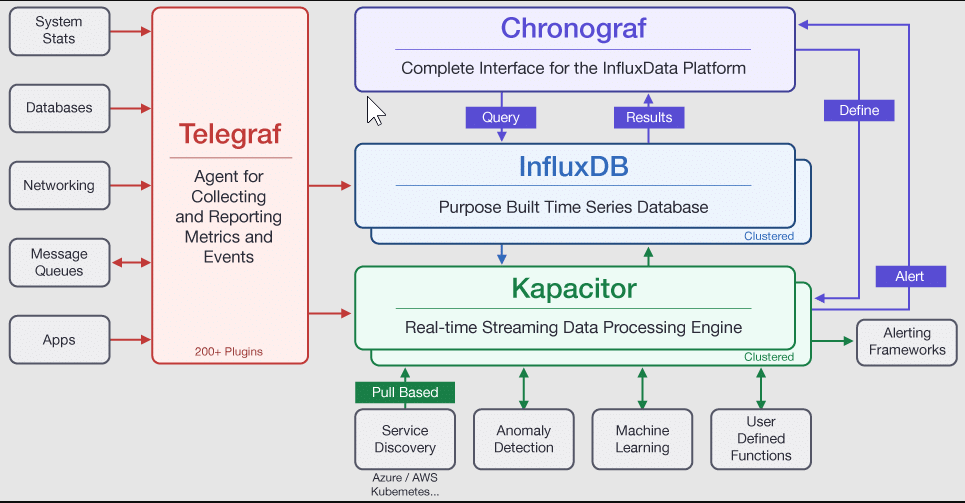
**InfluxDB**



InfluxDB 2.7 is the platform purpose-built to collect, store, process and visualize time series data. **Time series data** is a sequence of data points indexed in time order. Data points typically consist of successive measurements made from the same source and are used to track changes over time. Examples of time series data include:

* Industrial sensor data
* Server performance metrics
* Heartbeats per minute
* Electrical activity in the brain
* Rainfall measurements
* Stock prices

This multi-part tutorial walks you through writing time series data to InfluxDB 2.7, querying that data, processing and alerting on the data, and then visualizing the data.

### Data organization

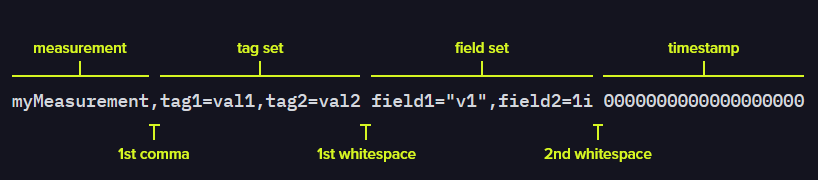
The InfluxDB data model organizes time series data into buckets and measurements. A bucket can contain multiple measurements. Measurements contain multiple tags and fields.

* **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–for example, something to identify the source of the data like host, location, station, etc.
    - **Fields**: Key-value pairs with values that change over time–for example: temperature, pressure, stock price, etc. Fields are required in InfluxDB data and are not indexed. Queries that filter field values must scan all field values to match query conditions. As a result, queries on tags are more performant than queries on fields.
    - **Timestamp**: Timestamp associated with the data. When stored on disk and queried, all data is ordered by time.

# Writing data

## 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.



You can write the data with line protocol using influxdb UI or cli or api. But remember the data should be in image above format. Also you can use either telegraf or client libraries without worrying about the format.

## Telegraf

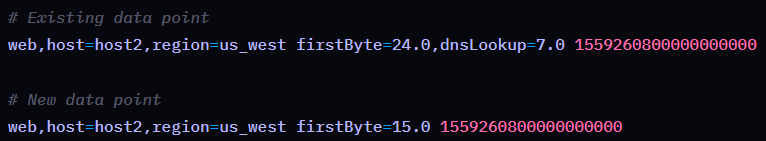
Telegraf is InfluxData’s data collection agent for collecting and reporting metrics. Its vast library of input plugins and “plug-and-play” architecture lets you quickly and easily collect metrics from many different sources.

