CS4221 Tutorial 3:
Timeseries databases: InfluxDB

Yao LU

2024 Semester 2

National University of Singapore
School of Computing

Objective

By the end of this tutorial, you will:

- ➢ Set up InfluxDB server using Docker and monitor its status in the InfluxDB UI.
- ➢ Gain a fundamental understanding of schema design principles in InfluxDB.
- ➢ Interact with InfluxDB using the Python client and execute queries with Flux.
- ➢ Insert data into influxDB from various data sources.
- ➢ Write Flux queries to retrieve, analyze, and manipulate time series data

Setup

Notebooks: https://mlsys.io/t3.zip

Step 1: Install InfluxDB

- Download the provided docker-compose file.
- Create configuration files such as name, password and token.
- Verify InfluxDB is running through influxDB UI.

Step 2: Prepare the environment

- Make sure you have installed the *anaconda*
- Download the provided environment file and set up the virtual environment.
- Execute hello_influxdb.ipynb to make sure everything is ok.

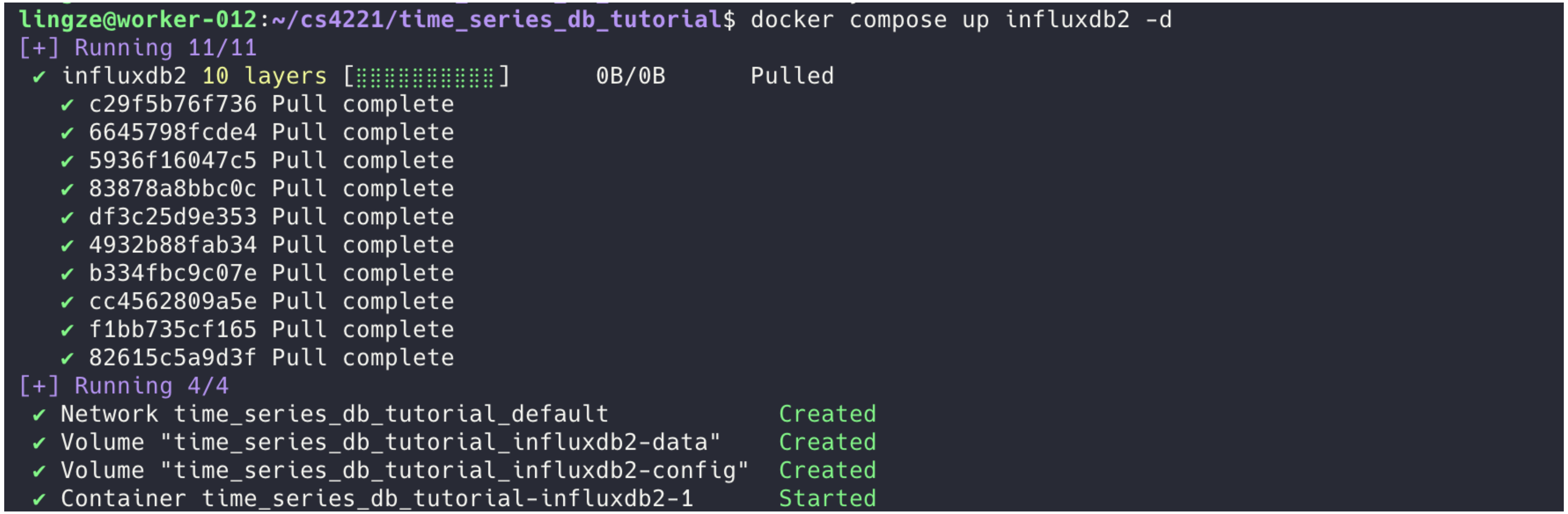
Prerequisites

Before starting, ensure you have gone through :

- InfluxDB key concepts: https://docs.influxdata.com/influxdb/cloud/reference/key-concepts
- Flux language introduction: https://docs.influxdata.com/flux/v0/

