Working with the InfluxDB CLI and Query Language

Michael Desa & Jack Zampolin

By the end of this section, participants will be able to...

1. Navigate an InfluxDB instance using the CLI
2. Query InfluxDB using the InfluxDB CLI
3. Understand the structure of the data that is returned from a query
4. Articulate what InfluxQL can do
5. Create novel queries of their own

Before we get into things let's add our dataset

```
$ curl https://s3-us-west-2.amazonaws.com/influx-sample-data/NOAA.txt > NOAA_data.txt
```

Import our Dataset

```
influx -import -path=NOAA_data.txt -precision s
```

```
~$ influx -import -path=NOAA_data.txt -precision s
2016/03/09 22:38:34 Processed 1 commands
2016/03/09 22:38:34 Processed 76290 inserts
2016/03/09 22:38:34 Failed 0 inserts
~$
```

influxdata

CLI Tips and Tricks

influxdata

Why Use the CLI?

- It’s always there - comes with every InfluxDB package
- It’s better than the web admin UI
- It’s easier than just using cURL
- Provides a number of convenient utilities for interacting with the database

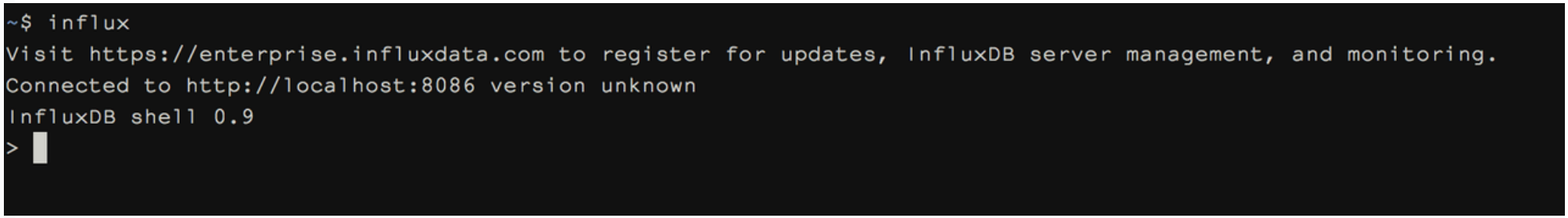
InfluxDB CLI Options

```
$ influx -h
```

```
user@sf7:~$ influx -h
Usage of influx:
  -version
       Display the version and exit.
  -host 'host name'
       Host to connect to.
  -port 'port #'
       Port to connect to.
  -database 'database name'
       Database to connect to the server.
  -password 'password'
      Password to connect to the server.  Leaving blank will prompt for password (--password '').
  -username 'username'
       Username to connect to the server.
  -ssl
        Use https for requests.
  -unsafeSsl
        Set this when connecting to the cluster using https and not use SSL verification.
  -execute 'command'
       Execute command and quit.
  -format 'json|csv|column'
       Format specifies the format of the server responses:  json, csv, or column.
  -precision 'rfc3339|h|m|s|ms|u|ns'
       Precision specifies the format of the timestamp:  rfc3339, h, m, s, ms, u or ns.
  -consistency 'any|one|quorum|all'
       Set write consistency level: any, one, quorum, or all
  -pretty
       Turns on pretty print for the json format.
  -import
       Import a previous database export from file
  -pps
       How many points per second the import will allow.  By default it is zero and will not throttle importing.
  -path
       Path to file to import
  -compressed
       Set to true if the import file is compressed
```

Pop into the CLI

```
$ influx
```



influxdata

Verify that we created the NOAA DB

```
SHOW DATABASES
```

```
> show databases
name: databases
---------------
name
_internal
NOAA_water_database
```

Use the NOAA database

```
USE NOAA_water_database
```

```
> use NOAA_water_database
Using database NOAA_water_database
>
```

Basic Select Statement

```
SELECT * FROM h2o_quality LIMIT 10
```

```
> SELECT * FROM h2o_quality LIMIT 10
name: h2o_quality
------------------
time                    index   location        randtag
1439856000000000000     99      santa_monica    2
1439856000000000000     41      coyote_creek    1
1439856360000000000     56      santa_monica    2
1439856360000000000     11      coyote_creek    3
1439856720000000000     65      santa_monica    3
1439856720000000000     38      coyote_creek    1
1439857080000000000     57      santa_monica    3
1439857080000000000     50      coyote_creek    1
1439857440000000000     8       santa_monica    3
1439857440000000000     35      coyote_creek    3
```

