Medium article:

Hey Kaiko,

this is part of my ongoing script. It is sort of outsourced to medium, as the article there is our deliverable for this module—two birds with one stone.

And hi Liane and Christopher,

you may have noticed, that my target group is I. Therefore i am not sure this will be a real pleasure to read through, but since you made no specifications and i am a sucker for efficiency—there you go ;)

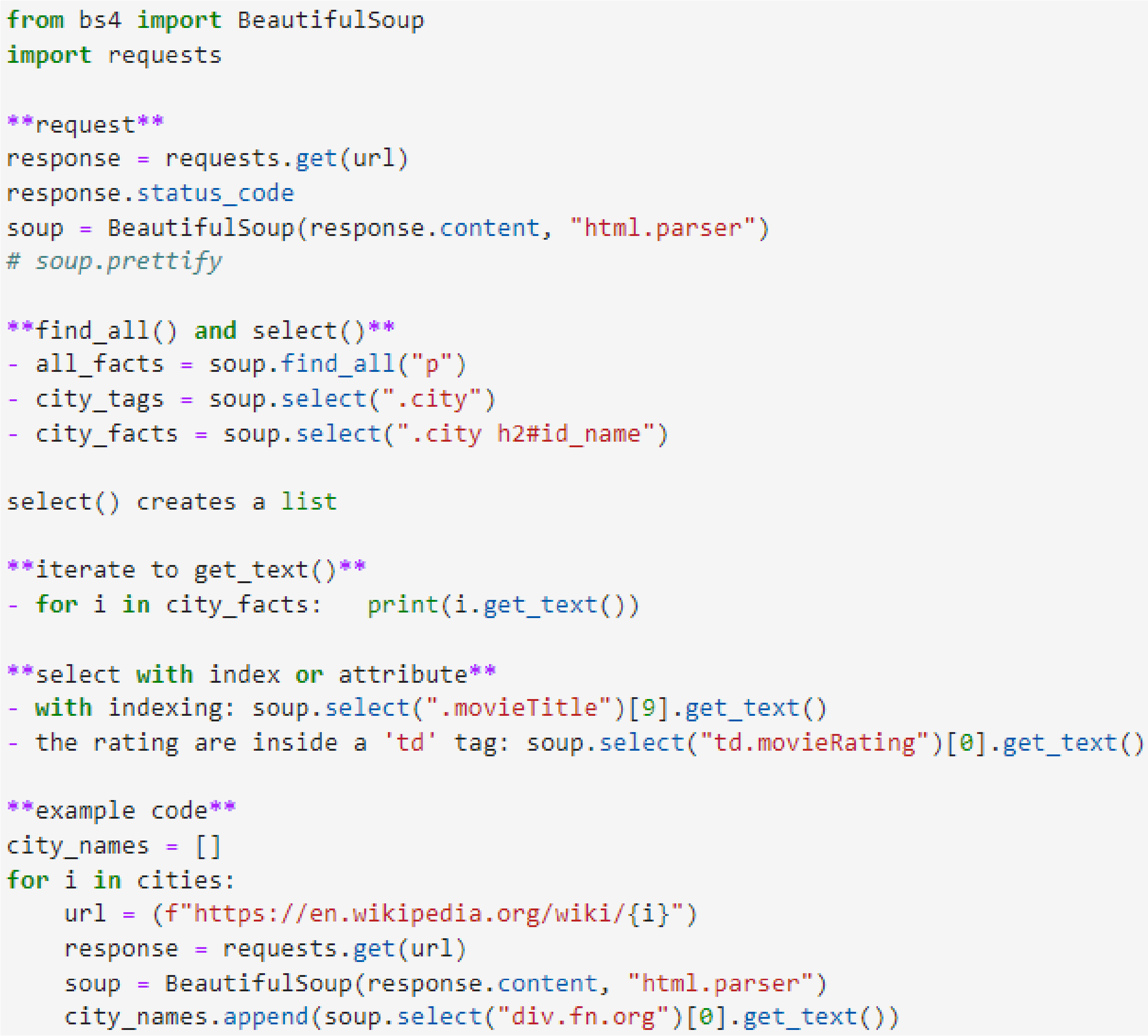
script for module_50 in WBSCS DS023 “data engineering”

- Datacollection (WebScraping and API, rapidAPI)
- Pipelines (ipy-sql and ipy-RDS)
- Modularisation (ipy functions to lambda modules)
- Automatisation (rules and schedules)

general learning of module_50 has been how to modularize my code, build and automise a pipeline.

