**Time Series Database**

**What?**

A time series database is a database designed to store and retrieve time-stamped or time series data. The time series is a series of numeric data points of some particular metric that are tracked, monitored, down sampled, and aggregated over time. The event could be server metrics, application performance monitoring, network data, sensor data, clicks, trades in a market, and many other types of analytics data.

**Why?**

Using Time Series Database is because every event (server metrics, application performance monitoring, network data, sensor data) will emit a huge amount of metrics or time series data, to support this huge workload of large data points, data sources, monitoring and controls there is need for a highly scalable database like Time Series Database.

Time series databases have key architectural design properties that make them very different from other databases. These include time-stamp data storage and compression, data lifecycle management, data summarization, ability to handle large time series dependent scans of many records, and time series queries.

The reason for using Time Series Database than other databases,

1) **Scale**: Time-series data accumulates very quickly. (For example, a single connected car will collect 4,000 GB of data per day.) And normal databases are not designed to handle that scale. Relational databases fare poorly with very large datasets; NoSQL databases fare better at scale but can still be outperformed by a database fine-tuned for time-series data. In contrast, time-series databases (which can be based on relational or NoSQL databases) handle scale by introducing efficiencies that is possible when the time is considered as a primary field. These efficiencies result in performance improvements, including higher ingest rates, faster queries at scale, and better data compression.

2) **Usability:** TSDBs also typically include functions and operations common to time-series data analysis such as data retention policies, continuous queries, flexible time aggregations, etc.

**When?**

A time-series database is used when there is a use case to store large volumes of timestamped data in a format that allows fast insertion and fast retrieval to support complex analysis on that data. Timeseries Databases can be used in the below mentioned areas,

* Monitoring software systems: Virtual machines, containers, services, applications
* Monitoring physical systems: Equipment, machinery, connected devices, the environment, our homes, our bodies
* Asset tracking applications: Vehicles, trucks, physical containers, pallets
* Financial trading systems: Classic securities, newer cryptocurrencies
* Event applications: Tracking user/customer interaction data
* Business intelligence tools: Tracking key metrics and the overall health of the business

**TICK Stack:**

**TICK Stack** is a collection of open source components that combine to deliver a platform for easily **storing, visualizing and monitoring time series data** such as metrics and events. The components are:

* + [**Telegraf**](https://www.influxdata.com/time-series-platform/telegraf/), a server agent for collecting and reporting metrics
  + [**InfluxDB**](https://www.influxdata.com/time-series-platform/influxdb/), a high-performance time series database
  + [**Chronograf**](https://www.influxdata.com/time-series-platform/chronograf/), a user interface for the platform
  + [**Kapacitor**](https://www.influxdata.com/time-series-platform/kapacitor/), a data-processing engine that can process, stream and batch data from InfluxDB.

*Refer*: <https://portal.influxdata.com/downloads/> to download the TICK components

**Telegraf:**

Telegraf is a plugin-driven agent that collects, processes, aggregates, and writes metrics from databases, systems, and sensors. It supports four categories of plugins including input,output, aggregator, and processor.

**Input Plugin:**

Telegraf input plugins are used with the InfluxData time series platform to collect metrics from the system, services, or third-party APIs. Telegraf has 166 input plugins

**Output Plugin:**

Telegraf output plugins write metrics to various destinations. Telegraf has 33 output plugins.

**Aggregator Plugin:**

Telegraf aggregator plugins create aggregate metrics for example, mean, min, max, quantiles, etc. and it has 5 aggregator plugins.

**Processor Plugin:**

Telegraf processor plugins transform, decorate, and filter metrics. Telegraf has 13 processor plugins.

**InfluxDB**

InfluxDB is a [time series database](https://www.influxdata.com/time-series-database/) designed to handle high write and query loads. It is an integral component of the [TICK stack](https://influxdata.com/time-series-platform/). InfluxDB is meant to be used as a backing store for any use case involving large amounts of timestamped data, including DevOps monitoring, application metrics, IoT sensor data, and real-time analytics.

**Chronograf**

Chronograf is the administrative user interface and visualization engine of the stack. It makes it easy to setup and maintain the monitoring and alerting for your infrastructure. It’s simple to use and includes templates and libraries that allow you to rapidly build dashboards with real-time visualizations of your data and to easily create alerting and automation rules.

**Key Features of Chronograf**

* Data Visualization

*Refer*: <https://docs.influxdata.com/chronograf/v1.7/guides/querying-data/> for quering and visualization of data through UI

* Alerting

*Refer:* <https://docs.influxdata.com/chronograf/v1.7/guides/create-alert-rules/> rules to create alert.