Hello-World

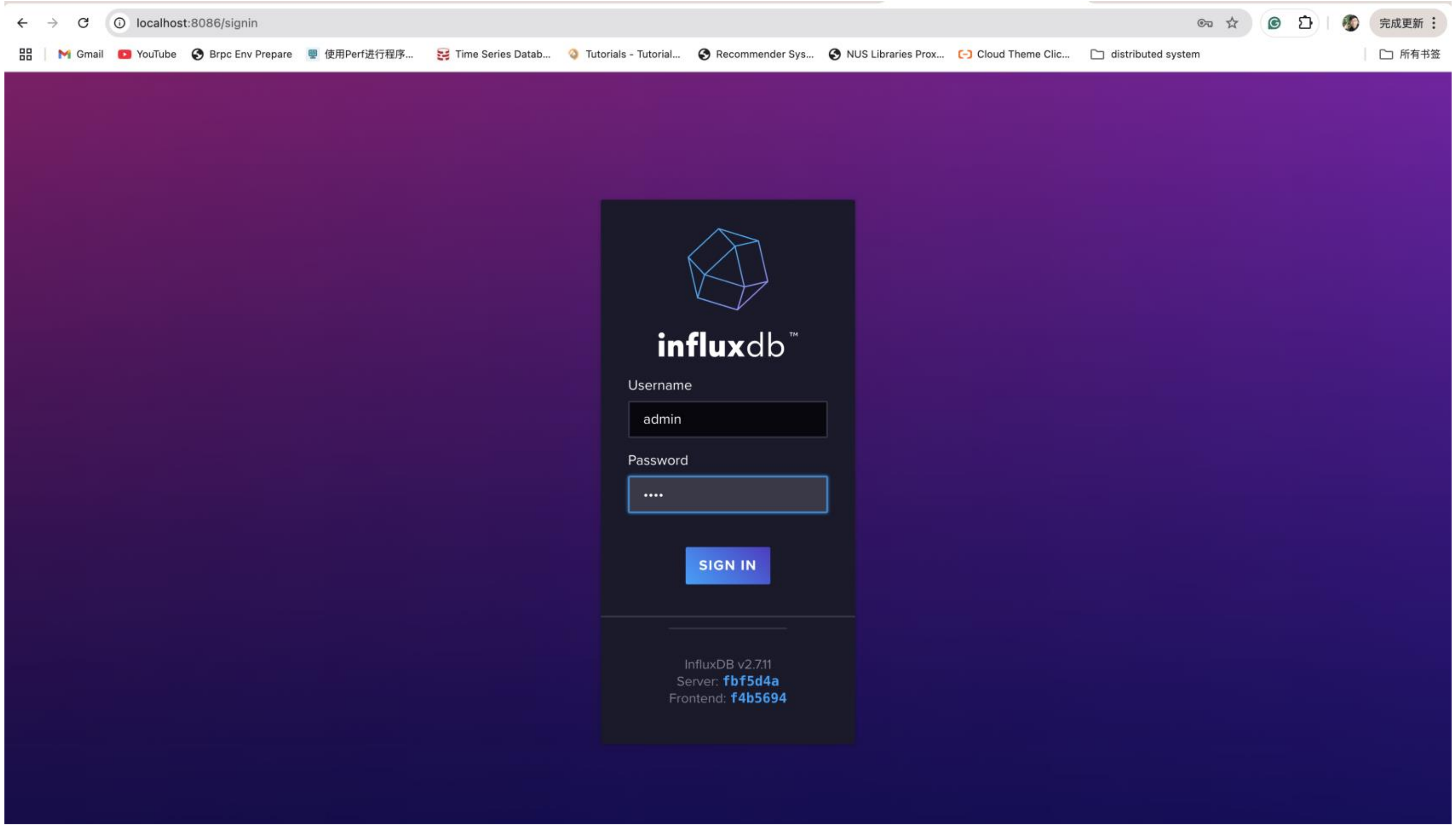
After you start docker-compose file following the README, you will get



```
lingze@worker-012:~/cs4221/time_series_db_tutorial$ docker compose up influxdb2 -d
[+] Running 11/11
 ✔ influxdb2 10 layers [⣿⣿⣿⣿⣿⣿⣿⣿⣿⣿]      0B/0B      Pulled
   ✔ c29f5b76f736 Pull complete
   ✔ 6645798fcde4 Pull complete
   ✔ 5936f16047c5 Pull complete
   ✔ 83878a8bbc0c Pull complete
   ✔ df3c25d9e353 Pull complete
   ✔ 4932b88fab34 Pull complete
   ✔ b334fbc9c07e Pull complete
   ✔ cc4562809a5e Pull complete
   ✔ f1bb735cf165 Pull complete
   ✔ 82615c5a9d3f Pull complete
[+] Running 4/4
 ✔ Network time_series_db_tutorial_default            Created
 ✔ Volume "time_series_db_tutorial_influxdb2-data"    Created
 ✔ Volume "time_series_db_tutorial_influxdb2-config"  Created
 ✔ Container time_series_db_tutorial-influxdb2-1      Started
```

Hello-World

Open the InfluxDB UI to verify server is running.