Changing the Precision on Time

Precision specifies the format of the timestamp:
rfc3339, h, m, s, ms, u or ns.

```
$ influx -precision rfc3339
```

or

```
$ influx
> precision rfc3339
```

Basic Select Statement (rfc3339)

```
PRECISION rfc3339
```

```
> SELECT * FROM h2o_quality LIMIT 10
name: h2o_quality
-----------------
time                    index   location        randtag
2015-08-18T00:00:00Z    41      coyote_creek    1
2015-08-18T00:00:00Z    99      santa_monica    2
2015-08-18T00:06:00Z    11      coyote_creek    3
2015-08-18T00:06:00Z    56      santa_monica    2
2015-08-18T00:12:00Z    65      santa_monica    3
2015-08-18T00:12:00Z    38      coyote_creek    1
2015-08-18T00:18:00Z    57      santa_monica    3
2015-08-18T00:18:00Z    50      coyote_creek    1
2015-08-18T00:24:00Z    8       santa_monica    3
2015-08-18T00:24:00Z    35      coyote_creek    3
```

Basic Select Statement (hours)

```
PRECISION h
```

```
> SELECT * FROM h2o_quality LIMIT 10
name: h2o_quality
-----------------
time    index   location        randtag
399960  41      coyote_creek    1
399960  99      santa_monica    2
399960  11      coyote_creek    3
399960  56      santa_monica    2
399960  65      santa_monica    3
399960  38      coyote_creek    1
399960  57      santa_monica    3
399960  50      coyote_creek    1
399960  8       santa_monica    3
399960  35      coyote_creek    3
```

Changing the Format of the results

Format specifies the format of the server responses:
json, csv, or column.

```
$ influx -format json

or

$ influx
> format json
```

Basic Select Statement (json)

```
FORMAT JSON
```

```
> SELECT * FROM h2o_quality LIMIT 10
{"results":[{"series":[{"name":"h2o_quality","columns":["time","index","location","randtag"]
,"values":[[399960,41,"coyote_creek","1"],[399960,99,"santa_monica","2"],[399960,11,"coyote_
creek","3"],[399960,56,"santa_monica","2"],[399960,38,"coyote_creek","1"],[399960,65,"santa_
monica","3"],[399960,50,"coyote_creek","1"],[399960,57,"santa_monica","3"],[399960,35,"coyot
e_creek","3"],[399960,8,"santa_monica","3"]]}]}]}
```

Thats awful!

influxdata

Basic Select Statement (json pretty)

PRETTY

```
> SELECT * FROM h2o_quality LIMIT 10
{
    "results": [
        {
            "series": [
                {
                    "name": "h2o_quality",
                    "columns": [
                        "time",
                        "index",
                        "location",
                        "randtag"
                    ],
                    "values": [
                        [
                            399960,
                            99,
                            "santa_monica",
                            "2"
                        ],
```

influxdata

Basic Select Statement (csv)

```
FORMAT csv
```

```
> SELECT * FROM h2o_quality LIMIT 10
name,time,index,location,randtag
h2o_quality,399960,99,santa_monica,2
h2o_quality,399960,41,coyote_creek,1
h2o_quality,399960,56,santa_monica,2
h2o_quality,399960,11,coyote_creek,3
h2o_quality,399960,65,santa_monica,3
h2o_quality,399960,38,coyote_creek,1
h2o_quality,399960,57,santa_monica,3
h2o_quality,399960,50,coyote_creek,1
h2o_quality,399960,8,santa_monica,3
h2o_quality,399960,35,coyote_creek,3
```

Using the -execute Flag

```
influx -database NOAA_water_database —execute [query]
```

```
~$ influx -database NOAA_water_database -execute "SELECT * FROM h2o_quality LIMIT 10"
name: h2o_quality
-----------------
time                    index   location        randtag
1439856000000000000     41      coyote_creek    1
1439856000000000000     99      santa_monica    2
1439856360000000000     11      coyote_creek    3
1439856360000000000     56      santa_monica    2
1439856720000000000     38      coyote_creek    1
1439856720000000000     65      santa_monica    3
1439857080000000000     50      coyote_creek    1
1439857080000000000     57      santa_monica    3
1439857440000000000     35      coyote_creek    3
1439857440000000000     8       santa_monica    3
```

