Week 4

Chapter 4

Data Engineer

Trainer: Balazs Balogh

CUBIX

INSTITUTE OF TECHNOLOGY

AWS Lambda – Transform function

- Folder structure in transformed_data/

Name
community_areas/
company/
date/
master_table_previous_version/
payment_type/
taxi_trips/
weather/

AWS Lambda – Transform function

- Taxi trips DataFrame transformations:

```
taxi_trips.drop(["pickup_census_tract", "dropoff_census_tract"], axis=1, inplace=True)
taxi_trips.drop(["pickup_centroid_location", "dropoff_centroid_location"], axis=1, inplace=True)

taxi_trips.dropna(inplace=True)

taxi_trips.rename(columns={"pickup_community_area": "pickup_community_area_id",
                           "dropoff_community_area": "dropoff_community_area_id"}, inplace=True)

taxi_trips["datetime_for_weather"] = pd.to_datetime(taxi_trips["trip_start_timestamp"]).dt.floor("H")
```

AWS Lambda – Transform function

```python
def taxi_trips_transformations(taxi_trips: pd.DataFrame) -> pd.DataFrame:
    """Perform transformations with the taxi data.

    Parameters
    ----------
    taxi_trips : pd.DataFrame
        The DataFrame holding the daily taxi trips

    Returns
    -------
    pd.DataFrame
        The cleaned, transformed DataFrame holding the daily taxi trips.
    """

    if not isinstance(taxi_trips, pd.DataFrame):
        raise TypeError("taxi_trips is not a valid pandas DataFrame.")

    taxi_trips.drop(["pickup_census_tract", "dropoff_census_tract",
                     "pickup_centroid_location", "dropoff_centroid_location"], axis=1, inplace=True)

    taxi_trips.dropna(inplace=True)

    taxi_trips.rename(columns={"pickup_community_area": "pickup_community_area_id",
                               "dropoff_community_area": "dropoff_community_area_id"}, inplace=True)

    taxi_trips["datetime_for_weather"] = pd.to_datetime(taxi_trips["trip_start_timestamp"]).dt.floor("H")

    return taxi_trips
```

- Function created from taxi trips DataFrame transformations.
- It now has type hints and docstring, and return the taxi_trips DataFrame.

CUBIX INSTITUTE OF TECHNOLOGY

AWS Lambda – Error handling

- There are multiple ways of error handling. In the previous function we used a type check for the only parameter (taxi_trips).

```
if not isinstance(taxi_trips, pd.DataFrame):
    raise TypeError("taxi_trips is not a valid pandas DataFrame.")
```

- It's a good practice to use at least these kind of error handlings, to notify the users with a specific message.
- You can come up other ideas, like checking if all the columns are in the DataFrame we want to drop. If not, then throw an error "Column not found in DataFrame.".
- There are multiple types of errors built in in Python: TypeError, ValueError, AttributeError, etc.

AWS Lambda – Handling company data

```python
def update_company_master(taxi_trips: pd.DataFrame, company_master: pd.DataFrame) -> pd.DataFrame:
    """Extend the company master with new companies if there are new companies.

    Parameters
    ----------
    taxi_trips : pd.DataFrame
        DataFrame holding the daily taxi trips.
    company_master : pd.DataFrame
        DataFrame holding the company_master data.

    Returns
    -------
    pd.DataFrame
        The updated company_master data, if new companies are in the taxi data, they will be loaded to it.
    """

    company_max_id = company_master["company_id"].max()

    new_companies_list = [company for company in taxi_trips["company"].values if company not in company_master["company"].values]
    new_companies_df = pd.DataFrame({
        "company_id": range(company_max_id + 1, company_max_id + len(new_companies_list) + 1),
        "company": new_companies_list
    })

    updated_company_master = pd.concat([company_master, new_companies_df], ignore_index=True)

    return updated_company_master
```

- Get the new companies from the current day's taxi data and compare to last day's company master data.
- If there are new companies, give them a max(company_id) + 1 company_id, and the name of the company.