* Infrastructure Monitoring

To monitor the infrastructure for example

*Refer*: <https://docs.influxdata.com/chronograf/v1.7/guides/monitoring-influxenterprise-clusters/#step-4-explore-the-monitoring-data-in-chronograf>

**Kapacitor**

* Kapacitor is a Data Processing Engine
* Kapacitor task defines work to do on a set of data. There are two types of tasks: Stream and Batch
* A Stream task mirrors all data written from InfluxDB to Kapacitor. This offloads the query overhead from InfluxDB to Kapacitor, but requires Kapacitor to store the data on disk
* A Bach task queries data from InfluxDB at a set interval and processes the data as it’s queried
* It can process both stream and batch data from InfluxDB
* Kapacitor lets user to plug their own custom logic or user-defined functions to process alerts with dynamic thresholds, match metrics for patterns, compute statistical anomalies, and perform specific actions based on these alerts, like dynamic load rebalancing
* [Kapacitor integrates](https://www.influxdata.com/products/integrations/) with HipChat, OpsGenie, Alerta, Sensu, PagerDuty, Slack and more

**Timestamps**

If a data point does not include a timestamp when it is received by the database, InfluxDB uses the current system time (UTC) of its host machine.

**Writing data into InfluxDB**

We can write data on Influx by using

* Line protocol
* Telegraf and
* Data scrapers

**Line protocol**

InfluxDB line protocol is a text-based format for writing points to InfluxDB. Field values in can be boolean, int64, float64, or string

**Basic Line protocol data format**

mem,host=host1 used\_percent=23.43234543 1556892576842902000

cpu,host=host1 usage\_user=3.8234,usage\_system=4.23874 1556892726597397000

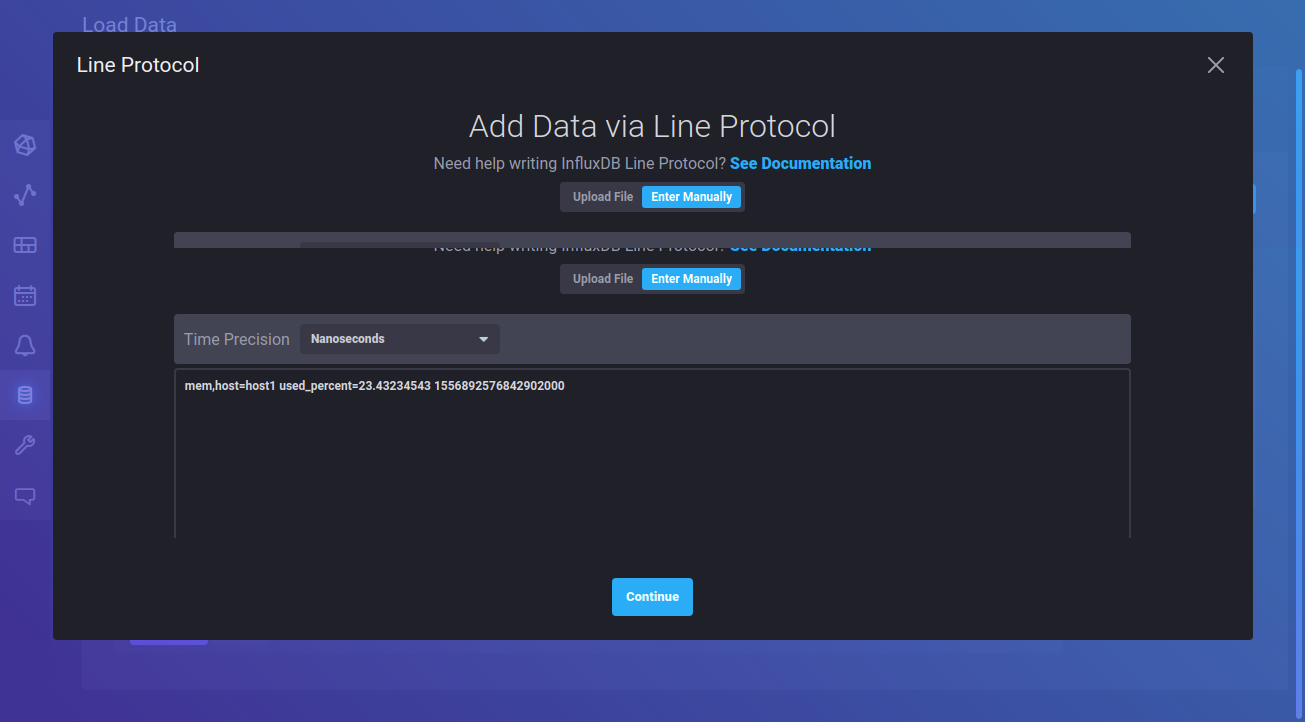
mem,host=host1 used\_percent=21.83599203 1556892777007291000