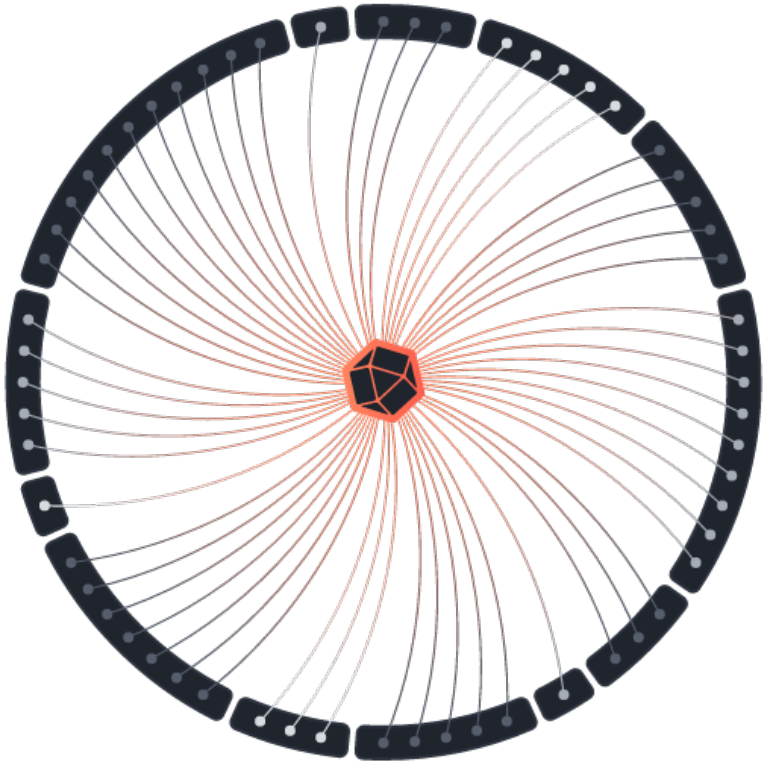
Connecting to a Remote Host

```
influx -database [db] —execute [query] -host 107.23.91.18
```

```
~$ influx -database NOAA_water_database -execute "SELECT * FROM h2o_quality LIMIT 10"
name: h2o_quality
-----------------
time                    index   location        randtag
1439856000000000000     41      coyote_creek    1
1439856000000000000     99      santa_monica    2
1439856360000000000     11      coyote_creek    3
1439856360000000000     56      santa_monica    2
1439856720000000000     38      coyote_creek    1
1439856720000000000     65      santa_monica    3
1439857080000000000     50      coyote_creek    1
1439857080000000000     57      santa_monica    3
1439857440000000000     35      coyote_creek    3
1439857440000000000     8       santa_monica    3
```

influxdata

InfluxQL



- Meta-Queries
- Queries

influxdata

InfluxQL Basics

If you're familiar with basic SQL, you're familiar with InfluxQL

Meta Queries

…tell you about the underlying structure of the data.

- `SHOW DATABASES`
- `SHOW SERIES`
- `SHOW MEASUREMENTS`
- `SHOW TAG KEYS`
- `SHOW FIELD KEYS`

Exercise

Use the meta-queries `SHOW SERIES` and `SHOW FIELD KEYS` to extrapolate out the schema of the dataset we loaded in.

Queries

Basic Select Statement

SELECT <field> FROM <measurement>

- `SELECT * FROM cpu`
- `SELECT free FROM mem`
- `SELECT x + y FROM vars`
- `SELECT x,y FROM nums`

Try Running:

- `SELECT * FROM h2o_quality LIMIT 10`

Select Statement with WHERE Clause

SELECT <field> FROM <measurement> WHERE <conditions>

- `SELECT * FROM cpu WHERE busy > 50`
- `SELECT free FROM mem WHERE host = 'server1'`
- `SELECT x + y FROM vars WHERE some_tag = 'some_key'`
- `SELECT x,y FROM nums WHERE domain =~ /.*/`

Try Running:

