



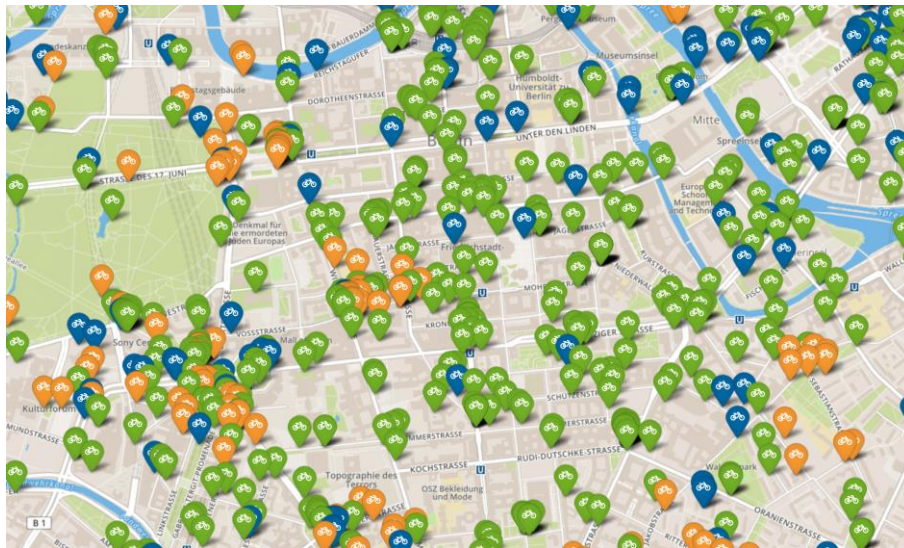
# How to store thousands of shared bike locations every 4 minutes into a database

Alexandra Kapp



# Background

- Shared bike providers have openly accessible APIs with information on current bike locations
  - *Only current location, not historical data!*



- If this data is continuously stored, analyses on movements and bike hotspots are possible

# Setup

## External Server:

So the script does not have to run continuously on your laptop

UBER  
SPACE

## Cron Job:

Makes the python script run every 4 minutes

```
# min (0 - 59)
# hour (0 - 23)
# day of month (1 - 31)
# month (1 - 12)
# day of week (0 - 6) (0 to 6 are Sunday to Saturday, or use names; 7 is also Sunday)
# * * * * * command to execute
```

„only“ every 4 min:  
bc of query time and  
time to store in DB

## Python Script:

Query bike locations and store into database



mobike



## External database Amazon RDS for PostgreSQL

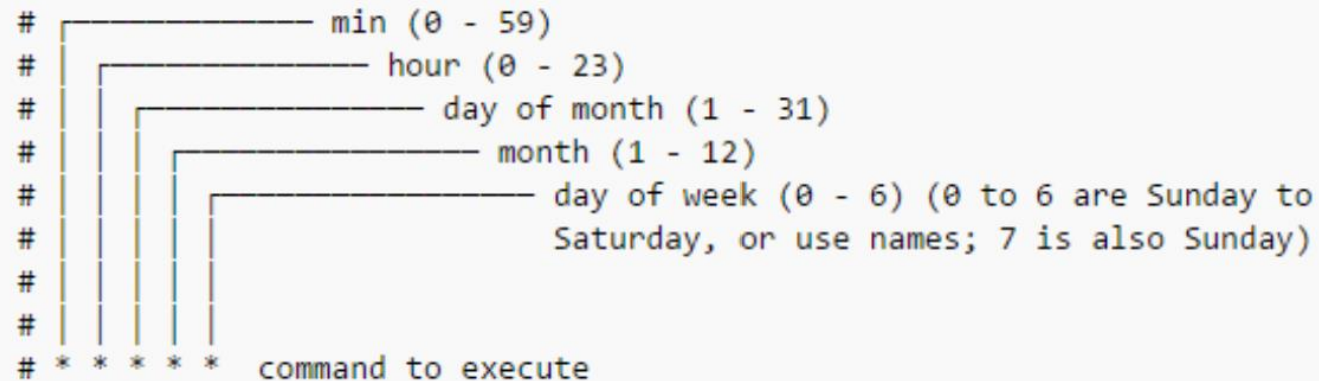


	id [PK] integer	bikeid integer	providerid integer	timestamp timestamp without time zone	latitude double precision	longitude double precision
1	141642990	816045426	2	2019-07-11 16:20:03.865712	52.552241	13.41243
2	141642989	810026544	2	2019-07-11 16:20:03.865712	52.55452	13.404591
3	141642988	810025600	2	2019-07-11 16:20:03.865712	52.551759	13.40387
4	141642987	816024768	2	2019-07-11 16:20:03.865712	52.550606	13.41033
5	141642986	810005477	2	2019-07-11 16:20:03.865712	52.550805	13.405647
6	141642985	810000434	2	2019-07-11 16:20:03.865712	52.550837	13.405675
7	141642984	816024074	2	2019-07-11 16:20:03.865712	52.551634	13.410825
8	141642983	816023109	2	2019-07-11 16:20:03.865712	52.554412	13.406741
9	141642982	810000472	2	2019-07-11 16:20:03.865712	52.551139	13.409768
10	141642981	810005354	2	2019-07-11 16:20:03.865712	52.554342	13.407722
11	141642980	810003649	2	2019-07-11 16:20:03.865712	52.552277	13.410472
12	141642979	816045508	2	2019-07-11 16:20:03.865712	52.552103	13.408455