CUBIX
INSTITUTE OF TECHNOLOGY

AWS Lambda – Handling payment type data

```python
def update_payment_type_master(taxi_trips: pd.DataFrame, payment_type_master: pd.DataFrame) -> pd.DataFrame:
    """Extend the payment type master with new payment types if there are new payment types.

    Parameters
    ----------
    taxi_trips : pd.DataFrame
        DataFrame holding the daily taxi trips.
    payment_type_master : pd.DataFrame
        DataFrame holding the payment_type_master data.

    Returns
    -------
    pd.DataFrame
        The updated payment_type_master data, if new payment types are in the taxi data, they will be loaded to it.
    """

    payment_type_max_id = payment_type_master["payment_type_id"].max()

    new_payment_types_list = [payment_type for payment_type in taxi_trips["payment_type"].values if payment_type not
    new_payment_type_df = pd.DataFrame({
        "payment_type_id": range(payment_type_max_id + 1, payment_type_max_id + len(new_payment_types_list) + 1),
        "payment_type": new_payment_types_list
    })

    updated_payment_type_master = pd.concat([payment_type_master, new_payment_type_df], ignore_index=True)

    return updated_payment_type_master
```

- The same method with the payment type data.

CUBIX INSTITUTE OF TECHNOLOGY

AWS Lambda – Rethink the updates

- Since we are doing the same for the payment type and company, we should shorten our code with unifying the two functions.
- Duplicate code should be avoided.

```
max_id = master[id_column].max()

new_values_list = [value for value in taxi_trips[value_column].values if value not in master[value_column].values]
new_values_df = pd.DataFrame({
    id_column: range(max_id + 1, max_id + len(new_values_list) + 1),
    value_column: new_values_list
})

updated_master = pd.concat([master, new_values_df], ignore_index=True)

return updated_master
```

AWS Lambda – Update with master data

- Since we now have the two master tables, we can merge them to our base table, taxi_trips.

```
taxi_trips_id = taxi_trips.merge(payment_type_master, on="payment_type")
taxi_trips_id = taxi_trips_id.merge(company_master, on="company")

taxi_trips_id.drop(["payment_type", "company"], axis=1, inplace=True)
```

AWS Lambda – Transform weather data

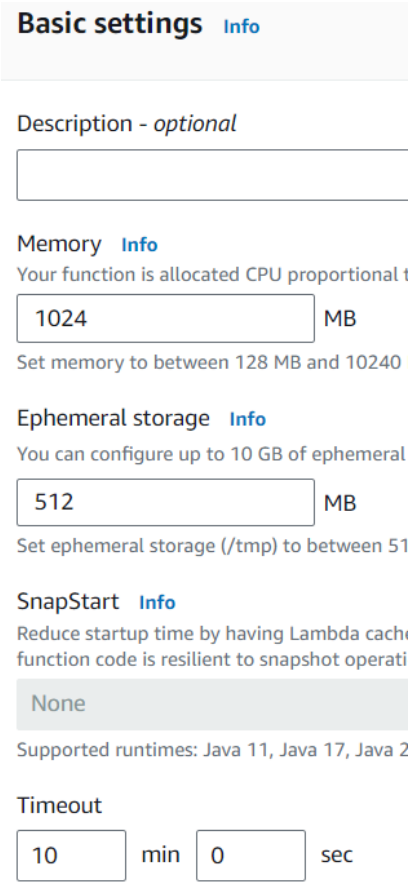
```python
def transform_weather_data(weather_data: json) -> pd.DataFrame:
    """Make transformations on the daily weather api response.

    Parameters
    ----------
    weather_data : json
        The daily weather data from the Open Meteo API.

    Returns
    -------
    pd.DataFrame
        A DataFrame representation of the data.
    """
    weather_data_filtered = {
            "datetime": weather_data["hourly"]["time"],
            "tempretaure": weather_data["hourly"]["temperature_2m"],
            "wind_speed": weather_data["hourly"]["wind_speed_10m"],
            "rain": weather_data["hourly"]["rain"],
            "precipitation": weather_data["hourly"]["precipitation"],
        }

    weather_df = pd.DataFrame(weather_data_filtered)

    weather_df["datetime"] = pd.to_datetime(weather_df["datetime"])

    return weather_df
```

- Copy the already written code from the weather api notebook, and create a function from it.