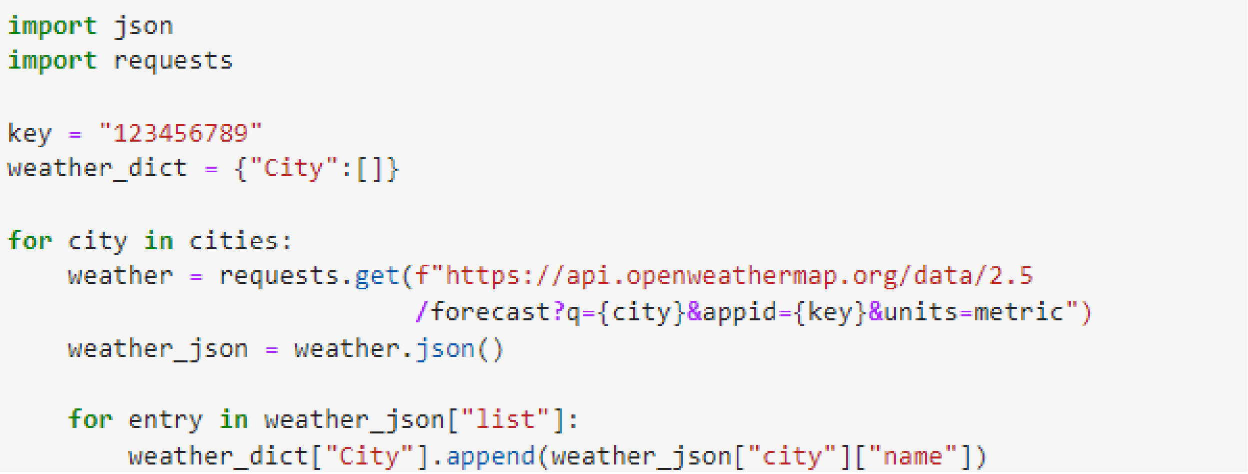
# Datacollection via WebScraping and API

## Webscraping, beautifulSoup

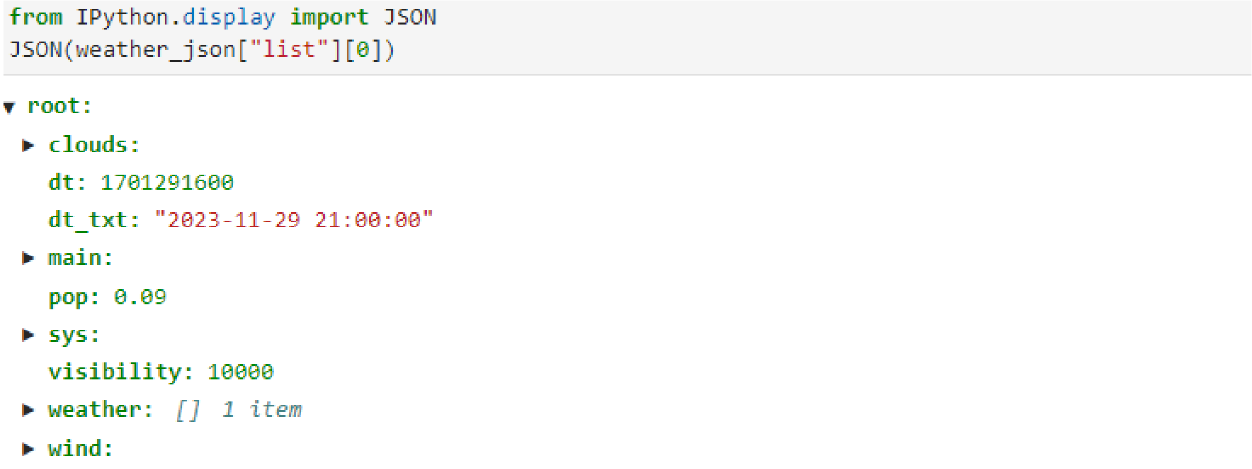


```
from bs4 import BeautifulSoup
import requests

**request**
response = requests.get(url)
response.status_code
soup = BeautifulSoup(response.content, "html.parser")
# soup.prettify

**find_all() and select()**
- all_facts = soup.find_all("p")
- city_tags = soup.select(".city")
- city_facts = soup.select(".city h2#id_name")

select() creates a list

**iterate to get_text()**
- for i in city_facts:   print(i.get_text())

**select with index or attribute**
- with indexing: soup.select(".movieTitle")[9].get_text()
- the rating are inside a 'td' tag: soup.select("td.movieRating")[0].get_text()

**example code**
city_names = []
for i in cities:
    url = (f"https://en.wikipedia.org/wiki/{i}")
    response = requests.get(url)
    soup = BeautifulSoup(response.content, "html.parser")
    city_names.append(soup.select("div.fn.org")[0].get_text())
```

## API, json



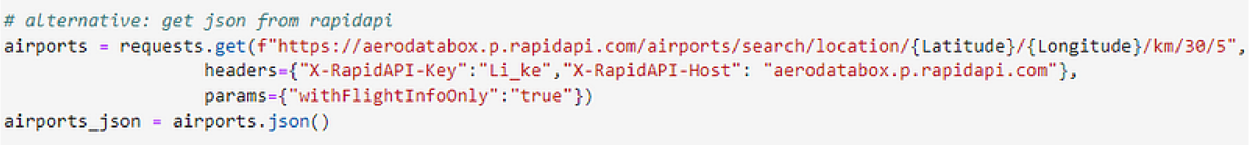
```python
import json
import requests

key = "123456789"
weather_dict = {"City":[]}

for city in cities:
    weather = requests.get(f"https://api.openweathermap.org/data/2.5
                           /forecast?q={city}&appid={key}&units=metric")
    weather_json = weather.json()

    for entry in weather_json["list"]:
        weather_dict["City"].append(weather_json["city"]["name"])
```



```python
from IPython.display import JSON
JSON(weather_json["list"][0])
```

