

Business Concept	1
Architecture	2
Branching strategy	3
Ingestion (Landing layer)	3
List of tasks	4
Optional tasks	6
Materials	7
Storage Account	7
Azure Data Factory	7
Azure Data Factory git integration	8
BlobStorage contributor access	8

## 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:

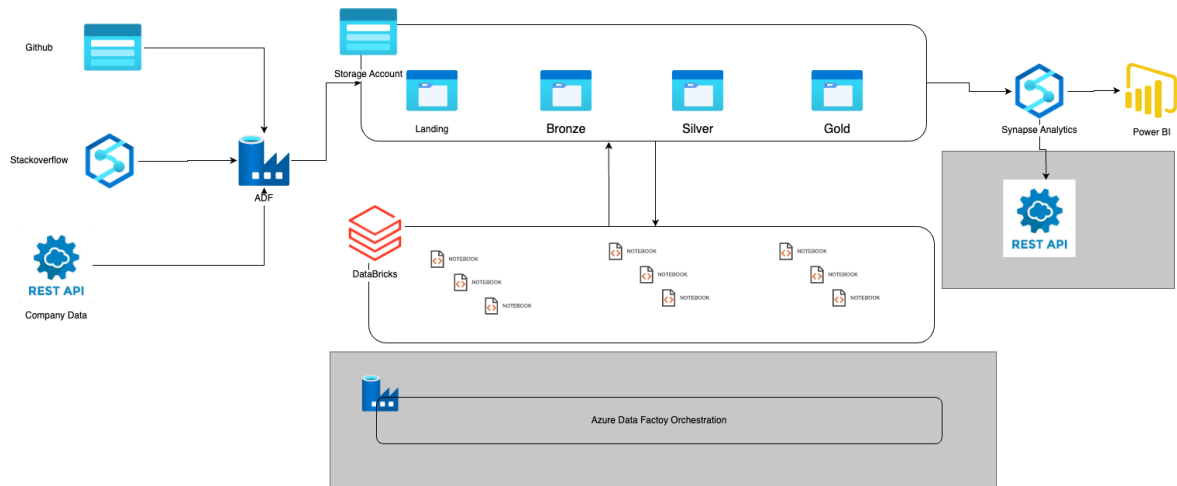
We will get the configuration data from an REST API

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.

# Architecture



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

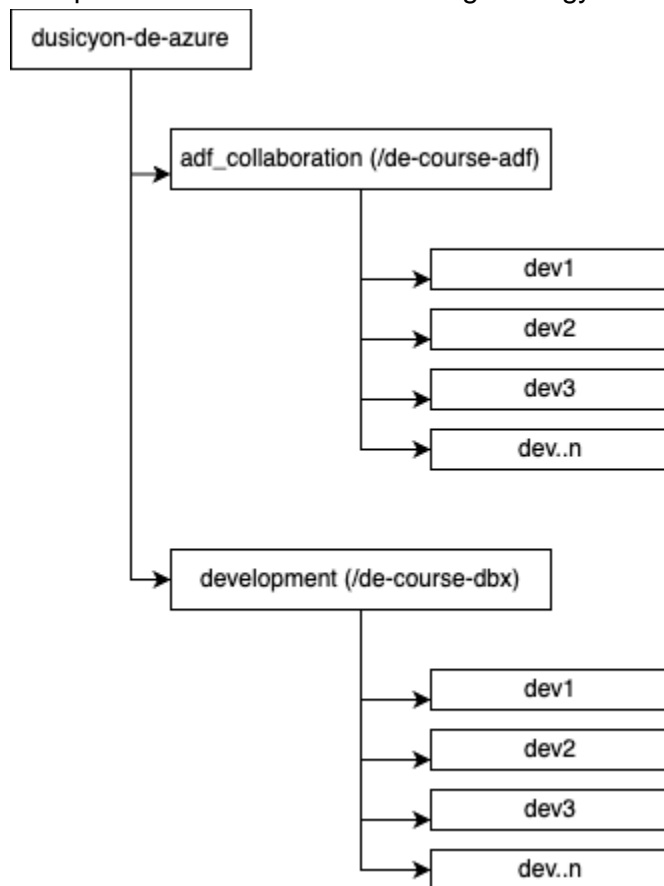
- **landing layer:** target of the ingestion pipelines, data is untyped, untransformed here, contains source data “as-is” (loaded with an ADF copy activity)
- **bronze/raw layer:** source data with minimal transformations but already in delta format (loaded with Databricks Notebooks called from an ADF 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.



# 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.



## 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.

## 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.azure.synapse.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: VUFSrvMVgBg6u4uqwrBpN8qM9NpS9pwHMbKQCtsZI8bVImHFBpx1jYr4Jp3wTEpmVudEYt+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=yyymmdd
      - 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)

## 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

## Materials

### 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:

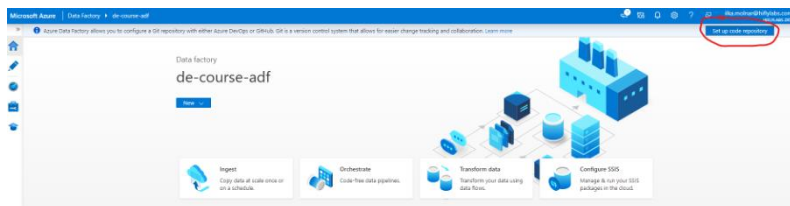
- We create only one environment at start (for testing and production purpose we should create separate storage accounts)
- In the dev storage account it is also a possible approach to create separate containers for the developers (in that case the bronze/silver/gold layers would be folders in dev environments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

### Azure Data Factory

No special configuration is needed.

## 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

Hifylabs

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

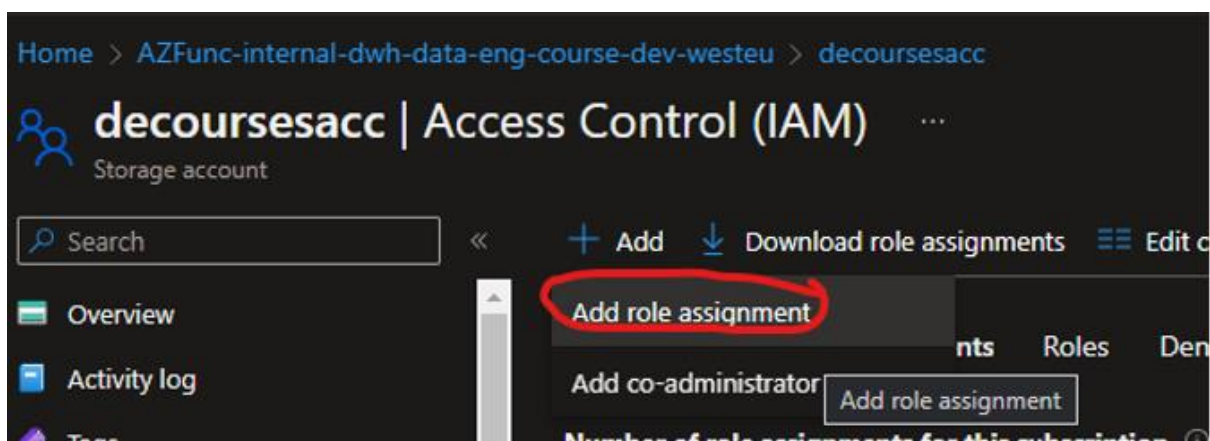
/de-course-adf

Import existing resources

☐


Import existing resources to repository

## BlobStorage contributor access






## 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](#) 


× 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**

#### Select members







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](#)