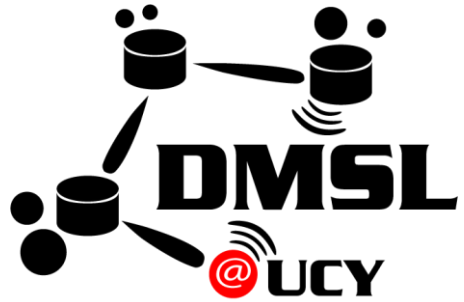
EPL646 – Advanced Topics in Databases
Introduction to InfluxDB

Christoforos Panayiotou
http://www.cs.ucy.ac.cy/~dzeina/courses/epl646/labs/lab.html



Install InfluxDB OSS v2

- Version 3 is still in alpha so we'll use version 2
- Go to https://docs.influxdata.com/influxdb/v2/install/ and follow the instruction for your OS
  - Admin rights are needed
- After installing, start the InfluxDB deamon
  - If successful, you can view the InfluxDB UI at http://localhost:8086
  - After the initial configuration (see next slide) you will be presented with your API token – Make sure to save it!
- Continue to install the InfluxDB CLI
  - Follow the instructions found here: https://docs.influxdata.com/influxdb/v2/tools/influx-cli/

Initial setup process

- During the initial setup process you set the following
  - An organization with the name you provide
  - A bucket with the name you provide
  - An admin authorization with the following properties
    - The username and password that you provide
    - An ***API Operator*** token
  - Read-write permissions for all resources in the InfluxDB instance
- Create an ***All Access API*** token
  - The API Operator token has all permissions to manage everything in your InfluxDB instance
  - An ***All Access*** token has narrower scope
  - Navigate to *Load Data* → *API Tokens* using the left navigation bar and click on *"+ Generate API token"* and select *"All Access API Token"*
  - Enter a description for the API token, save it and copy it for safe keeping and usage

Data organization in InfluxDB

- The InfluxDB data model organizes time series data into ***buckets*** and ***measurements***
- **Bucket**: Named location where time series data is stored
  - A bucket can contain multiple ***measurements***
- **Measurement**: Logical grouping for time series data
  - All ***points*** in a given measurement should have the same ***tags***
  - A measurement contains multiple ***tags*** and ***fields***
- **Tags**: Key-value pairs with values that differ, but do not change often
  - Tags are meant for storing metadata for each point
  - Something to identify the source of the data like host, location, station, etc.
- **Fields**: Key-value pairs with values that change over time
  - E.g., temperature, pressure, stock price, etc.
- **Timestamp**: Timestamp associated with the data
  - When stored on disk and queried, all data is ordered by time
- **Point**: Single data record identified by its measurement, tag keys, tag values, field key, and timestamp
- **Series**: A group of points with the same measurement, tag keys, and tag values

Example InfluxDB query results

_time	_measurement	city	country	_field	_value
2022-01-01T12:00:00Z	weather	London	UK	temperature	12.0
2022-02-01T12:00:00Z	weather	London	UK	temperature	12.1
2022-03-01T12:00:00Z	weather	London	UK	temperature	11.5
2022-04-01T12:00:00Z	weather	London	UK	temperature	5.9