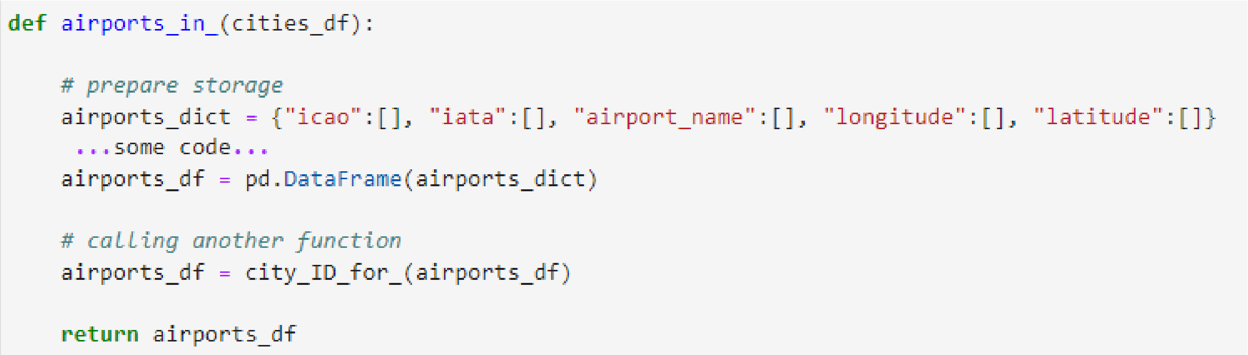
```
▼ root:
  ▶ clouds:
    dt: 1701291600
    dt_txt: "2023-11-29 21:00:00"
  ▶ main:
    pop: 0.09
  ▶ sys:
    visibility: 10000
  ▶ weather:  []  1 item
  ▶ wind:
```

## rapidAPI, json



```python
# alternative: get json from rapidapi
airports = requests.get(f"https://aerodatabox.p.rapidapi.com/airports/search/location/{Latitude}/{Longitude}/km/30/5",
                        headers={"X-RapidAPI-Key":"Li_ke","X-RapidAPI-Host": "aerodatabox.p.rapidapi.com"},
                        params={"withFlightInfoOnly":"true"})
airports_json = airports.json()
```

Modularisation ipy functions

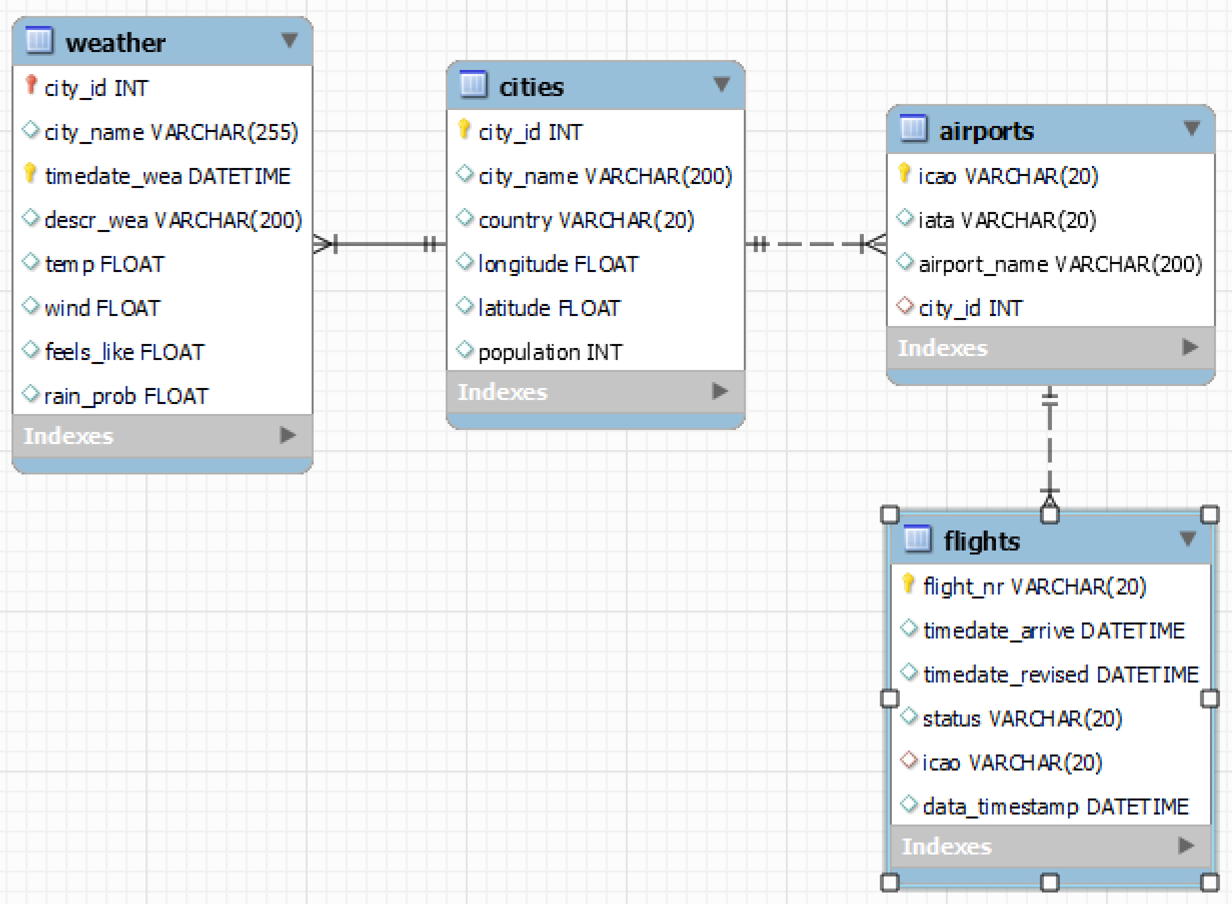


```
def airports_in_(cities_df):

    # prepare storage
    airports_dict = {"icao":[], "iata":[], "airport_name":[], "longitude":[], "latitude":[]}
     ...some code...
    airports_df = pd.DataFrame(airports_dict)

    # calling another function
    airports_df = city_ID_for_(airports_df)

    return airports_df
```