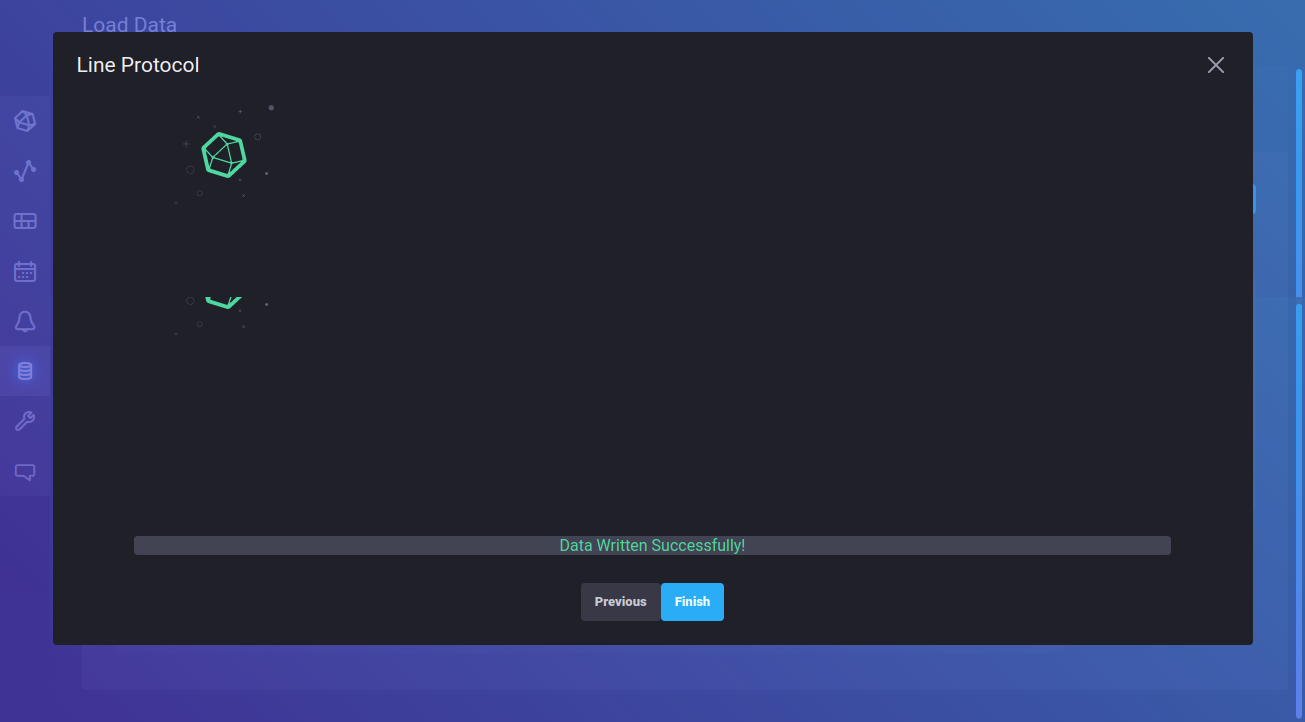
*Refer*: [https://docs.influxdata.com/influxdb/v1.7/write\_protocols/line\_protocol\_reference/#:~:targetText=\*%20InfluxDB%20line%20protocol%20allows%20users,(see%20the%20example%20below](https://docs.influxdata.com/influxdb/v1.7/write_protocols/line_protocol_reference/#:~:targetText=*%20InfluxDB%20line%20protocol%20allows%20users,(see%20the%20example%20below)

**Inserting the data by using Line protocol**

* Go to Load data.
* Inside that select the specific bucket (here sample-line)
* In that in select add data, Line protocol and select enter manually.
* In line protocol format provide the measurements and fields and click finish.
* Data will be inserted to our bucket.
* View it in data explorer.

**Screenshots**





**Inserting the data by using Telegraf**

* Install Telegraf
* Go to Load Data. Select add data by Telegraf agent.
* Go to telegraf create configuration.
* Default configuration plugins (system, redis, docker, kubernetes, nginx).
* Add any plugins and click finish. It automaticallly generates a config file and token.
* We need to set the export token in terminal and the key to start the telegraf