localhost:8086/signin

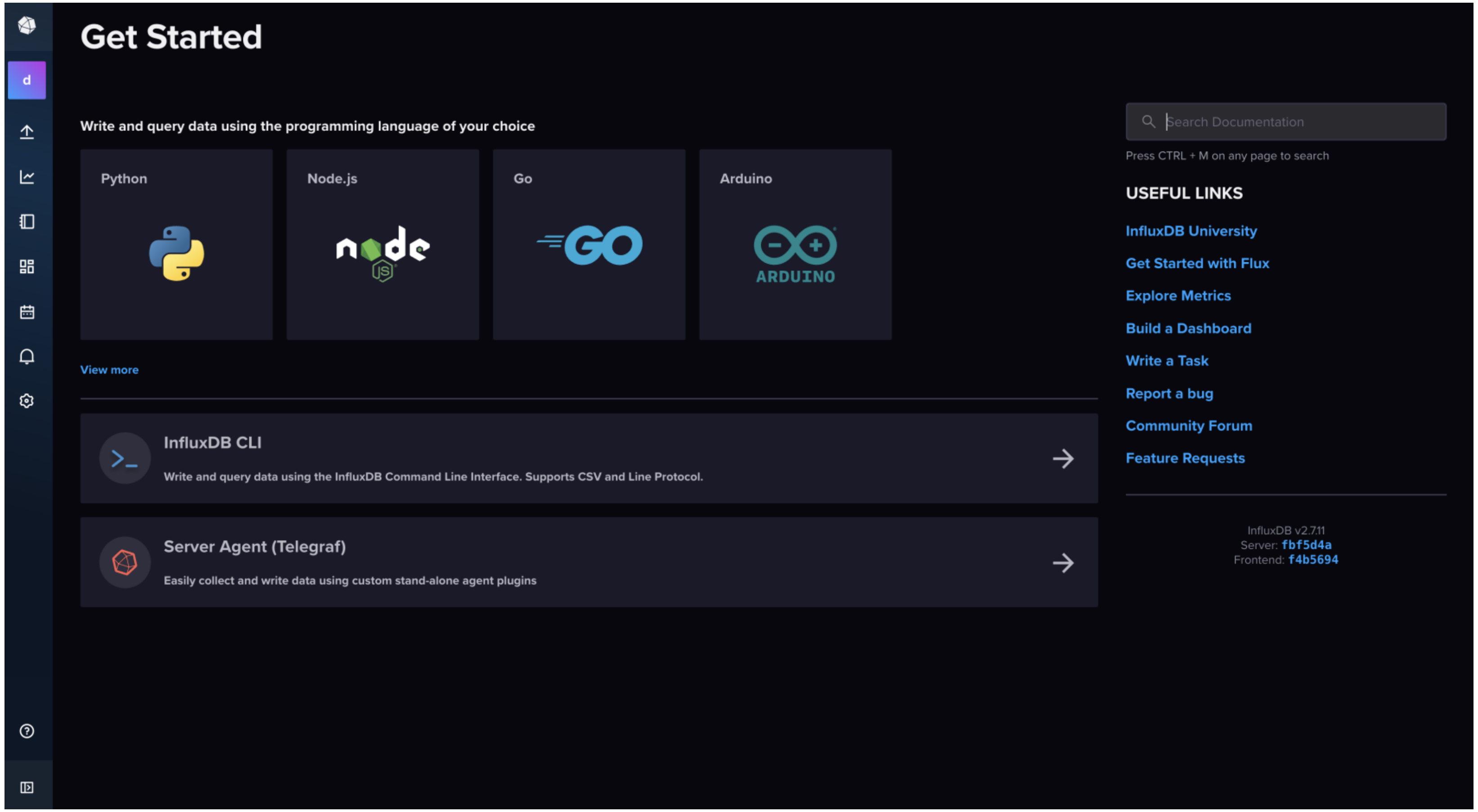
Username: admin

Password: ••••

InfluxDB v2.7.11
Server: fbf5d4a
Frontend: f4b5694

Hello-World

Open the InfluxDB UI to verify server is running.



Get Started

Write and query data using the programming language of your choice

Python

Node.js

Go

Arduino

View more

InfluxDB CLI

Write and query data using the InfluxDB Command Line Interface. Supports CSV and Line Protocol.

Server Agent (Telegraf)