Point

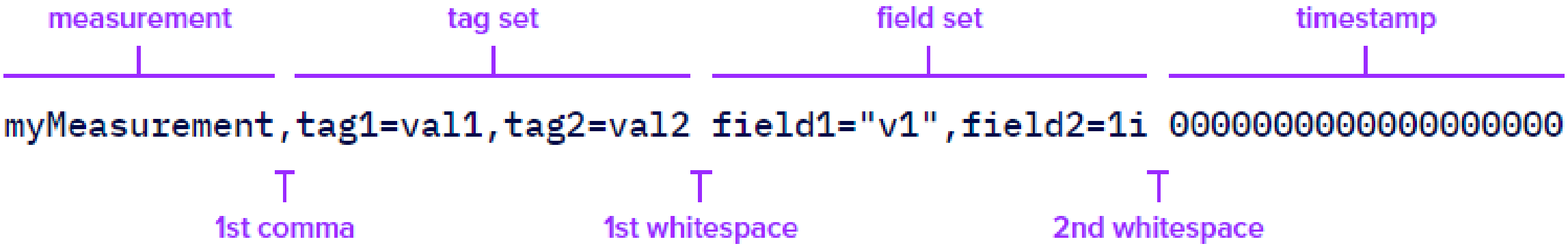
Series

Writing data

- InfluxDB provides many different options for ingesting or writing data, including the following:
  - Influx user interface (UI)
  - InfluxDB HTTP API
  - influx CLI
  - Telegraf
  - InfluxDB client libraries
    - Arduino, C#, Dart, Go, Java, JavaScript for browsers, Kotlin, Node.js, PHP, ***Python***, R, Ruby, Scala, Swift
- Line protocol
  - All data written to InfluxDB is written using line protocol
  - A text-based format that lets you provide the necessary information to write a data point to InfluxDB

Line protocol

- Line protocol elements
  - ***measurement**: String that identifies the measurement to store the data in
  - **tag set**: Comma-delimited list of key value pairs, each representing a tag
    - Tag keys and values are unquoted strings
    - Spaces, commas, and equal characters must be escaped
  - ***field set**: Comma-delimited list key value pairs, each representing a field
    - Field keys are unquoted strings
    - Spaces and commas must be escaped
    - Field values can be strings (quoted), floats, integers, unsigned integers, or booleans
  - **timestamp**: Unix timestamp associated with the data
    - InfluxDB supports up to nanosecond precision
    - If the precision of the timestamp is not in nanoseconds, you must specify the precision when writing the data to InfluxDB



Example data line protocol

- Consider a use case where you collect data from sensors in your home
  - Each sensor collects temperature, humidity, and carbon monoxide readings
- To collect this data, use the following schema:
  - measurement: home
  - tags
    - room: Living Room or Kitchen
      - **Notice the escaped space for *Living Room*!**
  - fields
    - temp: temperature in °C (float)
    - hum: percent humidity (float)
    - co: carbon monoxide in parts per million (integer)
  - timestamp: Unix timestamp in second precision

```
home,room=Living\ Room temp=21.1,hum=35.9,co=0i 1641024000
home,room=Kitchen temp=21.0,hum=35.9,co=0i 1641024000
home,room=Living\ Room temp=21.4,hum=35.9,co=0i 1641027600
home,room=Kitchen temp=23.0,hum=36.2,co=0i 1641027600
home,room=Living\ Room temp=21.8,hum=36.0,co=0i 1641031200
home,room=Kitchen temp=22.7,hum=36.1,co=0i 1641031200
home,room=Living\ Room temp=22.2,hum=36.0,co=0i 1641034800
home,room=Kitchen temp=22.4,hum=36.0,co=0i 1641034800
home,room=Living\ Room temp=22.2,hum=35.9,co=0i 1641038400
home,room=Kitchen temp=22.5,hum=36.0,co=0i 1641038400
home,room=Living\ Room temp=22.4,hum=36.0,co=0i 1641042000
home,room=Kitchen temp=22.8,hum=36.5,co=1i 1641042000
home,room=Living\ Room temp=22.3,hum=36.1,co=0i 1641045600
home,room=Kitchen temp=22.8,hum=36.3,co=1i 1641045600
home,room=Living\ Room temp=22.3,hum=36.1,co=1i 1641049200
home,room=Kitchen temp=22.7,hum=36.2,co=3i 1641049200
home,room=Living\ Room temp=22.4,hum=36.0,co=4i 1641052800
home,room=Kitchen temp=22.4,hum=36.0,co=7i 1641052800
home,room=Living\ Room temp=22.6,hum=35.9,co=5i 1641056400
home,room=Kitchen temp=22.7,hum=36.0,co=9i 1641056400
home,room=Living\ Room temp=22.8,hum=36.2,co=9i 1641060000
home,room=Kitchen temp=23.3,hum=36.9,co=18i 1641060000
home,room=Living\ Room temp=22.5,hum=36.3,co=14i 1641063600
home,room=Kitchen temp=23.1,hum=36.6,co=22i 1641063600
home,room=Living\ Room temp=22.2,hum=36.4,co=17i 1641067200
home,room=Kitchen temp=22.7,hum=36.5,co=26i 1641067200
```