- `SELECT * FROM average_temperature WHERE location='santa_monica' LIMIT 10`

Select Statement with Relative Time

SELECT <field> FROM <measurement> WHERE <time condition>

- `SELECT * FROM cpu WHERE time > now() - 1h`
- `SELECT * FROM cpu WHERE time > now() - 10s`
- `SELECT free FROM mem WHERE time > now() - 4d`
- `SELECT x + y FROM vars WHERE time > now() - 10w`
- `SELECT x,y FROM nums WHERE time > now() + 15m`

Using `now()` implies relative time

Try Running:

- `SELECT * FROM h2o_pH WHERE time > now() - 300d LIMIT 10`

influxdata

Select Statement with Absolute Time

SELECT <field> FROM <measurement> WHERE <time condition>

- `SELECT * FROM cpu WHERE time > '2015-08-18 23:00:01.232000000'`
- `SELECT free FROM mem WHERE time < '2015-09-19'`
- `SELECT x + y FROM vars WHERE time > '2015-08-18T23:00:01.232000000Z'`
- `SELECT x,y FROM nums WHERE time > 1388534400s`

Absolute time can be specified in two ways

- Date Time: YYYY-MM-DD HH:MM:SS.nnnnnnnnn (RFC 3339)
- Epoch: number of nanoseconds since January 1, 1970

Try Running:

- `SELECT * FROM h2o_pH WHERE time > '2015-08-18 23:00:01.232000000' LIMIT 10`

influxdata

Select Statement with ORDER BY time

[SELECT STATEMENT] ORDER BY time DESC

- `SELECT * FROM cpu WHERE time > now() - 1h ORDER BY time DESC`
- `SELECT free FROM mem ORDER BY time DESC`

Currently ordering by time is the only supported functionality, but ordering by arbitrary fields is to come eventually. Using `ASC` is redundant.

Try Running:

- `SELECT * FROM h2o_quality ORDER BY time DESC LIMIT 10`

Select Statement with Conjunction

SELECT <field> FROM <measurement> WHERE <conditions> [AND | OR] <conditions>

- `SELECT * FROM cpu WHERE busy > 50 AND location = 'us-west'`
- `SELECT free FROM mem WHERE time > now() - 1h AND host = 'server1'`
- `SELECT x + y FROM vars WHERE time < '2015-09-19' AND time > '2015-08-19'`
- `SELECT x, y FROM nums WHERE x = 10 OR x = 100`

Try Running:

- `SELECT * FROM h2o_feet WHERE location = 'santa_monica' AND water_level > 7`

influxdata

Select Statement with GROUP BY Clause

[SELECT STATEMENT] GROUP BY <tag>

- `SELECT * FROM cpu GROUP BY host`
- `SELECT * FROM cpu GROUP BY *`
- `SELECT free FROM mem WHERE time > now() - 4d GROUP BY location`

Try Running:

- `SELECT * FROM h2o_quality GROUP BY * LIMIT 10`
- `SELECT * FROM h2o_quality GROUP BY location LIMIT 10`

Functions

…in InfluxDB fall into 3 major types…

- Aggregators
- Selectors
- Transformers

Aggregators

- `count()`
- `distinct()`
- `integral()`
- `mean()`
- `median()`
- `spread()`
- `sum()`
- `stddev()`

Using an Aggregator

SELECT <aggregator>(<field>) FROM <measurement> [extra stuff]

- `SELECT count(value) FROM cpu`
- `SELECT mean(free) FROM mem WHERE time > now() - 1h`
- `SELECT sum(x) FROM vars WHERE x > 100`
- `SELECT median(y) FROM nums WHERE domain = 'Z'`

Try Running:

- `SELECT count(index) FROM h2o_quality`
- `SELECT count(index) FROM h2o_quality WHERE location = 'coyote_creek'`

Select Statement with GROUP BY time

[SELECT STATEMENT] WHERE <time condition> GROUP BY time(<period>)

- `SELECT max(busy) FROM cpu WHERE time > now() - 1h GROUP BY time(10m)`
- `SELECT mean(free) FROM free WHERE time > now() - 1d GROUP BY time(1h), host`

Note that the following queries are not valid:

- `SELECT busy FROM cpu WHERE time > now() - 1h GROUP BY time(10m)`
- `SELECT mean(busy) FROM cpu GROUP BY time(10m)`
- `SELECT mean(busy) FROM cpu GROUP BY time(10m) WHERE time > now() - 1h`

