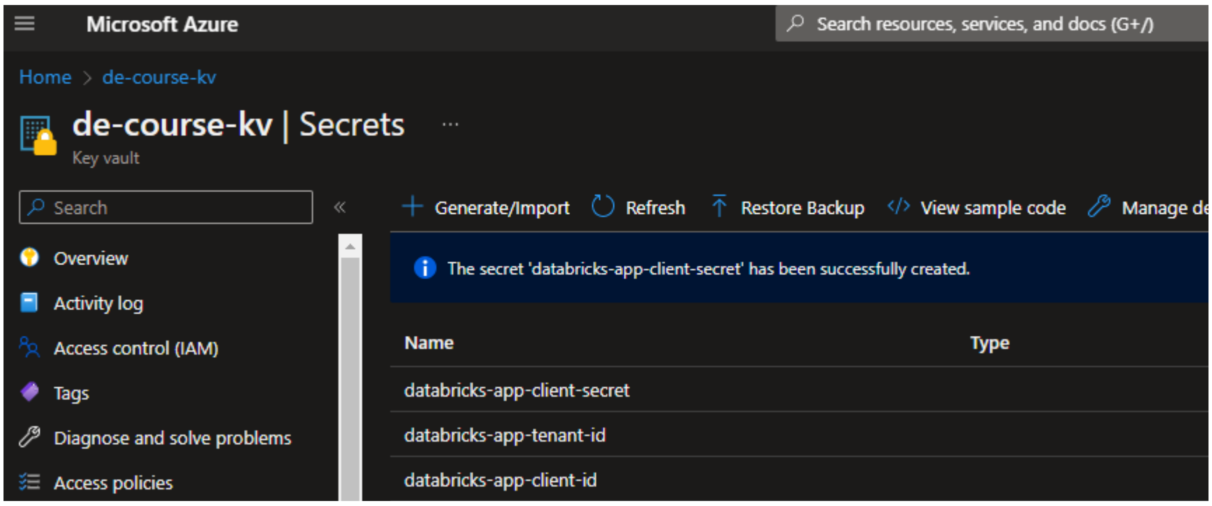
Name	Type
There are no secrets available.	



Home > de-course-kv | Secrets >

Create a secret

Upload options	Manual
Name *	databricks-app-client-id
Secret value *	••••••••••••••••••••••••••••••••••••••
Content type (optional)	
Set activation date	☐
Set expiration date	☐
Enabled	[✓ Yes] No
Tags	0 tags


Microsoft Azure | Search resources, services, and docs (G+/)

Home > de-course-kv

de-course-kv | Secrets

Key vault

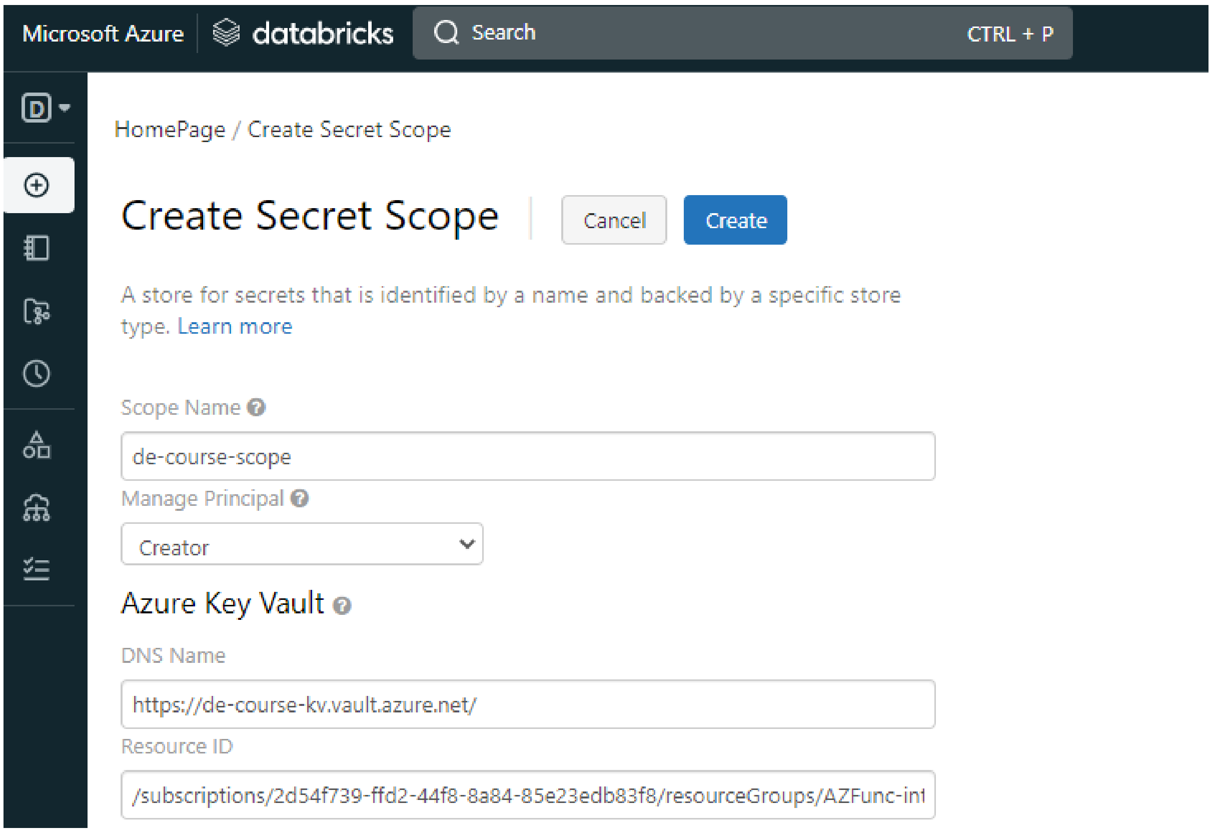
Search | Overview | Activity log | Access control (IAM) | Tags | Diagnose and solve problems | Access policies