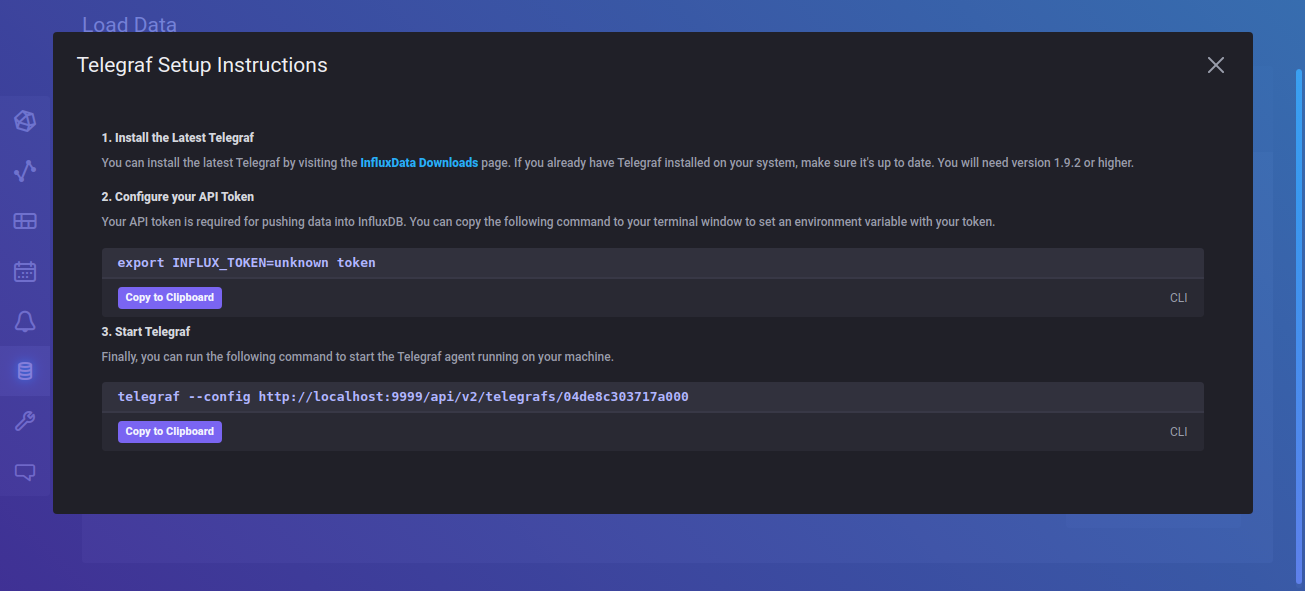
**For Custom configuration**

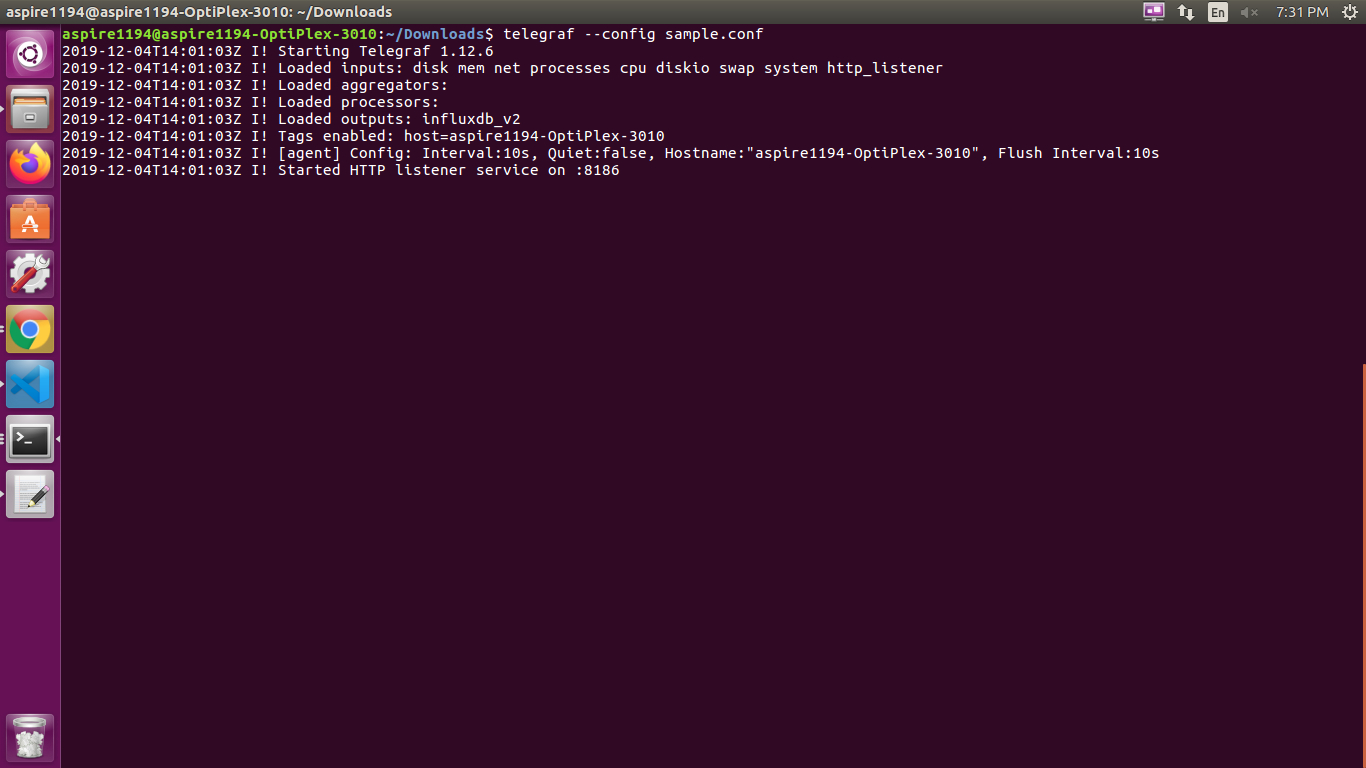
* Create a .conf file
* For a telegraf to run we must need the following plugins (ie) input, outputs, token, bucket and organisation
* Provide the following and add the plugins
* Start telegraf by using telegraf --config filename.conf