# Pipelines

## jpy to sql

first create schema in mysql with exactly corresponding data model as used in ipy dataframes.



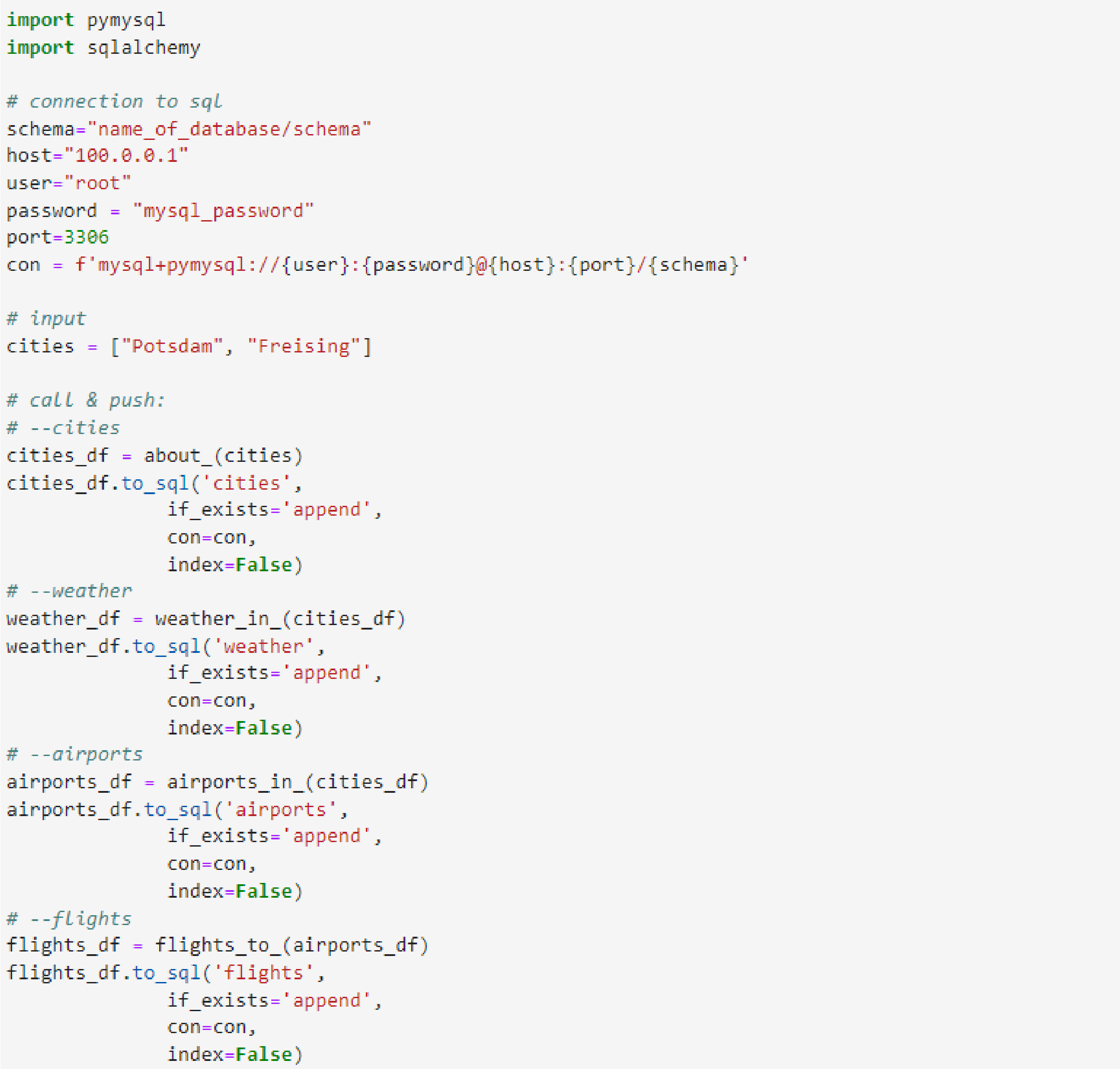
weather
city_id INT
city_name VARCHAR(255)
timedate_wea DATETIME
descr_wea VARCHAR(200)
temp FLOAT
wind FLOAT
feels_like FLOAT
rain_prob FLOAT
Indexes