Easily collect and write data using custom stand-alone agent plugins

Search Documentation

Press CTRL + M on any page to search

USEFUL LINKS

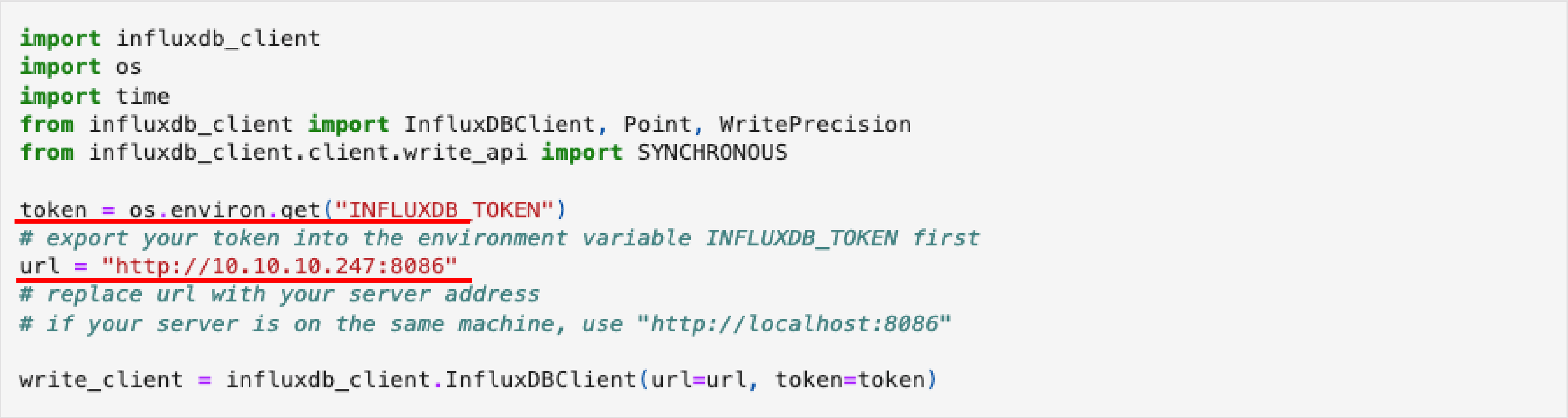
- InfluxDB University
- Get Started with Flux
- Explore Metrics
- Build a Dashboard
- Write a Task
- Report a bug
- Community Forum
- Feature Requests

InfluxDB v2.7.11
Server: fbf5d4a
Frontend: f4b5694

Hello-World

Before executing the scripts, something to note:

- Update the server address (URL) to match your configuration.
- Export your access token as an environment variable or define it directly in the code.



```python
import influxdb_client
import os
import time
from influxdb_client import InfluxDBClient, Point, WritePrecision
from influxdb_client.client.write_api import SYNCHRONOUS

token = os.environ.get("INFLUXDB_TOKEN")
# export your token into the environment variable INFLUXDB_TOKEN first
url = "http://10.10.10.247:8086"
# replace url with your server address
# if your server is on the same machine, use "http://localhost:8086"

write_client = influxdb_client.InfluxDBClient(url=url, token=token)
```

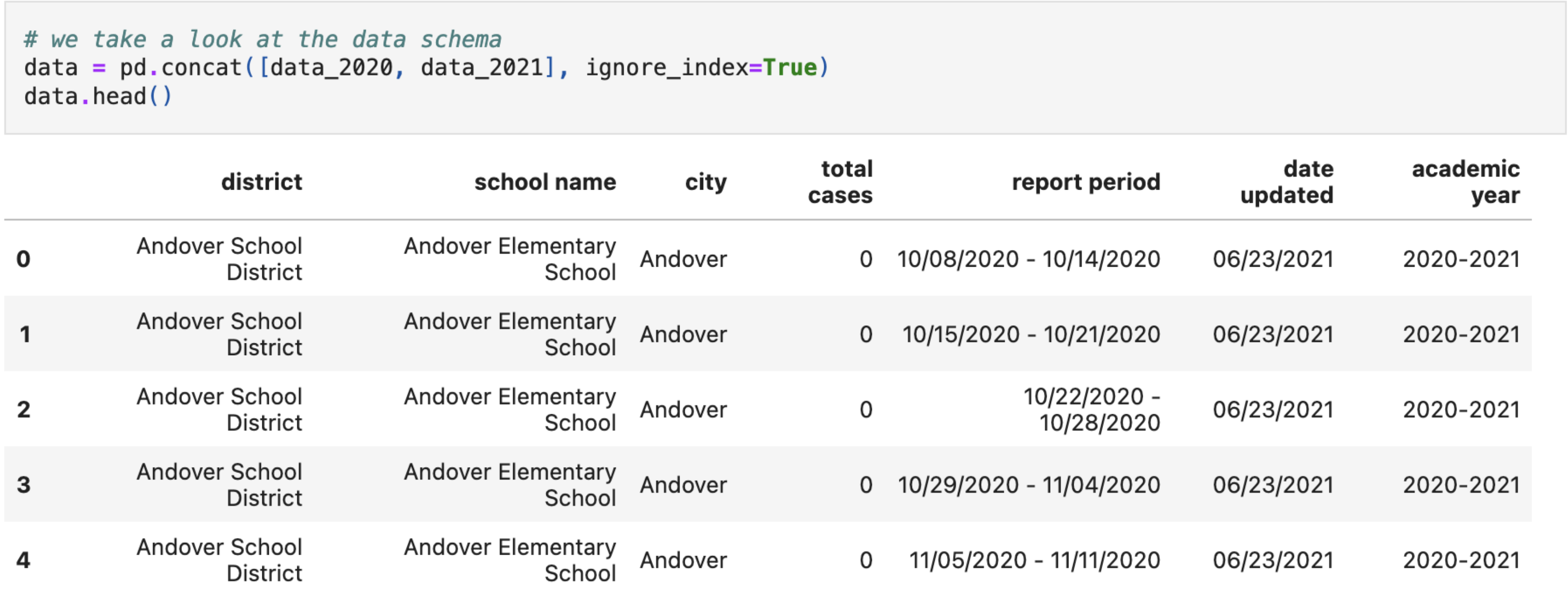
Example

Let’s try analyzing time-series data using InfluxDB.