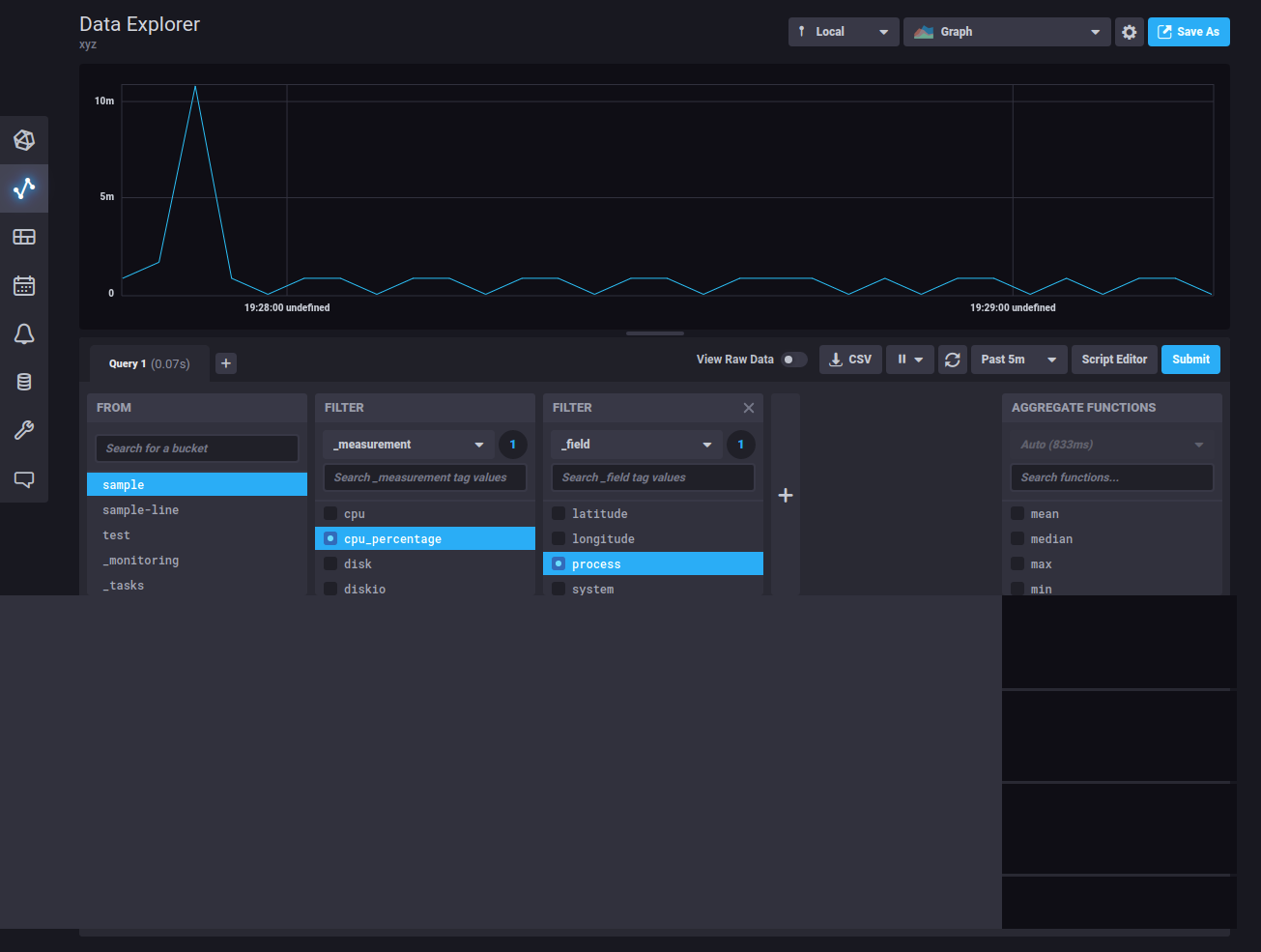
Config file refer: <https://aspiresysinc-my.sharepoint.com/:w:/g/personal/ranjit_gaudaman_aspiresys_com/EUYldihtuIRLsX5MXk0tIS0BsZDy0bY_kHan4yl_a1HMww?e=ehsCsL>