Query data with Flux

- Flux is a functional scripting language that lets you query and process data from InfluxDB and other data sources
- When querying InfluxDB with Flux, there are three primary functions you use:
  - **from()**: Queries data from an InfluxDB bucket
  - **range()**: Filters data based on time bounds
    - Flux requires "bounded" queries—queries limited to a specific time range
  - **filter()**: Filters data based on column values
    - Each row is represented by r and each column is represented by a property of r
    - You can apply multiple subsequent filters
- Pipe-forward operator: Flux uses the pipe-forward operator (|>) to pipe the output of one function as input the next function
- The following query returns the co, hum, and temp fields stored in the *home* measurement with timestamps between *2022-01-01T08:00:00Z* and *2022-01-01T20:00:01Z*

```
from(bucket: "get-started")
    |> range(start: 2022-01-01T08:00:00Z, stop: 2022-01-01T20:00:01Z)
    |> filter(fn: (r) => r._measurement == "home")
    |> filter(fn: (r) => r._field== "co" or r._field == "hum" or r._field == "temp")
```

Groups and Aggregate or select specific data

- Use the **group()** function to regroup your data by specific column values in preparation for further processing

```
from(bucket: "get-started")
    |> range(start: 2022-01-01T08:00:00Z, stop: 2022-01-01T20:00:01Z)
    |> filter(fn: (r) => r._measurement == "home")
    |> group(columns: ["room", "_field"])
```

  - For more information about how data is grouped see the Flux data model documentation (https://docs.influxdata.com/flux/v0/get-started/data-model/)

- Use Flux aggregate or selector functions to return aggregate or selected values from each input table
  - https://docs.influxdata.com/flux/v0/function-types/#aggregates

Practice

- Follow the get started guide to add sample data to your InfluxDB (https://docs.influxdata.com/influxdb/v2/get-started/write/)
- Continue the guide to read the sample data from your InfluxDB

Write data to InfluxDB with Python

- Install the InfluxDB Python library: ***pip install influxdb-client***
- In your Python program, import the InfluxDB client library and use it to write data to InfluxDB

```
import influxdb_client
from influxdb_client.client.write_api import SYNCHRONOUS
```

- Define a few variables with the name of your bucket, organization, and token

```
bucket = "<my-bucket>"        # The bucket in which you want to write data
org = "<my-org>"              # The organization in which you want to write data
token = "<my-token>"          # An All Access API token predefined in your InfluxDB instance
url= "http://localhost:8086"  # The URL of your InfluxDB instance
```

- Instantiate the client
  - The **InfluxDBClient** object takes three named parameters: ***url***, ***org***, and ***token***

```
client = influxdb_client.InfluxDBClient(url=url, token=token,  org=org)
```

- Instantiate a write client using the client object and the **write_api** method
  - Use the **write_api** method to configure the *write_api* object

```
write_api = client.write_api(write_options=SYNCHRONOUS)
```

- Create a *point* object and write it to InfluxDB using the **write** method of the API *write_api* object
  - The write method requires three parameters: ***bucket***, ***org***, and ***record***

```
p = influxdb_client.Point("my_measurement").tag("location", "Prague").field("temperature", 25.3)
write_api.write(bucket=bucket, org=org, record=p)
```

Query data from InfluxDB with Python

- Instantiate the query client

```
query_api = client.query_api()
```

- Create a Flux query, and then format it as a Python string

```
query = 'from(bucket:"my-bucket")\
|> range(start: -10m)\
|> filter(fn:(r) => r._measurement == "my_measurement")\
|> filter(fn:(r) => r.location == "Prague")\
|> filter(fn:(r) => r._field == "temperature")'
```

  - The query client sends the Flux query to InfluxDB and returns a Flux object with a table structure

- Pass to the **query()** method two named parameters: ***org*** and ***query***

```
result = query_api.query(org=org, query=query)
```

- Iterate through the tables and records in the Flux object
  - Use the **get_value()** method to return values and the **get_field()** method to return fields

```
results = []
for table in result:
    for record in table.records:
        results.append((record.get_field(), record.get_value()))

print(results)
```