## Optimize writes to InfluxDB

* Batch writes
* Sort tags by key
* Use the coarsest time precision possible
* Use gzip compression
* Synchronize hosts with NTP

## Duplicate data points

For points that have the same measurement name, tag set, and timestamp, InfluxDB creates a union of the old and new field sets. For any matching field keys, InfluxDB uses the field value of the new point.



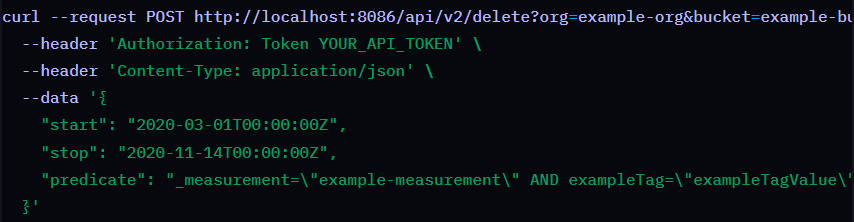


To preserve both old and new field values in duplicate points, use one of the following strategies:

* Add an arbitrary tag
* Increment the timestamp

# Delete Records

You can delete records using cli or api like below:



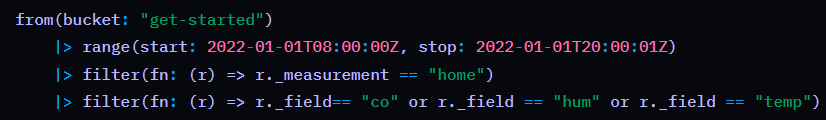
**Querying data**

InfluxDB supports many different tools for querying data, including:

* InfluxDB user interface (UI)
* [InfluxDB HTTP API](https://docs.influxdata.com/influxdb/v2/reference/api/)
* [influx CLI](https://docs.influxdata.com/influxdb/v2/tools/influx-cli/)
* [Chronograf](https://docs.influxdata.com/chronograf/v1/)
* [Grafana](https://docs.influxdata.com/influxdb/v2/tools/grafana/)
* [InfluxDB client libraries](https://docs.influxdata.com/influxdb/v2/api-guide/client-libraries/)

You can query data with two languages:

* **Flux**: A functional scripting language designed to query and process data from InfluxDB and other data sources.
* **InfluxQL**: A SQL-like query language designed to query time series data from InfluxDB.

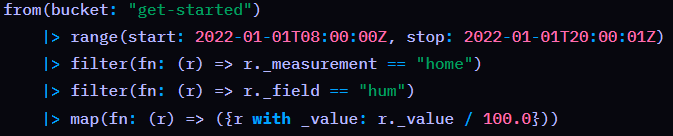


# Processing data

“Processing” data could mean transforming, aggregating, downsampling, or alerting on data.

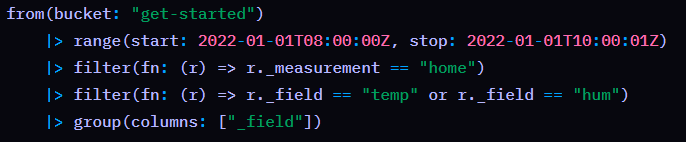
## Remap or assign values in your data

Use the [map() function](https://docs.influxdata.com/flux/v0/stdlib/universe/map/) to iterate over each row in your data and update the values in that row. map() is one of the most useful functions in Flux and will help you accomplish many of they data processing operations you need to perform.



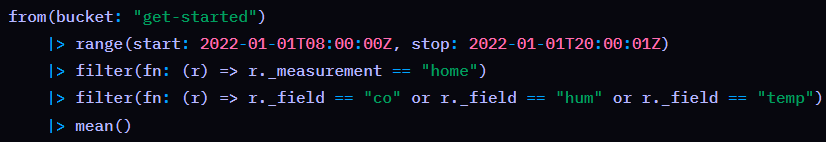
## Group data

Use the [group() function](https://docs.influxdata.com/flux/v0/stdlib/universe/group/) to regroup your data by specific column values in preparation for further processing.