Generate/Import | Refresh | Restore Backup | View sample code | Manage de

The secret 'databricks-app-client-secret' has been successfully created.

Name	Type
databricks-app-client-secret	
databricks-app-tenant-id	
databricks-app-client-id	

- 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)


Microsoft Azure | databricks | Search CTRL + P

HomePage / Create Secret Scope

Create Secret Scope | Cancel | Create

A store for secrets that is identified by a name and backed by a specific store type. Learn more

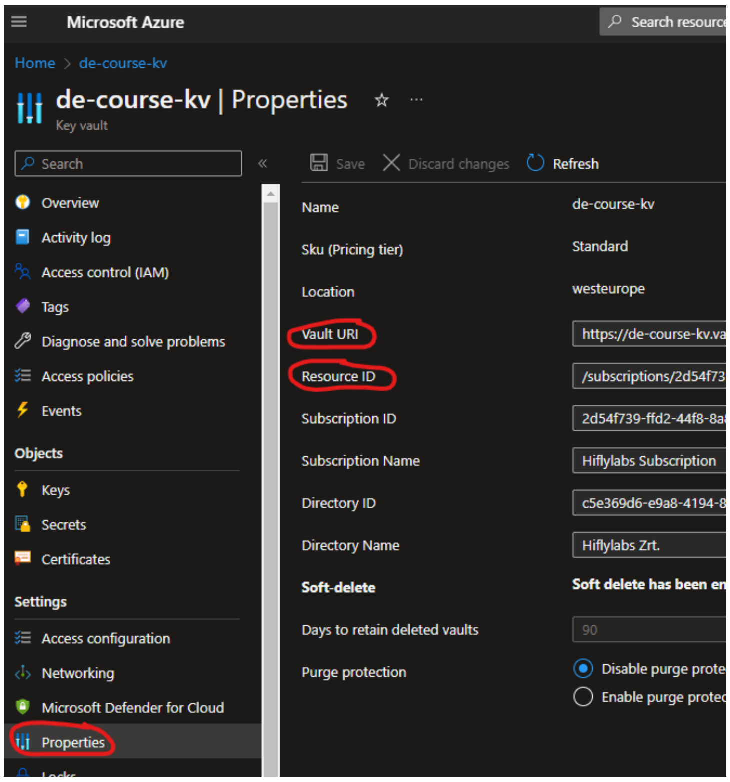
Scope Name: de-course-scope

Manage Principal: Creator

Azure Key Vault

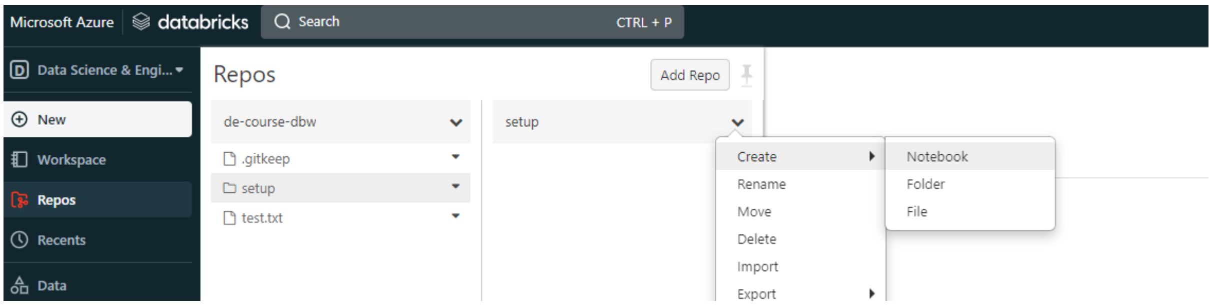
DNS Name: https://de-course-kv.vault.azure.net/