Check the details in file *hello_influxdb_2.ipynb*.

We will work with a publicly available dataset from the State of Connecticut, which provides records of school COVID-19 cases from 2020 to 2022 in multiple formats.

After basic preprocessing, we obtain the following data schema:



```
# we take a look at the data schema
data = pd.concat([data_2020, data_2021], ignore_index=True)
data.head()
```

	district	school name	city	total cases	report period	date updated	academic year
0	Andover School District	Andover Elementary School	Andover	0	10/08/2020 - 10/14/2020	06/23/2021	2020-2021
1	Andover School District	Andover Elementary School	Andover	0	10/15/2020 - 10/21/2020	06/23/2021	2020-2021
2	Andover School District	Andover Elementary School	Andover	0	10/22/2020 - 10/28/2020	06/23/2021	2020-2021
3	Andover School District	Andover Elementary School	Andover	0	10/29/2020 - 11/04/2020	06/23/2021	2020-2021
4	Andover School District	Andover Elementary School	Andover	0	11/05/2020 - 11/11/2020	06/23/2021	2020-2021

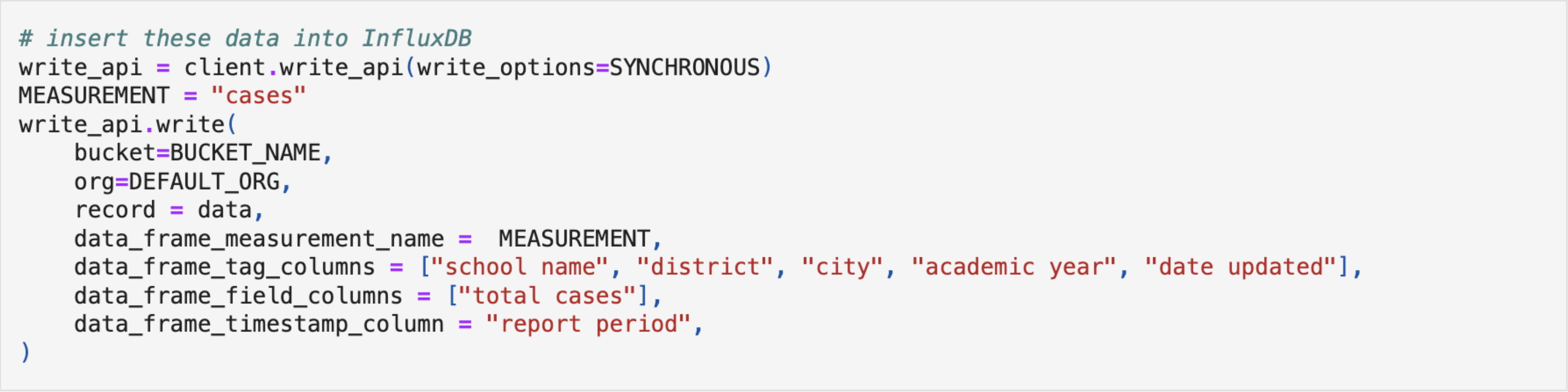
Example

Insert these data into InfluxDB following schema design principle.

Python client provide API to directly read csv data.

We define

- "school name" ,"district" ,"city" ,"academic year", "date updated" attributes as tags.
- "total cases" attributes as field.
- "report period" attributes as timestamp.



```python
# insert these data into InfluxDB
write_api = client.write_api(write_options=SYNCHRONOUS)
MEASUREMENT = "cases"
write_api.write(
    bucket=BUCKET_NAME,
    org=DEFAULT_ORG,
    record = data,
    data_frame_measurement_name =  MEASUREMENT,
    data_frame_tag_columns = ["school name", "district", "city", "academic year", "date updated"],
    data_frame_field_columns = ["total cases"],
    data_frame_timestamp_column = "report period",
)
```