## Aggregate or select specific data

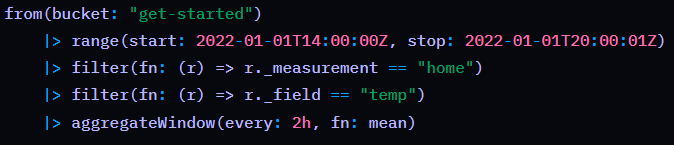
Use Flux [aggregate](https://docs.influxdata.com/flux/v0/function-types/#aggregates) or [selector](https://docs.influxdata.com/flux/v0/function-types/#selectors) functions to return aggregate or selected values from **each** input table.

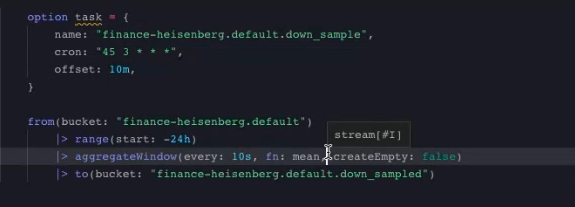


## Downsample data

Downsampling data is a strategy that improve performance at query time and also optimizes long-term data storage. Simply put, downsampling reduces the number of points returned by a query without losing the general trends in the data.

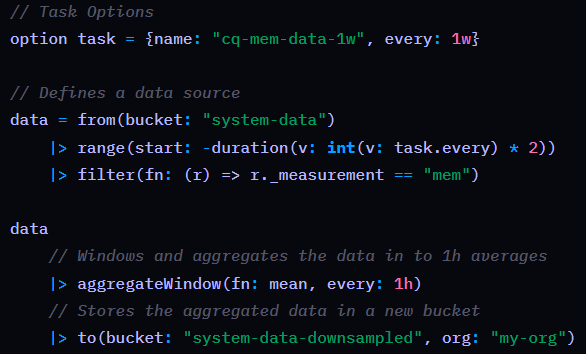
The most common way to downsample data is by time intervals or “windows.” For example, you may want to query the last hour of data and return the average value for every five minute window.





## Automate processing with InfluxDB tasks

InfluxDB tasks are scheduled queries that can perform any of the data processing operations described above. Generally tasks then use the to() function to write the processed result back to InfluxDB.



# Alerting

Monitor your time series data and send alerts by creating checks, notification rules, and notification endpoints. Steps are like below:

1. Create checks to monitor data and assign a status.
2. Add notification endpoints to send notifications to third parties.
3. Create notification rules to check statuses and send notifications to your notifications endpoints.

## checks

Checks in InfluxDB query data and apply a status or level to each data point based on specified conditions.

### [Check types](https://docs.influxdata.com/influxdb/v2/monitor-alert/checks/create/#check-types)

* [Threshold check](https://docs.influxdata.com/influxdb/v2/monitor-alert/checks/create/#threshold-check): A threshold check assigns a status based on a value being above, below, inside, or outside of defined thresholds.
* Deadman check: A deadman check assigns a status to data when a series or group doesn’t report in a specified amount of time.

## notification endpoints

Notification endpoints store information to connect to a third-party service. Create a connection to a HTTP, Slack, or PagerDuty endpoint.

## notification rules

They allow users to define conditions under which notifications should be sent to specified endpoints.

# Kapacitor

Kapacitor is a data processing framework that makes it easy to create alerts, run ETL jobs and detect anomalies.

# Chronograf

Chronograf is a data visualization and dashboarding tool designed to visualize data in InfluxDB 1.x. It is part of the TICKstack that provides an InfluxQL data explorer, Kapacitor integrations, and more.

# InfluxDB storage engine

When the storage engine receives a write request, the following steps occur:

1. The write request is appended to the end of the WAL file.
2. Data is written to disk using fsync().
3. The in-memory cache is updated.
4. When data is successfully written to disk, a response confirms the write request was successful.