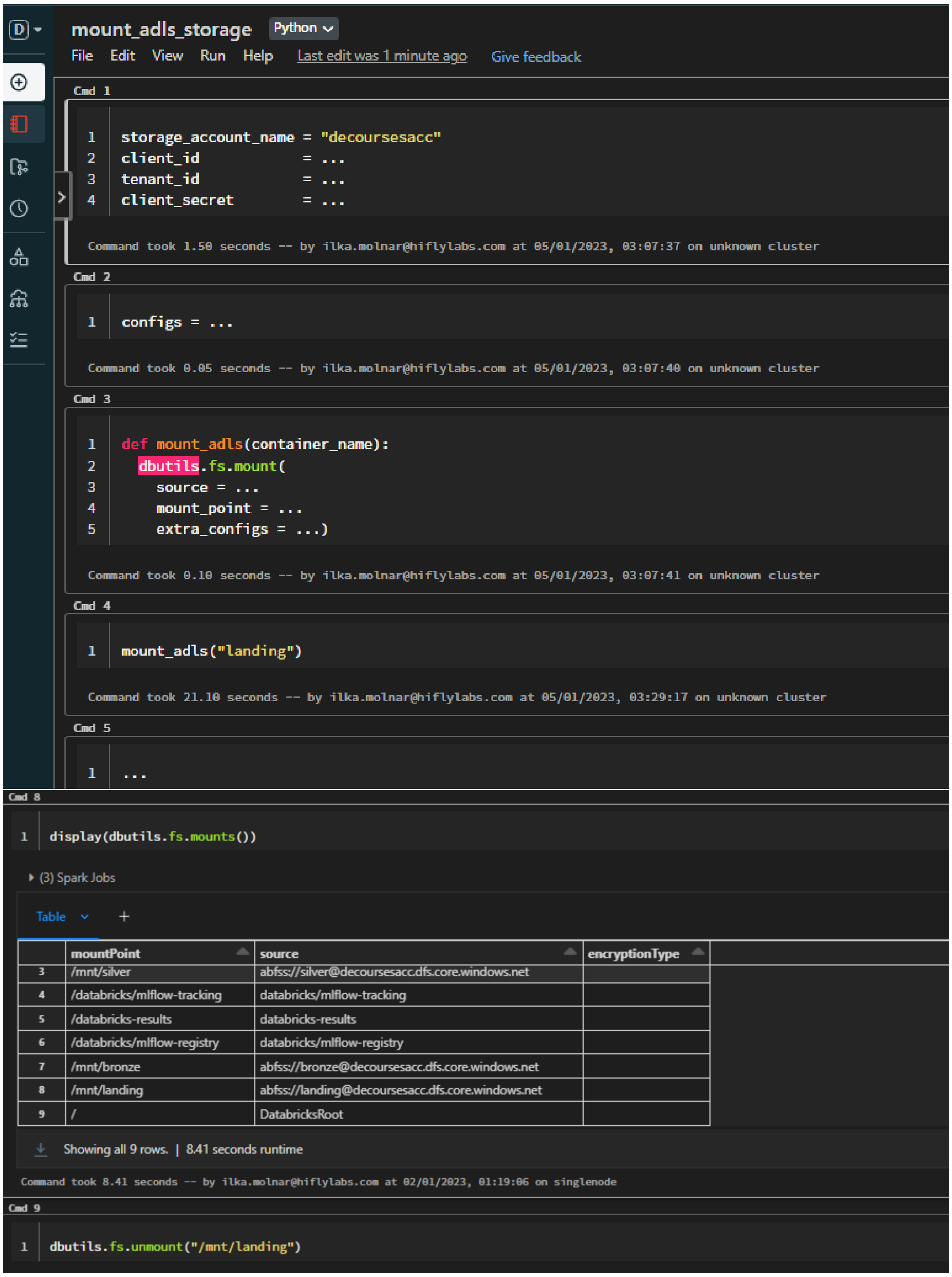
Resource ID: /subscriptions/2d54f739-ffd2-44f8-8a84-85e23edb83f8/resourceGroups/AZFunc-int



Name	de-course-kv
Sku (Pricing tier)	Standard
Location	westeurope
Vault URI	https://de-course-kv.va
Resource ID	/subscriptions/2d54f73
Subscription ID	2d54f739-ffd2-44f8-8a
Subscription Name	Hiflylabs Subscription
Directory ID	c5e369d6-e9a8-4194-8
Directory Name	Hiflylabs Zrt.
Soft-delete	**Soft delete has been en**
Days to retain deleted vaults	90
Purge protection	[✓ Disable purge prote] Enable purge protec

- Mount ADLS to Databricks using Secret Scope





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")
```