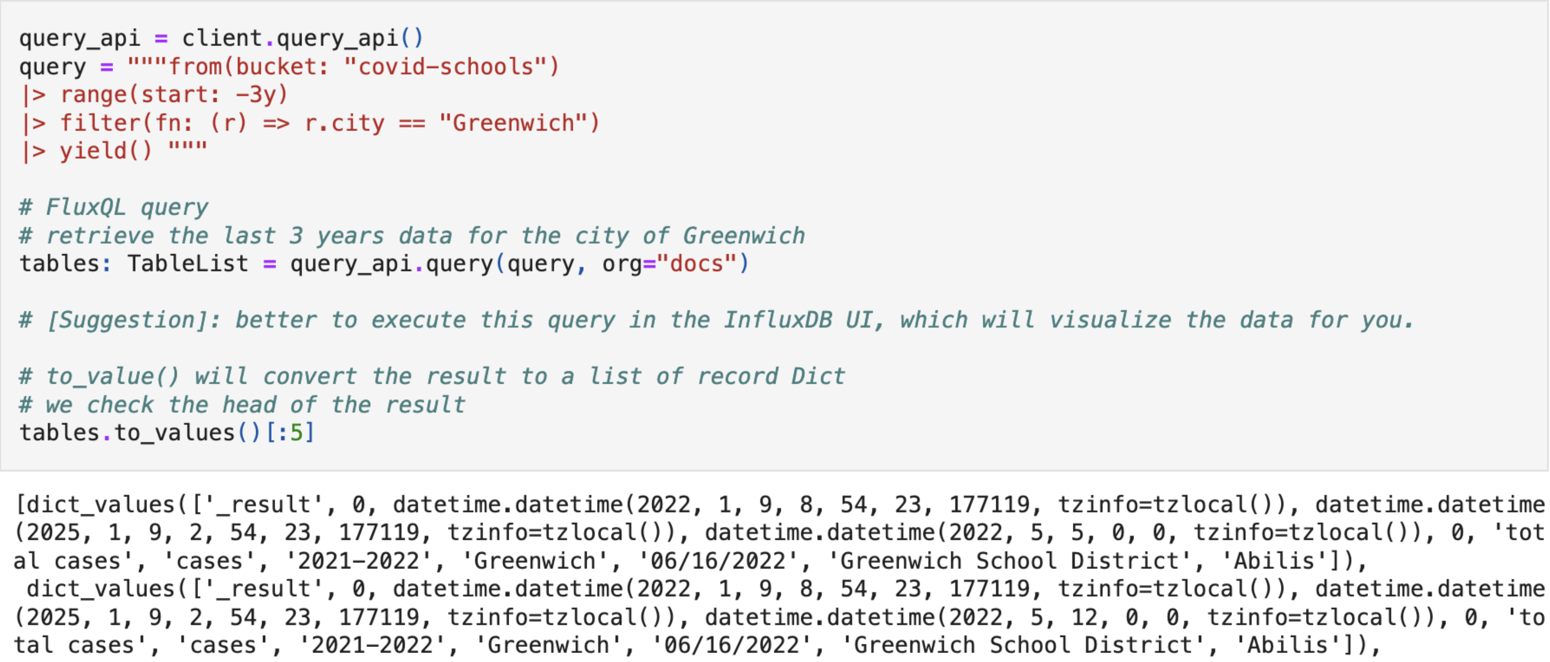
Example

After insertion, we execute several queries to do some analysis.

Target 1: retrieve the last 3 years covid-19 cases for the city of "Greenwich"

Query:

```
   from(bucket: "covid-schools")
|> range(start: -3y)
|> filter(fn: (r) => r.city == "Greenwich")
|> yield()
```



```
query_api = client.query_api()
query = """from(bucket: "covid-schools")
|> range(start: -3y)
|> filter(fn: (r) => r.city == "Greenwich")
|> yield() """

# FluxQL query
# retrieve the last 3 years data for the city of Greenwich
tables: TableList = query_api.query(query, org="docs")

# [Suggestion]: better to execute this query in the InfluxDB UI, which will visualize the data for you.

# to_value() will convert the result to a list of record Dict
# we check the head of the result
tables.to_values()[:5]
```

```
[dict_values(['_result', 0, datetime.datetime(2022, 1, 9, 8, 54, 23, 177119, tzinfo=tzlocal()), datetime.datetime
(2025, 1, 9, 2, 54, 23, 177119, tzinfo=tzlocal()), datetime.datetime(2022, 5, 5, 0, 0, tzinfo=tzlocal()), 0, 'tot
al cases', 'cases', '2021-2022', 'Greenwich', '06/16/2022', 'Greenwich School District', 'Abilis']),
 dict_values(['_result', 0, datetime.datetime(2022, 1, 9, 8, 54, 23, 177119, tzinfo=tzlocal()), datetime.datetime
(2025, 1, 9, 2, 54, 23, 177119, tzinfo=tzlocal()), datetime.datetime(2022, 5, 12, 0, 0, tzinfo=tzlocal()), 0, 'to
tal cases', 'cases', '2021-2022', 'Greenwich', '06/16/2022', 'Greenwich School District', 'Abilis']),
```

Example

After insertion, we execute several queries to do some analysis.

Target 2: count the number of cases in each city in the last three years.