When the storage engine restarts, the WAL file is read back into the in-memory database. InfluxDB then answers requests to the /read endpoint.

## Write Ahead Log (WAL)

The **Write Ahead Log** (WAL) retains InfluxDB data when the storage engine restarts. The WAL ensures data is durable in case of an unexpected failure.

## Cache

The **cache** is an in-memory copy of data points currently stored in the WAL. The WAL and cache are separate entities and do not interact with each other. The storage engine coordinates writes to both.

The cache:

* Organizes points by key (measurement, tag set, and unique field). Each field is stored in its own time-ordered range.
* Stores uncompressed data.
* Gets updates from the WAL each time the storage engine restarts. The cache is queried at runtime and merged with the data stored in TSM files.
* Uses a maximum maxSize bytes of memory.

## Time-Structured Merge Tree (TSM)

The storage engine uses a **Time-Structured Merge Tree** (TSM) data format. TSM files store compressed series data in a columnar format. To improve efficiency, the storage engine only stores differences (or deltas) between values in a series. Column-oriented storage lets the engine read by series key and omit extraneous data.

After fields are stored safely in TSM files, the WAL is truncated and the cache is cleared. The **compaction** process creates read-optimized TSM files. The TSM compaction code is quite complex. However, the high-level goal is quite simple: organize values for a series together into long runs to best optimize compression and scanning queries.

## Time Series Index (TSI)

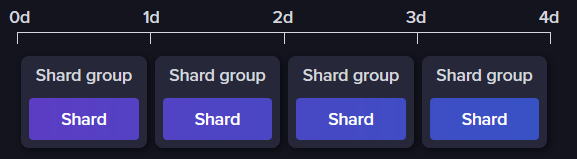
As data cardinality (the number of series) grows, queries read more series keys and become slower. The **Time Series Index** ensures queries remain fast as data cardinality grows. The TSI stores series keys grouped by measurement, tag, and field.

# InfluxDB shards and shard groups

A shard contains encoded and compressed time series data for a given time range defined by the shard group duration. All points in a series within the specified shard group duration are stored in the same shard. A single shard contains multiple series, one or more TSM files on disk, and belongs to a shard group.

A shard group belongs to an InfluxDB bucket and contains time series data for a specific time range defined by the shard group duration.

The **shard group duration** specifies the time range for each shard group and determines how often to create a new shard group. By default, InfluxDB sets the shard group duration according to the retention period of the bucket:



### Shard compaction

InfluxDB compacts shards at regular intervals to compress time series data and optimize disk usage. When compactions are enabled, InfluxDB checks to see whether shard compactions are needed every second. If there haven’t been writes during the compact-full-write-cold-duration period (by default, 4h), InfluxDB compacts all TSM files. Otherwise, InfluxDB groups TSM files into compaction levels (determined by the number of times the file have been compacted), and attempts to combine files and compress them more efficiently.

# Data retention

The **InfluxDB retention enforcement service** checks for and removes data with timestamps beyond the defined retention period of the bucket the data is stored in. This service is designed to automatically delete “expired” data and optimize disk usage without any user intervention.

By default, the retention enforcement service runs every 30 minutes. You can configure this interval with the storage-retention-check-interval configuration option.

The InfluxDB retention enforcement service runs at regular intervals and deletes shard groups, not individual points. The service will only delete a shard group when the entire time range covered by the shard group is beyond the bucket retention period.

# Replication

Use InfluxDB replication streams (InfluxDB Edge Data Replication) to replicate the incoming data of select buckets to one or more buckets on a remote InfluxDB OSS, InfluxDB Cloud, or InfluxDB Enterprise instance.

Once a replication stream is created, InfluxDB OSS will replicate all writes to the specified bucket to the remote InfluxDB bucket.

Only write operations are replicated. Other data operations (like deletes or restores) are not replicated.

In InfluxDB OSS, large write request bodies are written entirely. When replicated, write requests are sent to the remote bucket in batches. The maximum batch size is 500 kB (typically between 250 to 500 lines of line protocol). This may result in scenarios where some batches succeed and others fail.