**Screenshots**





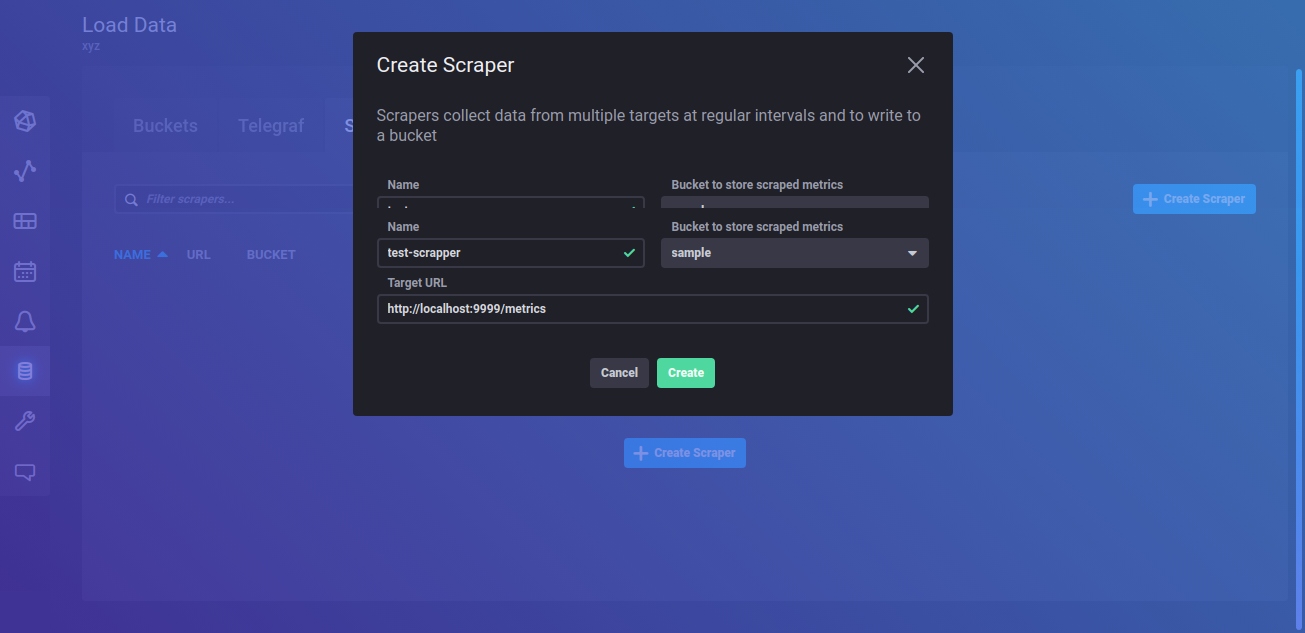


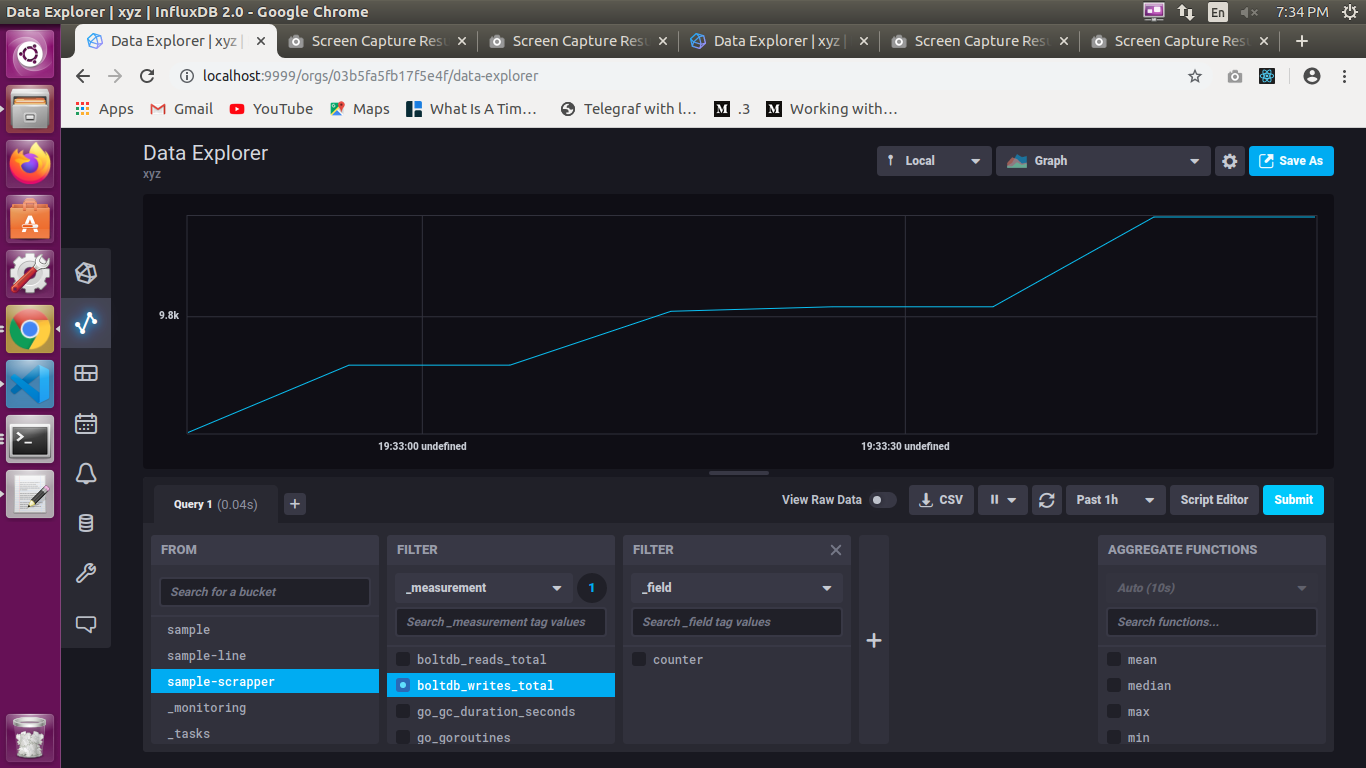


**Inserting the data by using Data scrappers**

* Similarly select add data by data scrappers
* Create scrapper (this measures many parameters at a time) and choose bucket
* Scrapper sends data to influxdb every 10seconds

**Screenshot**





**Query the results in influx UI by using FLUX**

* Flux is InfluxData’s functional data scripting language designed for querying, analyzing, and acting on data
* In flux we use pipe forward operator to connect two or more actions together (chain operations together) (|>)
* Flux returns tables where pipe operator pipes it to the next function

**Functions in flux**:

**range()** -- time range for our query

Eg: |>range(start:,stop:)

**filter()** -- filter the input according to the params

eg:|> filter(fn:(r)=>

r.\_measurement=="cpu"

We can also used and and or operators in it

(r) => (r.\_measurement == "cpu") and (r.\_field != "usage\_system" )

**yield()**--to yield to result of the following functions

flux assumes yield at end of every operation

**Mean()**

**Aggregatewindow()**

**Window()**

Whereas mean(), aggregatewindow() and window() are used to tranform the obtained ouput.