CUBIX
INSTITUTE OF TECHNOLOGY

AWS Lambda – Create the second function

- For daily use 2 minutes timeout with 256 MB memory will be enough.
- But when you create the function, most likely you will have multiple days of data, so when you first test it with all the raw files, raise the values to 10 minutes and 1024 MB memory, it will be more than enough.



CUBIX
INSTITUTE OF TECHNOLOGY

AWS Lambda – Start with weather data load

- Start the function with getting the weather data.
- Create an instance of the s3 client, and use in a for loop it's list_object method, which will list all the files in the specified folder (Bucket + Prefix).
- With the get_object method you retrieve the file, and in the "Body" part of it, you'll find the JSON loadable content.
- Use json.loads() on it.

lambda_function × Execution results ×

```python
import json

import boto3
import pandas as pd

def lambda_handler(event, context):
    s3 = boto3.client("s3")

    bucket = "cubix-chicago-taxi-bb"
    raw_weather_folder = "raw_data/to_processed/weather_data/"

    # WEATHER DATA
    for file in s3.list_objects(Bucket=bucket, Prefix=raw_weather_folder)['Contents']:
        weather_key = file['Key']

        if weather_key.split("/")[-1].strip() != "":
            if weather_key.split(".")[1] == "json":

                response = s3.get_object(Bucket=bucket, Key=weather_key)
                content = response['Body']
                json_content = json.loads(content.read())

                print(json_content)
```

CUBIX INSTITUTE OF TECHNOLOGY

AWS Lambda – Weather data transformations

- Copy the transform_weather_data function from the 07_transform_load notebook.
- Extend the main function (lambda_handler) with it:
  - weather_data DataFrame

lambda_function × Execution results ×

```python
    weather_df = pd.DataFrame(weather_data_filtered)

    weather_df["datetime"] = pd.to_datetime(weather_df["datetime"])

    return weather_df

def lambda_handler(event, context):
    s3 = boto3.client("s3")

    bucket = "cubix-chicago-taxi-bb"
    raw_weather_folder = "raw_data/to_processed/weather_data/"

    # WEATHER DATA TRANSFORMATION AND LOADING
    for file in s3.list_objects(Bucket=bucket, Prefix=raw_weather_folder)['Contents']:
        weather_key = file['Key']

        if weather_key.split("/")[-1].strip() != "":
            if weather_key.split(".")[1] == "json":

                response = s3.get_object(Bucket=bucket, Key=weather_key)
                content = response['Body']
                weather_data_json = json.loads(content.read())

                weather_data = transform_weather_data(weather_data_json)

                # upload to s3 function
```

AWS Lambda – Taxi data load

- Now do the same with the taxi data, just copy and modify the code from the weather load.

lambda_function × Execution results ×

```python
    weather_df = pd.DataFrame(weather_data_filtered)

    weather_df["datetime"] = pd.to_datetime(weather_df["datetime"])

    return weather_df

def lambda_handler(event, context):
    s3 = boto3.client("s3")

    bucket = "cubix-chicago-taxi-bb"
    raw_weather_folder = "raw_data/to_processed/weather_data/"
    raw_taxi_trips_folder = "raw_data/to_processed/taxi_data/"


    # TAXI DATA TRANSFORMATION AND LOADING
    for file in s3.list_objects(Bucket=bucket, Prefix=raw_taxi_trips_folder)["Contents"]:
        taxi_trip_key = file["Key"]

        if taxi_trip_key.split("/")[-1].strip() != "":
            if taxi_trip_key.split(".")[1] == "json":

                response = s3.get_object(Bucket=bucket, Key=taxi_trip_key)
                content = response["Body"]
                taxi_trip_data_json = json.loads(content.read())

                print(taxi_trip_data_json)


    # WEATHER DATA TRANSFORMATION AND LOADING
    for file in s3.list_objects(Bucket=bucket, Prefix=raw_weather_folder)["Contents"]:
        weather_key = file["Key"]

        if weather_key.split("/")[-1].strip() != "":
            if weather_key.split(".")[1] == "json":

                response = s3.get_object(Bucket=bucket, Key=weather_key)
                content = response["Body"]
                weather_data_json = json.loads(content.read())

                weather_data = transform_weather_data(weather_data_json)

                # upload to s3 function
```

