Query InfluxDB with Flux

This guide walks through the basics of using Flux to query data from InfluxDB. Every Flux query needs the following:

1. A data source
2. A time range
3. Data filters

1. Define your data source

Flux's `from()` function defines an InfluxDB data source. It requires a `bucket` parameter. The following examples use `example-bucket` as the bucket name.

```
from(bucket:"example-bucket")
```

2. Specify a time range

Flux requires a time range when querying time series data. "Unbounded" queries are very resource-intensive and as a protective measure, Flux will not query the database without a specified range.

Use the pipe-forward operator (`|>`) to pipe data from your data source into `range()`, which specifies a time range for your query. It accepts

two parameters: `start` and `stop`. Start and stop values can be **relative** using negative durations or **absolute** using timestamps.

Example relative time ranges

```
// Relative time range with start only. Stop defaults to
from(bucket:"example-bucket")
    |> range(start: -1h)

// Relative time range with start and stop
from(bucket:"example-bucket")
    |> range(start: -1h, stop: -10m)
```

> Relative ranges are relative to “now.”

Example absolute time range

```
from(bucket:"example-bucket")
    |> range(start: 2021-01-01T00:00:00Z, stop: 2021-01-0
```

Use the following:

For this guide, use the relative time range, `-15m`, to limit query results to data from the last 15 minutes:

```
from(bucket:"example-bucket")
    |> range(start: -15m)
```

3. Filter your data

Pass your ranged data into `filter()` to narrow results based on data attributes or columns. `filter()` has one parameter, `fn`, which expects a predicate function evaluates rows by column values.

`filter()` iterates over every input row and structures row data as a Flux record. The record is passed into the predicate function as `r` where it is evaluated using predicate expressions.

Rows that evaluate to `false` are dropped from the output data. Rows that evaluate to `true` persist in the output data.

```
// Pattern
(r) => (r.recordProperty comparisonOperator comparisonExp

// Example with single filter
(r) => (r._measurement == "cpu")

// Example with multiple filters
(r) => r._measurement == "cpu" and r._field != "usage_sys
```

Use the following:

For this example, filter by the `cpu` measurement, `usage_system` field, and `cpu-total` tag value:

```
from(bucket: "example-bucket")
    |> range(start: -15m)
    |> filter(fn: (r) => r._measurement == "cpu" and r._f
```

4. Yield your queried data

`yield()` outputs the result of the query.

```
from(bucket: "example-bucket")
    |> range(start: -15m)
    |> filter(fn: (r) => r._measurement == "cpu" and r._f
    |> yield()
```

Flux automatically assumes a `yield()` function at the end of each script to output and visualize the data. Explicitly calling `yield()` is only necessary when including multiple queries in the same Flux query. Each set of returned data needs to be named using the `yield()` function.

Congratulations!

You have now queried data from InfluxDB using Flux.

The query shown here is a basic example. Flux queries can be extended in many ways to form powerful scripts.

← Get started with Flux and InfluxDB

Transform data with Flux →

Related

Flux query basics
Query data with Flux
from() function
range() function
filter() function

query *flux*

Was this page helpful? Yes No

Support and feedback

Thank you for being part of our community! We welcome and encourage your feedback and bug reports for InfluxDB and this documentation. To find support, use the following resources:

InfluxData Community InfluxDB Community Slack

InfluxDB Cloud and InfluxDB Enterprise customers can contact InfluxData Support.

Edit this page

Submit docs issue

Submit InfluxDB issue