cities
city_id INT
city_name VARCHAR(200)
country VARCHAR(20)
longitude FLOAT
latitude FLOAT
population INT
Indexes

airports
icao VARCHAR(20)
iata VARCHAR(20)
airport_name VARCHAR(200)
city_id INT
Indexes

flights
flight_nr VARCHAR(20)
timedate_arrive DATETIME
timedate_revised DATETIME
status VARCHAR(20)
icao VARCHAR(20)
data_timestamp DATETIME
Indexes

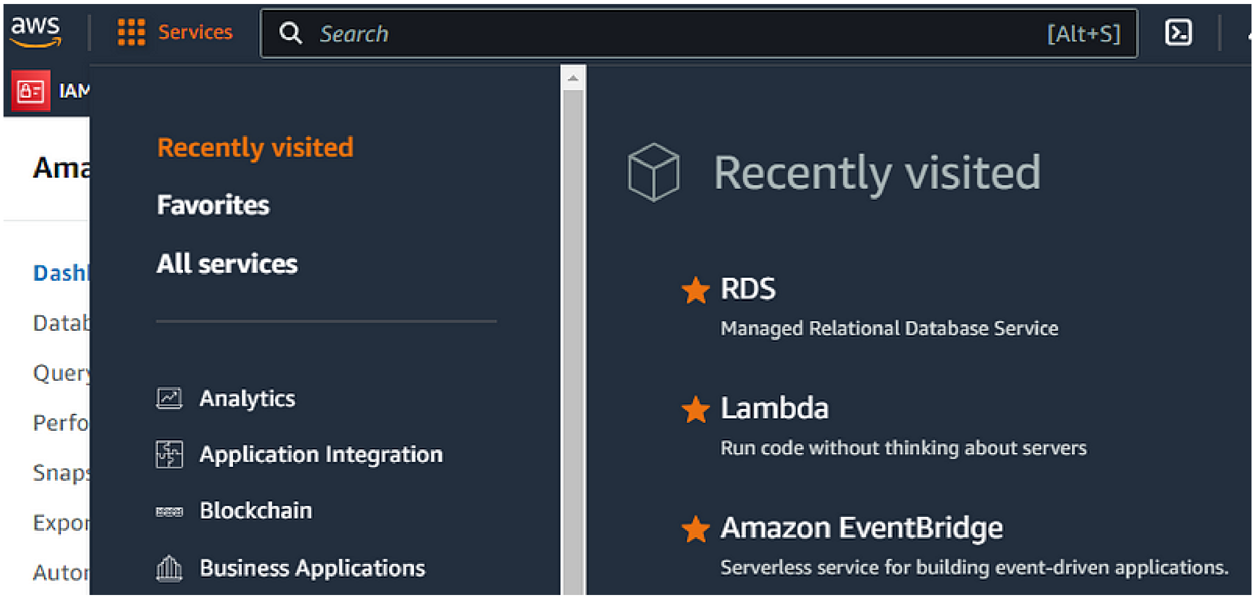
Then push data from ipy to sql (“master codebox”)



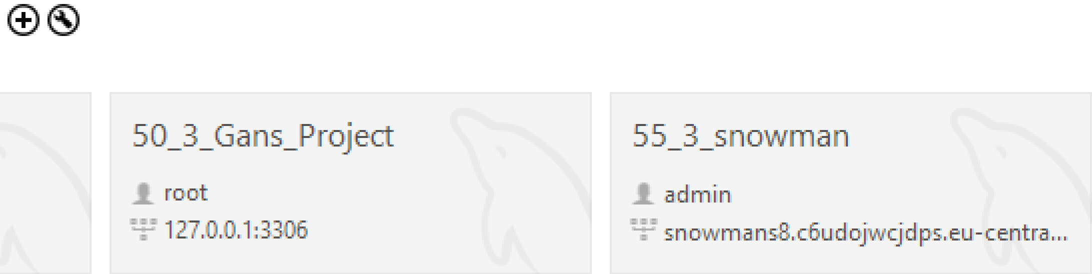
```python
import pymysql
import sqlalchemy

# connection to sql
schema="name_of_database/schema"
host="100.0.0.1"
user="root"
password = "mysql_password"
port=3306
con = f'mysql+pymysql://{user}:{password}@{host}:{port}/{schema}'

# input
cities = ["Potsdam", "Freising"]

# call & push:
# --cities
cities_df = about_(cities)
cities_df.to_sql('cities',
              if_exists='append',
              con=con,
              index=False)
# --weather
weather_df = weather_in_(cities_df)
weather_df.to_sql('weather',
              if_exists='append',
              con=con,
              index=False)
# --airports
airports_df = airports_in_(cities_df)
airports_df.to_sql('airports',
              if_exists='append',
              con=con,
              index=False)
# --flights
flights_df = flights_to_(airports_df)
flights_df.to_sql('flights',
              if_exists='append',
              con=con,
              index=False)
```