CUBIX INSTITUTE OF TECHNOLOGY

AWS Lambda – Taxi trips transformations

- Copy the taxi_trips_transformations function from the 07_transform_load notebook.
- Create a DataFrame from the json data.
- Create the taxi_trips variable with the taxi_trips_transformation function called on the just created DataFrame.

lambda_function

```python

def lambda_handler(event, context):
    s3 = boto3.client("s3")

    bucket = "cubix-chicago-taxi-bb"
    raw_weather_folder = "raw_data/to_processed/weather_data/"
    raw_taxi_trips_folder = "raw_data/to_processed/taxi_data/"

    # TAXI DATA TRANSFORMATION AND LOADING
    for file in s3.list_objects(Bucket=bucket, Prefix=raw_taxi_trips_folder)["Contents"]:
        taxi_trip_key = file["Key"]

        if taxi_trip_key.split("/")[-1].strip() != "":
            if taxi_trip_key.split(".")[1] == "json":

                response = s3.get_object(Bucket=bucket, Key=taxi_trip_key)
                content = response["Body"]
                taxi_trips_data_json = json.loads(content.read())

                taxi_trips_data_raw = pd.DataFrame(taxi_trips_data_json)
                taxi_trips = taxi_trips_transformations(taxi_trips_data_raw)

                print(taxi_trips.columns)
                print(taxi_trips.shape)

    # WEATHER DATA TRANSFORMATION AND LOADING
    for file in s3.list_objects(Bucket=bucket, Prefix=raw_weather_folder)["Contents"]:
        weather_key = file["Key"]

        if weather_key.split("/")[-1].strip() != "":
            if weather_key.split(".")[1] == "json":

                response = s3.get_object(Bucket=bucket, Key=weather_key)
                content = response["Body"]
                weather_data_json = json.loads(content.read())

                weather_data = transform_weather_data(weather_data_json)

                # upload to s3 function

```

CUBIX INSTITUTE OF TECHNOLOGY

AWS Lambda – Update master tables

- Copy the update_master function from the 07_transform_load notebook.
- Extend the code with the company_master_updated and payment_type_master_updated variables.
- Note, that at this point we don't have the master tables loaded yet.

```
def lambda_handler(event, context):
    s3 = boto3.client("s3")

    bucket = "cubix-chicago-taxi-bb"
    raw_weather_folder = "raw_data/to_processed/weather_data/"
    raw_taxi_trips_folder = "raw_data/to_processed/taxi_data/"

    # TAXI DATA TRANSFORMATION AND LOADING
    for file in s3.list_objects(Bucket=bucket, Prefix=raw_taxi_trips_folder)["Contents"]:
        taxi_trip_key = file["Key"]

        if taxi_trip_key.split("/")[-1].strip() != "":
            if taxi_trip_key.split(".")[1] == "json":

                response = s3.get_object(Bucket=bucket, Key=taxi_trip_key)
                content = response["Body"]
                taxi_trips_data_json = json.loads(content.read())

                taxi_trips_data_raw = pd.DataFrame(taxi_trips_data_json)
                taxi_trips = taxi_trips_transformations(taxi_trips_data_raw)

                company_master_updated = update_master(taxi_trips, company_master, "company_id", "company")
                payment_type_master_updated = update_master(taxi_trips, payment_type_master, "payment_type_id", "payment_type")
```