[(temperature, 25.3)]

Query data from InfluxDB with Python

- The Flux object provides the following methods for accessing your data:
  - **get_measurement()**: Returns the measurement name of the record
  - **get_field()**: Returns the field name
  - **get_value()**: Returns the actual field value
  - **values**: Returns a map of column values
  - **values.get("<your tag>")**: Returns a value from the record for given column
  - **get_time()**: Returns the time of the record
  - **get_start()**: Returns the inclusive lower time bound of all records in the current table
  - **get_stop()**: Returns the exclusive upper time bound of all records in the current table

Importing CSV files to InfluxDB with Python

- CSV annotations
  - Annotations, either in the CSV file itself or provided as CLI options, are properties of the columns in the CSV file. They describe how to translate each column into either a measurement name, tag, field, or timestamp
  - The following demonstrates adding annotations to our example data to a file:

```
#datatype measurement,tag,double,double,dateTime:RFC3339
name,building,temperature,humidity,time
iot-devices,5a,72.3,34.1,2022-10-01T12:01:00Z
iot-devices,5a,72.1,33.8,2022-10-02T12:01:00Z
iot-devices,5a,72.2,33.7,2022-10-03T12:01:00Z
```

- The datatypes in this example are specified as follows:
  - measurement: states which column to use as the measurement name
    - If no column exists, this can also be specified as a header via the CLI
  - tag: specifies which column or columns are to be treated as string tag data
    - These are optional, but help with querying and indexing data in InfluxDB
  - double: is used on two columns to specify that they contain double data types
  - dateTime: specifies that the final column contains the timestamp of the record (format used is RFC3339)
- Users can also specify additional data types for fields: double, long, unsignedLong, Boolean, string, ignored (used if a column is not useful or required)
- For timestamps, there are built-in parsing capabilities for: RFC3339 (e.g. 2020-01-01T00:00:00Z), RFC3339Nano (e.g. 2020-01-01T00:00:00.000000000Z), Unix timestamps (e.g. 1577836800000000000)
  - If the timestamp is not in one of these formats, then users need to specify the format of the timestamp themselves (e.g. dateTime:2006-01-02) as part of the annotation using Go reference time

Importing CSV files to InfluxDB with Python

- If a user has a very large CSV file or files they want to push to InfluxDB, Pandas provides an easy way to read a CSV file with headers quickly
  - Combined with the built-in functionality of the InfluxDB client libraries to write Pandas DataFrames, a user can read a CSV in chunks and then send those chunks into InfluxDB
- In the following example, a user is reading a CSV containing thousands of rows containing VIX stock data:

```
symbol,open,high,low,close,timestamp
vix,13.290000,13.910000,13.290000,13.570000,1359356400000000000
vix,13.870000,13.880000,13.040000,13.310000,1359442800000000000
vix,13.640000,14.330000,13.600000,14.320000,1359529200000000000
```

- To avoid reading the entire file into memory, the user can take advantage of Pandas' **read_csv** function, which will read the column names based on the CSV header and chunk the file into 1,000-row chunks
  - Finally, use the InfluxDB client library to send those groups of 1,000 rows to InfluxDB after specifying the measurement, tag, and timestamp columns (see next slide)

Importing CSV files to InfluxDB with Python

```
from influxdb_client import InfluxDBClient, WriteOptions
import pandas as pd

with InfluxDBClient.from_env_properties() as client:
    for df in pd.read_csv("data.csv", chunksize=1000):
        with client.write_api() as write_api:
            try:
                write_api.write(
                    record=df,
                    bucket="my-bucket",
                    data_frame_measurement_name="stocks",
                    data_frame_tag_columns=["symbol"],
                    data_frame_timestamp_column="date",
                )
            except InfluxDBError as e:
                print(e)
```

Practice

- Check https://github.com/influxdata/influxdb2-sample-data and Insert a large data set to your InfluxDB
- Write some simple queries to view your data
- Write some aggregate queries for your data

Questions?

http://www.cs.ucy.ac.cy/~dzeina/courses/epl646/labs/lab.html