As data is gathered into windows of time, each window is output as its own table. When visualized, each table is assigned a unique color.

**Ways to execute queries with InfluxDB**

**Data Explorer**

In the UI they have provided options in data explored column with script editor (in which we use flux) and query builder.

**Influx REPL**

The influx repl command starts an interactive read-eval-print-loop (REPL) where you can write and execute Flux queries.

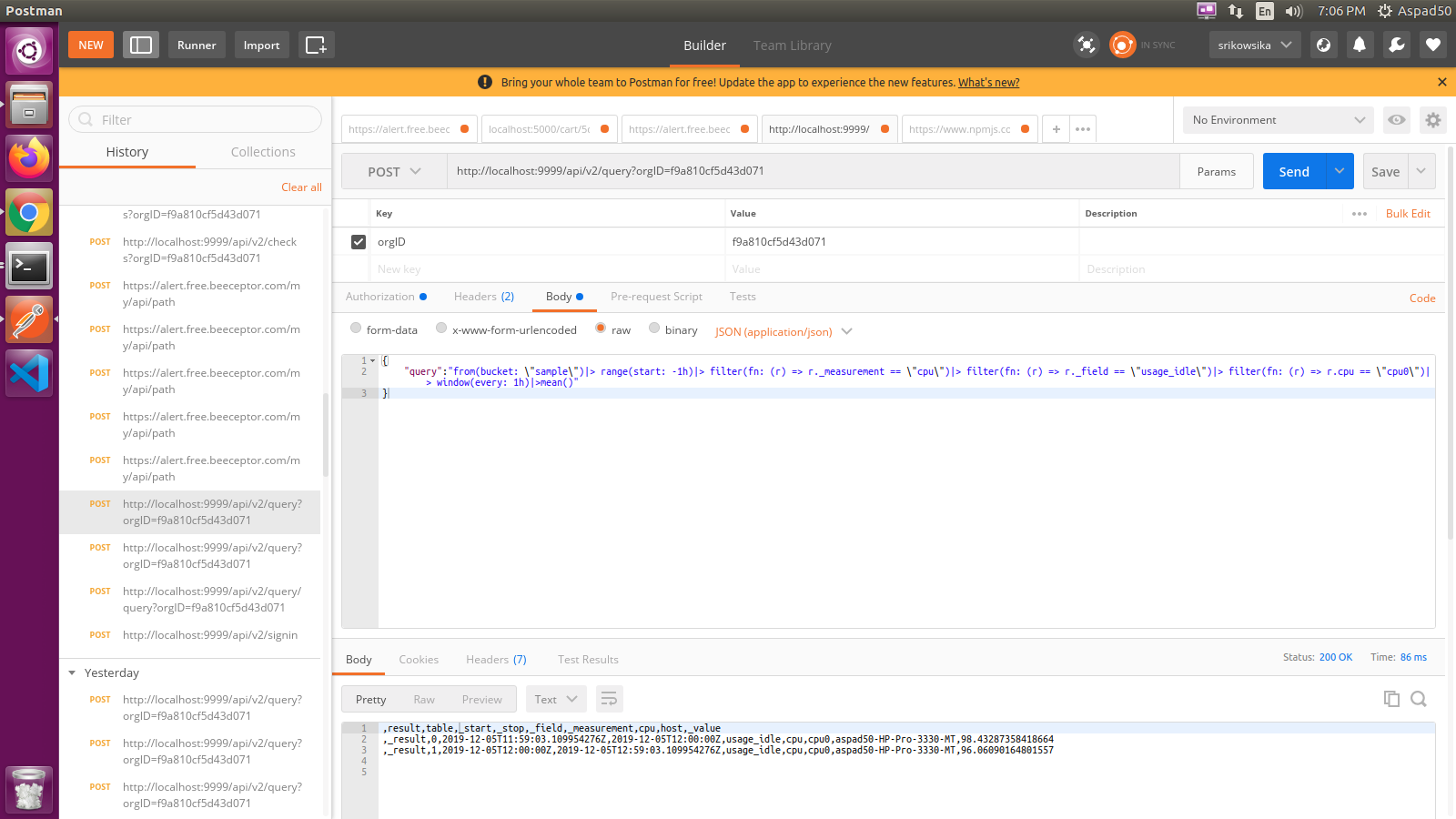
**Influx query command**

Run a query from a file

**InfluxDB API**

We can perform all the UI activitives through API by quering the InfluxDB running in <http://localhost:9999>. Query InfluxDB through the **/api/v2/**query endpoint, query result will be a CSV. The query should be of Post method with the **Basic Authorization, Username and password**, **Organization ID or Organization name** as a query string, content type - application/json

<http://localhost:9999> is queried posted via Postman to get the data from the InfluxDB, and the results are obtained in CSV. The query should be in Flux language