CUBIX
INSTITUTE OF TECHNOLOGY

AWS Lambda – Read csv from S3 bucket

- read_csv_from_s3 function is created here.
- Try to create error handlings, to check if the bucket / path / filename exists.
- Or if the file can't be loaded as a DataFrame.
- Load the files, and try the update_master function.

```python
def read_csv_from_s3(bucket: str, path: str, filename: str) -> pd.DataFrame:
    """Downloads a csv file from an S3 bucket.

    Parameters
    ----------
    bucket : str
        The bucket where the files at.

    path : str
        The folders to the file.

    filename : str
        Name of the file.

    Returns
    ------
    pd.DataFrame
        A DataFrame of the downloaded file.

    """
    s3 = boto3.client("s3")

    full_path = f"{path}{filename}"

    object = s3.get_object(Bucket=bucket, Key=full_path)
    object = object["Body"].read().decode("utf-8")
    output_df = pd.read_csv(StringIO(object))

    return output_df
```

AWS Lambda – Update taxi trips with master data

- Copy from the 07_transform_load notebook the update_taxi_trips_with_master_data function.
- Use it to create taxi_trips final DataFrame.

```python
    payment_type_master = read_csv_from_s3(bucket=bucket, path=payment_type_master_folder, filename=payment_type_master_file_name)
    company_master = read_csv_from_s3(bucket=bucket, path=company_type_master_folder, filename=company_master_file_name)

    # TAXI DATA TRANSFORMATION AND LOADING
    for file in s3.list_objects(Bucket=bucket, Prefix=raw_taxi_trips_folder)["Contents"]:
        taxi_trip_key = file["Key"]

        if taxi_trip_key.split("/")[-1].strip() != "":
            if taxi_trip_key.split(".")[1] == "json":

                response = s3.get_object(Bucket=bucket, Key=taxi_trip_key)
                content = response["Body"]
                taxi_trips_data_json = json.loads(content.read())

                taxi_trips_data_raw = pd.DataFrame(taxi_trips_data_json)
                taxi_trips_transformed = taxi_trips_transformations(taxi_trips_data_raw)

                company_master_updated = update_master(taxi_trips_transformed, company_master, "company_id", "company")
                payment_type_master_updated = update_master(taxi_trips_transformed, payment_type_master, "payment_type_id", "payment_type")

                taxi_trips = update_taxi_trips_with_master_data(taxi_trips_transformed, payment_type_master_updated, company_master_updated)
```

CUBIX
INSTITUTE OF TECHNOLOGY

AWS Lambda – Upload master data to S3

```python
def upload_master_data_to_s3(bucket: str, path: str, file_type: str, dataframe: pd.DataFrame):
    """
    Uploads master data (payment_type or company) to S3. Copies the previous version and creates the new one.

    Parameters
    ----------
    bucket : str
        Name of the S3 bucket where we want to store the files.

    path : str
        Path within the bucket to upload the files.

    file_type : str
        Either "company" or "payment_type".

    dataframe : pd.DataFrame
        The dataframe to be uploaded.

    Returns
    -------
    None
    """

    s3 = boto3.client("s3")

    master_file_path = f"{path}{file_type}_master.csv"
    previous_master_file_path = f"transformed_data/master_table_previous_version/{file_type}_master_previous_version.csv"

    s3.copy_object(
        Bucket=bucket,
        CopySource={"Bucket": bucket, "Key": master_file_path},
        Key=previous_master_file_path
    )

    upload_dataframe_to_s3(bucket=bucket, dataframe=dataframe, path=master_file_path)
```

- This is for the company and payment type updates.

CUBIX
INSTITUTE OF TECHNOLOGY

AWS Lambda – Upload master data to S3

- Extend the main code with the two upload_master_data_to_s3 function calls.
- You can use a for loop to shorten the code, and avoid duplications.

```python
# TAXI DATA TRANSFORMATION AND LOADING
for file in s3.list_objects(Bucket=bucket, Prefix=raw_taxi_trips_folder)["Contents"]:
    taxi_trip_key = file["Key"]

    if taxi_trip_key.split("/")[-1].strip() != "":
        if taxi_trip_key.split(".")[1] == "json":

            response = s3.get_object(Bucket=bucket, Key=taxi_trip_key)
            content = response["Body"]
            taxi_trips_data_json = json.loads(content.read())

            taxi_trips_data_raw = pd.DataFrame(taxi_trips_data_json)
            taxi_trips_transformed = taxi_trips_transformations(taxi_trips_data_raw)

            company_master_updated = update_master(taxi_trips_transformed, company_master, "company_id", "company")
            payment_type_master_updated = update_master(taxi_trips_transformed, payment_type_master, "payment_type_id", "payment_type")

            taxi_trips = update_taxi_trips_with_master_data(taxi_trips_transformed, payment_type_master_updated, company_master_updated)

            upload_master_data_to_s3(bucket=bucket, path=payment_type_master_folder, file_type="payment_type", dataframe=payment_type_master_updated)
            print("payment_type_master has been updated.")
            upload_master_data_to_s3(bucket=bucket, path=company_master_folder, file_type="company", dataframe=company_master_updated)
            print("payment_type_master has been updated.")
```