# What is a cron job?

**Cron** is one of the most useful utility that you can find in any Unix-like operating system. It is used to schedule commands at a specific time. These scheduled commands or tasks are known as "Cron Jobs".

<https://ostechnix.com/a-beginners-guide-to-cron-jobs/>



A diagram illustrating the cron job syntax. It shows a sequence of seven lines, each starting with a '#' followed by a field. The first five fields are connected by vertical lines to their respective labels on the right. The sixth field is connected to a label that spans two lines. The seventh field is connected to a label. The labels are: 'min (0 - 59)', 'hour (0 - 23)', 'day of month (1 - 31)', 'month (1 - 12)', 'day of week (0 - 6) (0 to 6 are Sunday to Saturday, or use names; 7 is also Sunday)', and 'command to execute'.

```
# min (0 - 59)
# hour (0 - 23)
# day of month (1 - 31)
# month (1 - 12)
# day of week (0 - 6) (0 to 6 are Sunday to
# Saturday, or use names; 7 is also Sunday)
#
# * * * * * command to execute
```

```
*/4 * * * * python3 [PATH TO FOLDER]/src/query_bike_apis.py
0 8 * * * python3 [PATH TO FOLDER]/src/query_nextbike_stations.py
0 23 * * * python3 [PATH TO FOLDER]/src/clean_script.py
```

# How do I access a database?

a) Read / write into database with python script

```
# insert into database
conn = psycopg2.connect("host=" + config.dbhost + " dbname=" + config.dbname + " user=" + config.dbuser + " password=" + config.dbpassword)

cur = conn.cursor()
sql = """INSERT INTO public."bikeLocations"("id", "bikeId", "providerId", "timestamp", latitude, longitude) VALUES %s ON CONFLICT DO NOTHING"""
psycopg2.extras.execute_values(cur, sql, nextbikes, template='(DEFAULT, %s, %s, %s, %s, %s)')
psycopg2.extras.execute_values(cur, sql, lidlbikes, template='(DEFAULT, %s, %s, %s, %s, %s)')
#psycopg2.extras.execute_values(cur, sql, mobikes, template='(DEFAULT, %s, %s, %s, %s, %s)')

conn.commit()
cur.close()
conn.close()
```

# How do I access a database?

b) View data with UI

e.g. with DBeaver



	id [PK] integer	bikeid integer	providerid integer	timestamp timestamp without time zone	latitude double precision	longitude double precision
1	141642990	816045426	2	2019-07-11 16:20:03.865712	52.552241	13.41243
2	141642989	810026544	2	2019-07-11 16:20:03.865712	52.55452	13.404591
3	141642988	810025600	2	2019-07-11 16:20:03.865712	52.551759	13.40387
4	141642987	816024768	2	2019-07-11 16:20:03.865712	52.550606	13.41033
5	141642986	810005477	2	2019-07-11 16:20:03.865712	52.550805	13.405647
6	141642985	810000434	2	2019-07-11 16:20:03.865712	52.550837	13.405675
7	141642984	816024074	2	2019-07-11 16:20:03.865712	52.551634	13.410825
8	141642983	816023109	2	2019-07-11 16:20:03.865712	52.554412	13.406741
9	141642982	810000472	2	2019-07-11 16:20:03.865712	52.551139	13.409768
10	141642981	810005354	2	2019-07-11 16:20:03.865712	52.554342	13.407722
11	141642980	810003649	2	2019-07-11 16:20:03.865712	52.552277	13.410472
12	141642979	816045608	2	2019-07-11 16:20:03.865712	52.552102	13.408456

# Watch out!



- Be sure to include **checks**, if you're query runs properly:
  - a. Is the cron job working properly?
  - b. Is the data queried properly? E.g. with a change in the API the query might start failing.
  - c. Is the database up and running?
  - d. E.g. send email notifications if script is failing
- **AWS** can do a lot, but can also be very **complicated**. Usually there are good tutorials and help texts. Be sure to follow the instructions closely
- Watch out for **costs**!
  - AWS can get expensive quickly. Create a cost alert when using AWS products!

# Thank you for your attention!

**Alexandra Kapp**

alexandra.k@correlaid.org

@lxndrkp 

Project repo: <https://github.com/technologiestiftung/bike-sharing> 