## jpy to RDS (AWS)

create RDS instance in AWS and create database in instance by copy & paste the sql schema.



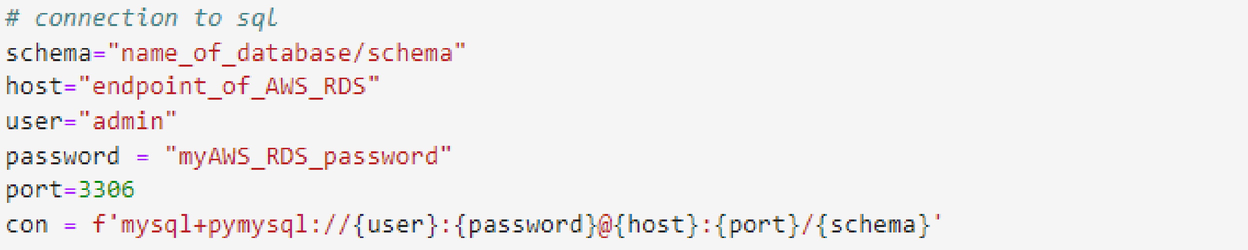
Check RDS database in sql-workbench: create new connection in sql-workbench to endpoint of RDS instance



50_3_Gans_Project
root
127.0.0.1:3306

55_3_snowman
admin
snowmans8.c6udojwcjdps.eu-centra...

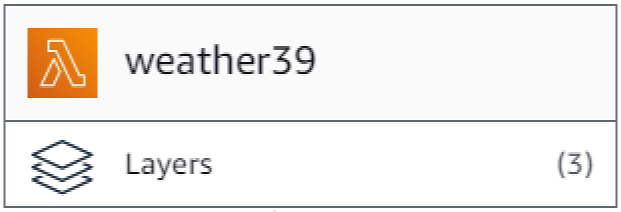
For the push from ipy to RDS, only change connection declarations from sqlroot-connetion to RDS-connection



```
# connection to sql
schema="name_of_database/schema"
host="endpoint_of_AWS_RDS"
user="admin"
password = "myAWS_RDS_password"
port=3306
con = f'mysql+pymysql://{user}:{password}@{host}:{port}/{schema}'
```

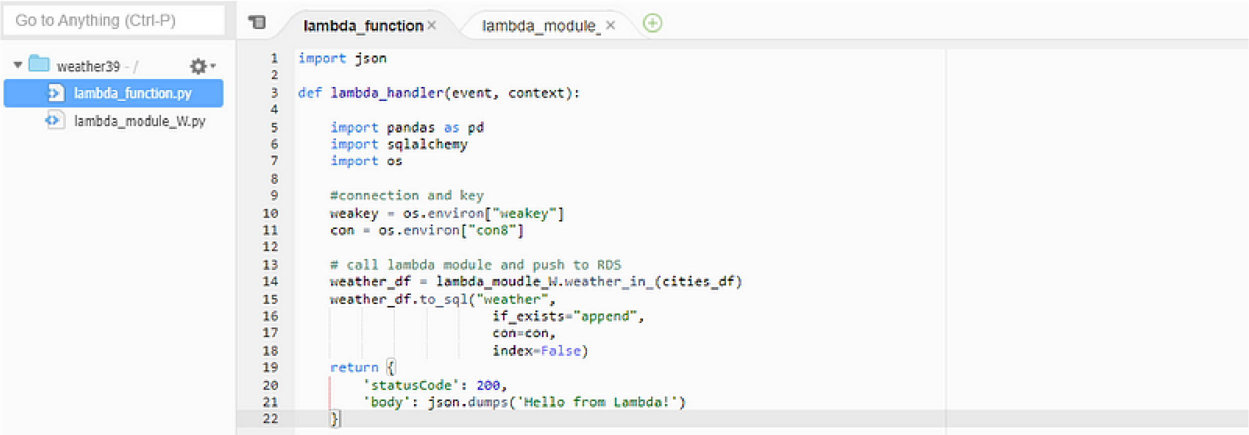
# lambda functions and modularisation by lambda modules