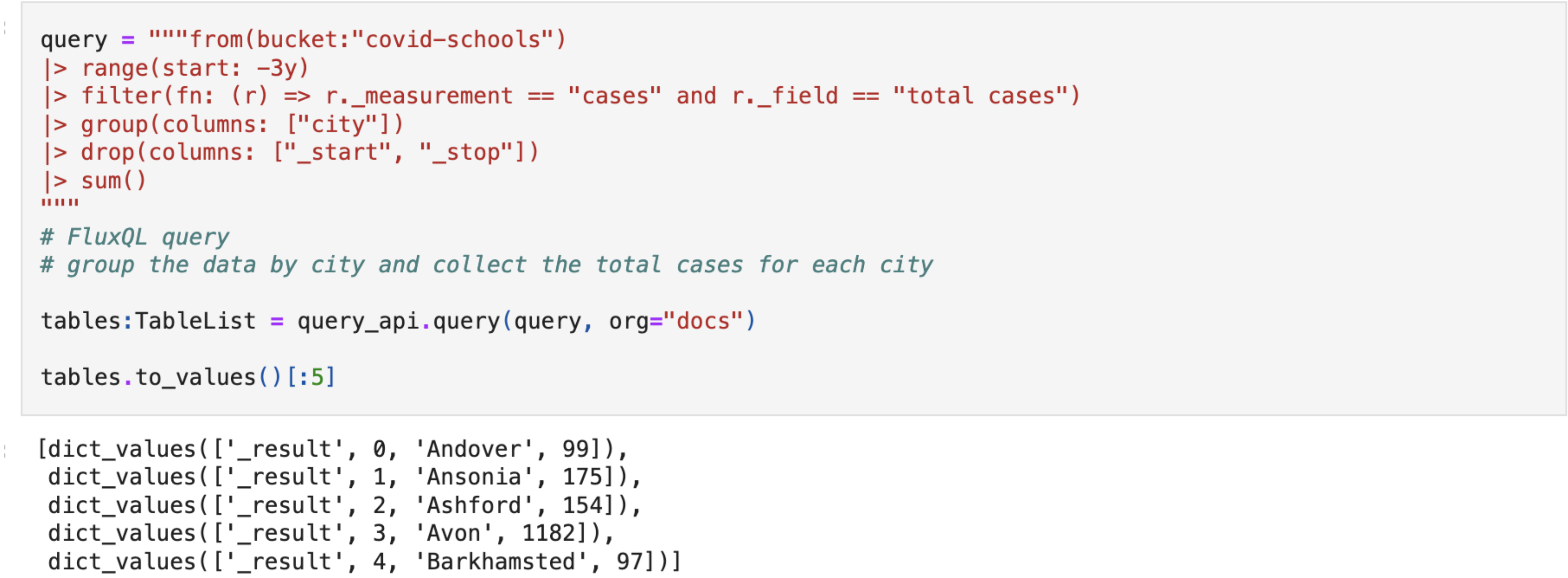
Query:

```
   from(bucket:"covid-schools")
|> range(start: -3y)
|> filter(fn: (r) => r._measurement == "cases" and r._field == "total cases")
|> group(columns: ["city"])
|> drop(columns: ["_start", "_stop"])
|> sum()
```

Example

After insertion, we execute several queries to do some analysis.

Target 2: count the number of cases in each city in the last three years.



```
query = """from(bucket:"covid-schools")
|> range(start: -3y)
|> filter(fn: (r) => r._measurement == "cases" and r._field == "total cases")
|> group(columns: ["city"])
|> drop(columns: ["_start", "_stop"])
|> sum()
"""
# FluxQL query
# group the data by city and collect the total cases for each city

tables:TableList = query_api.query(query, org="docs")

tables.to_values()[:5]
```

```
[dict_values(['_result', 0, 'Andover', 99]),
 dict_values(['_result', 1, 'Ansonia', 175]),
 dict_values(['_result', 2, 'Ashford', 154]),
 dict_values(['_result', 3, 'Avon', 1182]),
 dict_values(['_result', 4, 'Barkhamsted', 97])]
```

Example

After insertion, we execute several queries to do some analysis.