Try Running:

- `SELECT mean(degrees) FROM average_temperature WHERE time < '2015-09-19' AND time > '2015-09-18' GROUP BY time(1h)`

- `SELECT mean(degrees) FROM average_temperature WHERE time < '2015-09-19' AND time > '2015-09-18' GROUP BY time(1h), *`

influxdata

Selectors

- `bottom()`
- `first()`
- `last()`
- `max()`
- `min()`
- `percentile()`
- `top()`

Using a Selector

SELECT <selector>(<field>) FROM <measurement> [extra stuff]

- `SELECT percentile(busy,90) FROM cpu WHERE time > now() - 1h`
- `SELECT bottom(water_level,10) FROM factory WHERE location = 'SF'`
- `SELECT max(x) FROM vars`
- `SELECT last(y) FROM nums WHERE domain = 'Z'`

Try Running:

- `SELECT max(degrees) FROM average_temperature GROUP BY *`
- `SELECT top(degrees, 10) FROM average_temperature GROUP BY *`

Transformers

- `derivative()`
- `non_negative_derivative()`
- `difference()`
- `moving_average()`

Using a Transformer

SELECT <transformer>(<field>) FROM <measurement> [extra stuff]

- `SELECT derivative(mean(write_ops)) FROM disk WHERE time > now() - 10m GROUP BY time(10s)`
- `SELECT non_negative_derivative(x) FROM vars`

Select Statement with fill

[SELECT STATEMENT] GROUP BY time(<period>) fill(<value>)

- `SELECT max(busy) FROM cpu WHERE time > now() - 1h GROUP BY time(10m) fill(0)`
- `SELECT mean(free) FROM free WHERE time > now() - 1d GROUP BY time(1h) fill(previous)`
- `SELECT max(busy) FROM cpu WHERE time > now() - 1h GROUP BY time(10m) fill(none)`
- `SELECT mean(free) FROM free WHERE time > now() - 1d GROUP BY time(1h) fill(10)`

Try Running

- `SELECT mean(degrees) FROM average_temperature WHERE time > '2015-09-17' AND time < '2015-09-20' GROUP BY time(1h) fill(previous)`
- `SELECT mean(degrees) FROM average_temperature WHERE time > '2015-09-17' AND time < '2015-09-20' GROUP BY time(1h) fill(0)`

Queries with multiple functions

SELECT <thing>[,<thing>] FROM <field> [other stuff]

- `SELECT max(busy), mean(user) FROM cpu`
- `SELECT mean(free), median(used) FROM mem WHERE time > now() - 1d`
- `SELECT stddev(used), sum(free) FROM disk WHERE host = 'server1'`

Try Running:

- `SELECT max(degrees), min(degrees), mean(degrees) FROM average_temperature`

Select Statement with AS

SELECT <thing> as <other_thing> [,<thing> as <other_thing>] FROM <field> [other stuff]

- `SELECT max(busy) as busy, mean(user) as user FROM cpu`
- `SELECT mean(free) as free, median(used) as median_used FROM mem`
- `SELECT stddev(used) as dev, mean(used) as avg FROM disk WHERE host = 'server1'`

Try Running:

- `SELECT max(degrees) AS max_temp, min(degrees) AS min_temp, mean(degrees) AS avg_temp FROM average_temperature`

Queries with INTO Clause

SELECT <stuff> INTO <measurement> FROM <measurement> [other stuff]

- `SELECT max(busy) as busy, mean(user) as user INTO new_cpu FROM cpu`
- `SELECT mean(free) INTO mean_mem FROM mem`
- `SELECT stddev(used) as dev INTO other_disk FROM disk`

The INTO clause place the results of one query into another measurement in the database.

Try Running:

- `SELECT max(degrees) AS max_temp, min(degrees) AS min_temp, mean(degrees) AS avg_temp INTO downsampled_avg_tmp FROM average_temperature WHERE time > '2015-09-17' AND time < '2015-09-20' GROUP BY time(6h), * fill(0)`
- `SELECT * FROM downsampled_avg_tmp`

SHOW/Kill QUERIES

The `SHOW QUERIES` command shows all active running queries

The `KILL QUERY <qid>` command willl kill an actively running query