## preps: create layers (ipy-analog: import librarys)



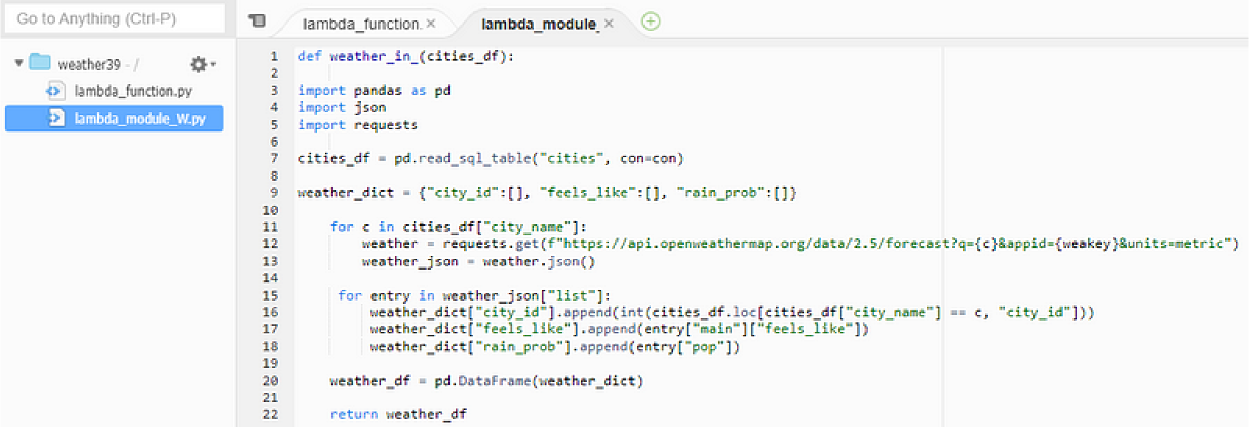
ARN: https://github.com/keithrozario/Klayers/blob/master/README.MD

## create lambda funtion (ipy-analog: “master codebox”)



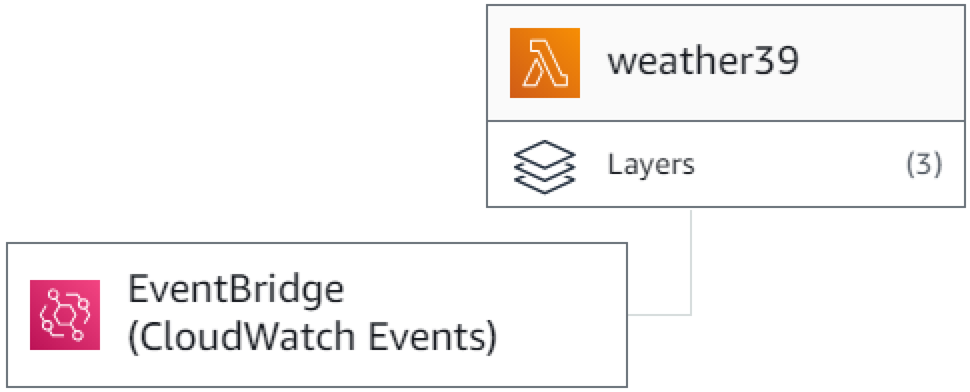
```python
import json

def lambda_handler(event, context):

    import pandas as pd
    import sqlalchemy
    import os

    #connection and key
    weakey = os.environ["weakey"]
    con = os.environ["con8"]

    # call lambda module and push to RDS
    weather_df = lambda_moudle_W.weather_in_(cities_df)
    weather_df.to_sql("weather",
                        if_exists="append",
                        con=con,
                        index=False)
    return {
        'statusCode': 200,
        'body': json.dumps('Hello from Lambda!')
    }
```