Target 3: Filter out the city with total cases less than 700 based on the result of Query 2

Query:

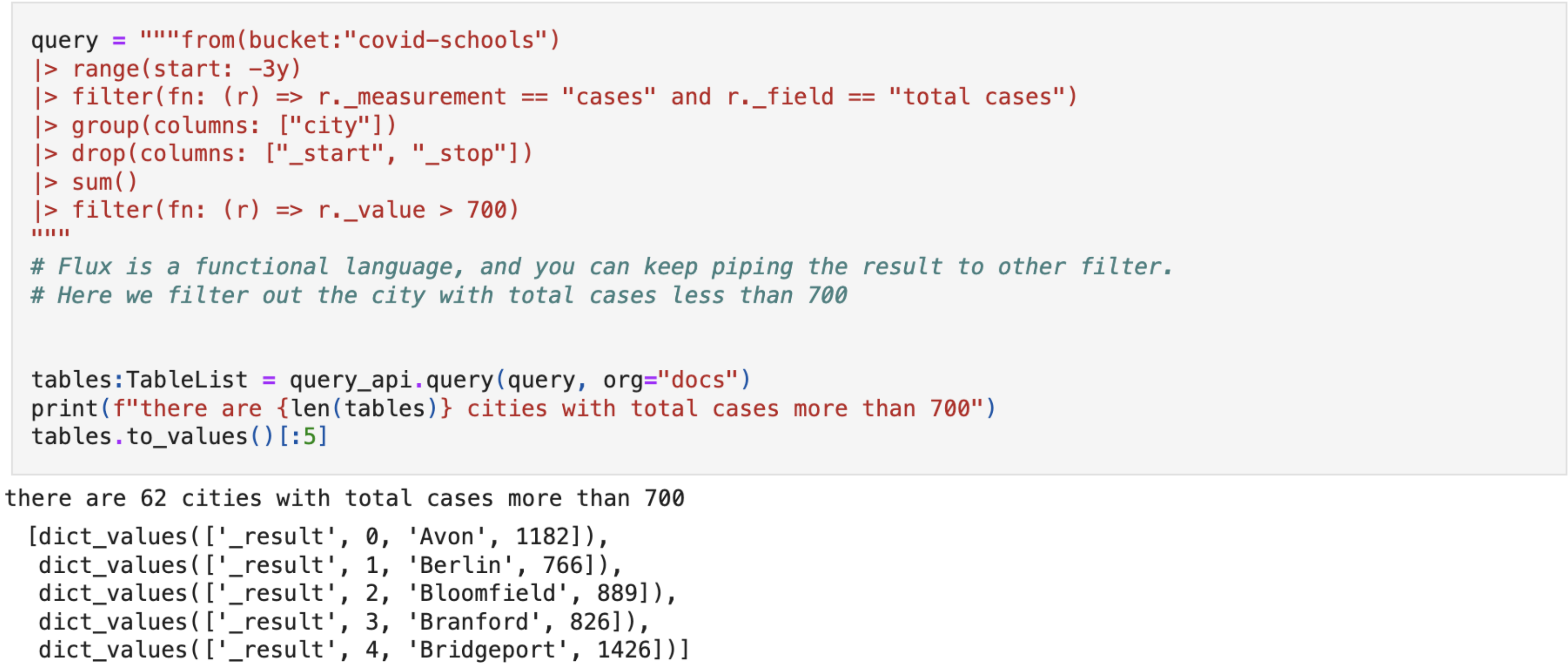
```
   from(bucket:"covid-schools")
|> range(start: -3y)
|> filter(fn: (r) => r._measurement == "cases" and r._field == "total cases")
|> group(columns: ["city"])
|> drop(columns: ["_start", "_stop"])
|> sum()
|> filter(fn: (r) => r._value > 700)
```

Example

After insertion, we execute several queries to do some analysis.

Target 3: Filter out the city with total cases less than 700 based on the result of Query 2

Query:



```
query = """from(bucket:"covid-schools")
|> range(start: -3y)
|> filter(fn: (r) => r._measurement == "cases" and r._field == "total cases")
|> group(columns: ["city"])
|> drop(columns: ["_start", "_stop"])
|> sum()
|> filter(fn: (r) => r._value > 700)
"""
# Flux is a functional language, and you can keep piping the result to other filter.
# Here we filter out the city with total cases less than 700


tables:TableList = query_api.query(query, org="docs")
print(f"there are {len(tables)} cities with total cases more than 700")
tables.to_values()[:5]
```

```
there are 62 cities with total cases more than 700
```

```
[dict_values(['_result', 0, 'Avon', 1182]),
 dict_values(['_result', 1, 'Berlin', 766]),
 dict_values(['_result', 2, 'Bloomfield', 889]),
 dict_values(['_result', 3, 'Branford', 826]),
 dict_values(['_result', 4, 'Bridgeport', 1426])]
```

Quiz

In this homework, we will analyze crime data from the Hartford Police Department.

This historical dataset includes reported crime incidents (excluding sexual assaults) that occurred in the City of Hartford from January 1, 2005, to May 18, 2021.

Check the details in file *hello_influxdb_3.ipynb*.

We have finished the data loading and preprocessing steps, please finish above tasks. During the process, carefully define tags, fields, and timestamps to ensure your queries are optimized for efficiency.

- Question 1: Retrieve top-10 most common types of cases in all time.
- Question 2: Count the Number of cases with code "1901" over time grouped by week.
- Question 3: Get the latest incident (most recent time) for the "ucr_1_code" case.