CUBIX
INSTITUTE OF TECHNOLOGY

AWS Lambda – Move taxi trips DataFrame

- The upload_and_move_file_on_s3 function is responsible for uploading a file, and move it from the base folder to another.
- It has a nested function, upload_dataframe_to_s3 which is responsible only the upload part.

```python
s3 = boto3.client("s3")

formatted_date = dataframe[datetime_col].iloc[0].strftime("%Y-%m-%d")
new_path_with_filename = f"{target_path_transformed}{file_type}_{formatted_date}.csv"

upload_dataframe_to_s3(bucket=bucket, dataframe=dataframe, path=new_path_with_filename)

s3.copy_object(
    Bucket=bucket,
    CopySource={"Bucket": bucket, "Key":f"{source_path}{filename}"},
    Key=f"{target_path_raw}{filename}"
)

s3.delete_object(Bucket=bucket, Key=f"{source_path}{filename}")
```

```python
upload_and_move_file_on_s3(
        dataframe=taxi_trips,
        datetime_col="datetime_for_weather",
        bucket=bucket,
        file_type="taxi",
        filename=filename,
        source_path=raw_taxi_trips_folder,
        target_path_raw=target_taxi_trips_folder,
        target_path_transformed=transformed_taxi_trips_folder
    )
print("taxi_trips is uploaded and moved.")
```

AWS Lambda – Testing the taxi trips ETL

- Now as the taxi trips codes are finalized it's time to test.
- First, test with only one file, so create a variable for it "file_name_for_testing_taxi_trips".
- Extend the for loop with an if statement, if the file name is the one we need, run the transformation.

```
# DELETE THESE VARIABLES AFTER TESTING
file_name_for_testing_taxi_trips = "taxi_raw_2023-09-17.json"
file_name_for_testing_weather = "weather_raw_2023-09-17.json"
# DELETE THESE VARIABLES AFTER TESTING

# TAXI DATA TRANSFORMATION AND LOADING
for file in s3.list_objects(Bucket=bucket, Prefix=raw_taxi_trips_folder)["Contents"]:
    taxi_trip_key = file["Key"]

    if taxi_trip_key.split("/")[-1].strip() != "":
        if taxi_trip_key.split(".")[1] == "json":

            filename = taxi_trip_key.split("/")[-1]

            if filename == file_name_for_testing_taxi_trips:
```

AWS Lambda – Testing the weather data ETL

- The same goes with the weather data.
- Choose one file, and extend the for loop.

AWS Lambda – Test the full process

- If working with one taxi and one weather file is a success, then run it to the whole folders.
- Remember to raise the timeout and the memory to 8 min and 1024 MB.
- After you finished, change them back to 2 min and 256 mbytes.

AWS Lambda – Set the automation

- Set the S3 trigger, when an object is created in the raw_data/to_processed/taxi_data folder, run the Lambda function.

Triggers (1) Info

Find triggers

	Trigger
	S3: cubix-chicago-taxi-bb arn:aws:s3:::cubix-chicago-taxi-bb ▼ **Details** Bucket arn: **arn:aws:s3:::cubix-chicago-taxi-bb** Event types: **s3:ObjectCreated:*** Notification name: **24d40cd2-db98-4bbb-b658-de3485583516** Prefix: **raw_data/to_processed/taxi_data**

CUBIX INSTITUTE OF TECHNOLOGY