## create lambda module (ipy-analog: def function)

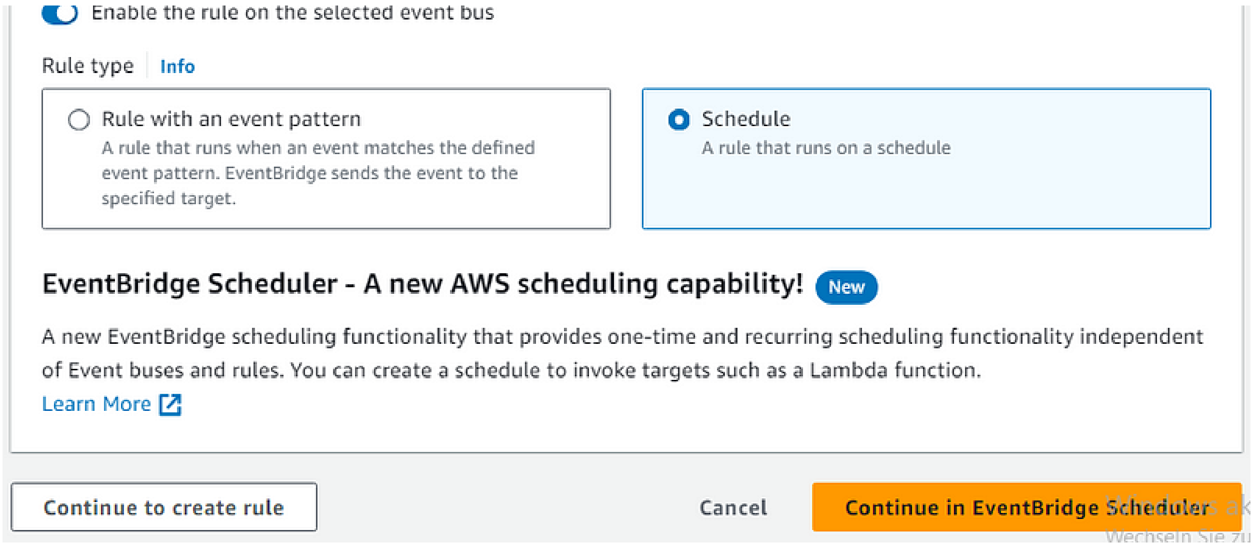


```python
def weather_in_(cities_df):

import pandas as pd
import json
import requests

cities_df = pd.read_sql_table("cities", con=con)

weather_dict = {"city_id":[], "feels_like":[], "rain_prob":[]}

    for c in cities_df["city_name"]:
        weather = requests.get(f"https://api.openweathermap.org/data/2.5/forecast?q={c}&appid={weakey}&units=metric")
        weather_json = weather.json()

     for entry in weather_json["list"]:
         weather_dict["city_id"].append(int(cities_df.loc[cities_df["city_name"] == c, "city_id"]))
         weather_dict["feels_like"].append(entry["main"]["feels_like"])
         weather_dict["rain_prob"].append(entry["pop"])

    weather_df = pd.DataFrame(weather_dict)

    return weather_df
```

# **Automatisation**



create rule in AWS eventbridge. Choice between rule and schedule.



Enable the rule on the selected event bus

Rule type | Info

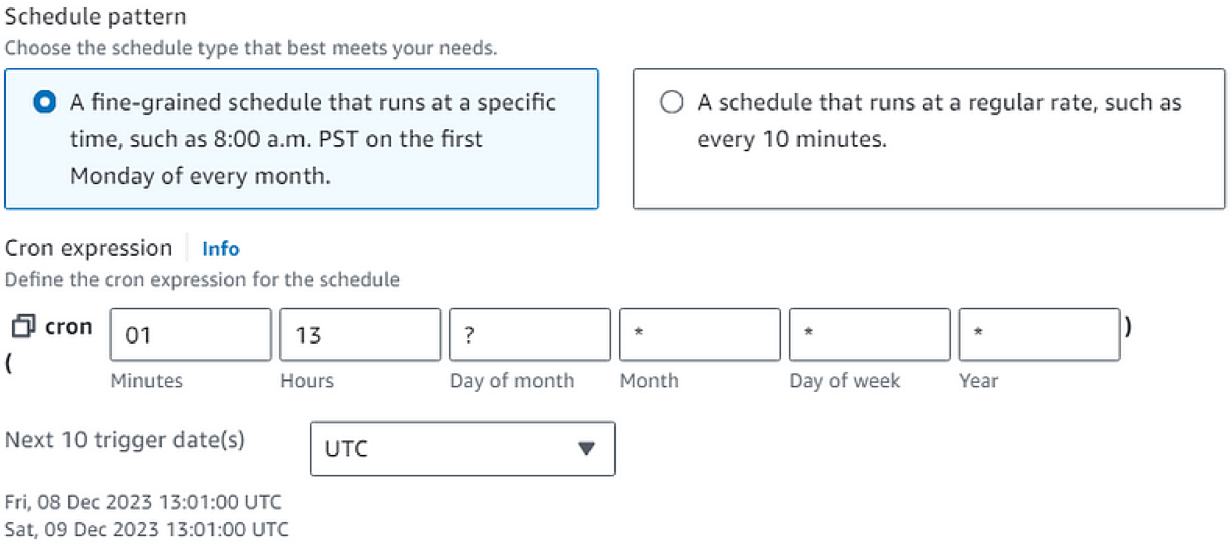
○ Rule with an event pattern
A rule that runs when an event matches the defined event pattern. EventBridge sends the event to the specified target.

● Schedule
A rule that runs on a schedule

EventBridge Scheduler - A new AWS scheduling capability! New

A new EventBridge scheduling functionality that provides one-time and recurring scheduling functionality independent of Event buses and rules. You can create a schedule to invoke targets such as a Lambda function.
Learn More

cron expressions:
https://docs.aws.amazon.com/eventbridge/latest/userguide/eb-cron-expressions.html



Schedule pattern

Choose the schedule type that best meets your needs.

● A fine-grained schedule that runs at a specific time, such as 8:00 a.m. PST on the first Monday of every month.

○ A schedule that runs at a regular rate, such as every 10 minutes.

Cron expression Info

Define the cron expression for the schedule

	Minutes	Hours	Day of month	Month	Day of week	Year	
cron (	01	13	?	*	*	*	)

Next 10 trigger date(s) UTC ▼

Fri, 08 Dec 2023 13:01:00 UTC
Sat, 09 Dec 2023 13:01:00 UTC

Whoever of you made it through: Kudos.