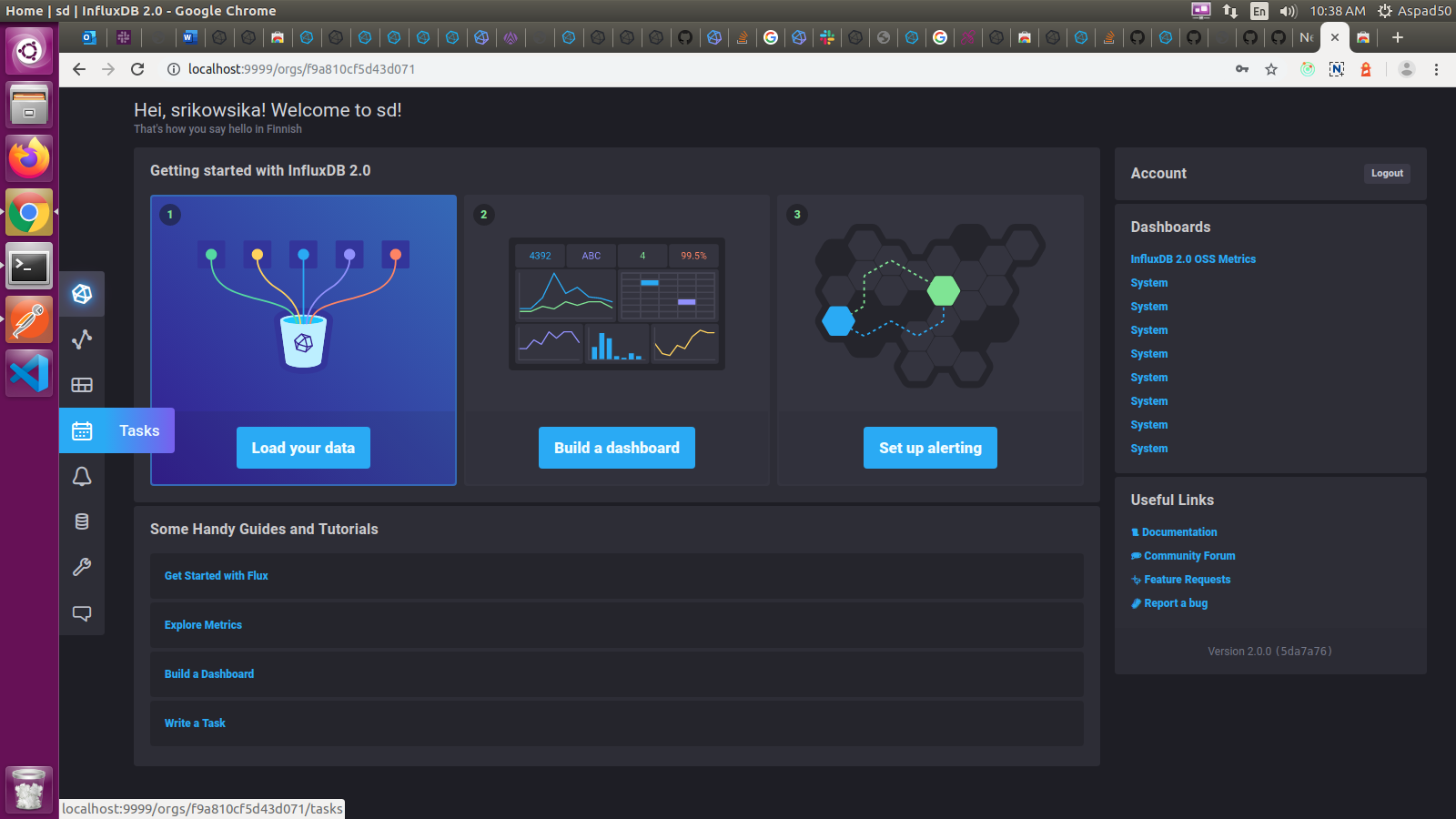


*Refer*: <https://v2.docs.influxdata.com/v2.0/api/> for Influx API Service

**Process the data**

**InfluxDb Task**

An InfluxDB task is a scheduled Flux script that takes a stream of input data, modifies or analyzes it in some way, then stores the modified data in a new bucket or performs other actions.



**Components of task**

These 4 parameters are needed for configuration of task to run in an orderly manner.

* **Task option** - must contain the name of the task, retry time, and repetition of every hours and offset value.
* **Data source** - must contain the data source from the bucket and we can also select the range of data from the bucket by applying the filter and other functions
* **Data processing** - by applying the aggregation functions we can downsample or apply some functions to it and store it to other bucket
* **Destination** – the source of other destination bucket

Creation of task and pushing the result to another buket is explained in the following URL <https://v2.docs.influxdata.com/v2.0/process-data/get-started/>

**Examples of tasks:**

* Data downsampling
* Anomaly detection
* Alerting and other common tasks

**Downsampling of data**

The data is downsampled by using built in aggregate functions. There are multiple aggregate functions provided.

*Refer:*<https://v2.docs.influxdata.com/v2.0/reference/flux/stdlib/built-in/transformations/aggregates/> , the aggregate functions are explained.

**Monitoring and Alerting**

Monitor your time series data and send alerts by creating

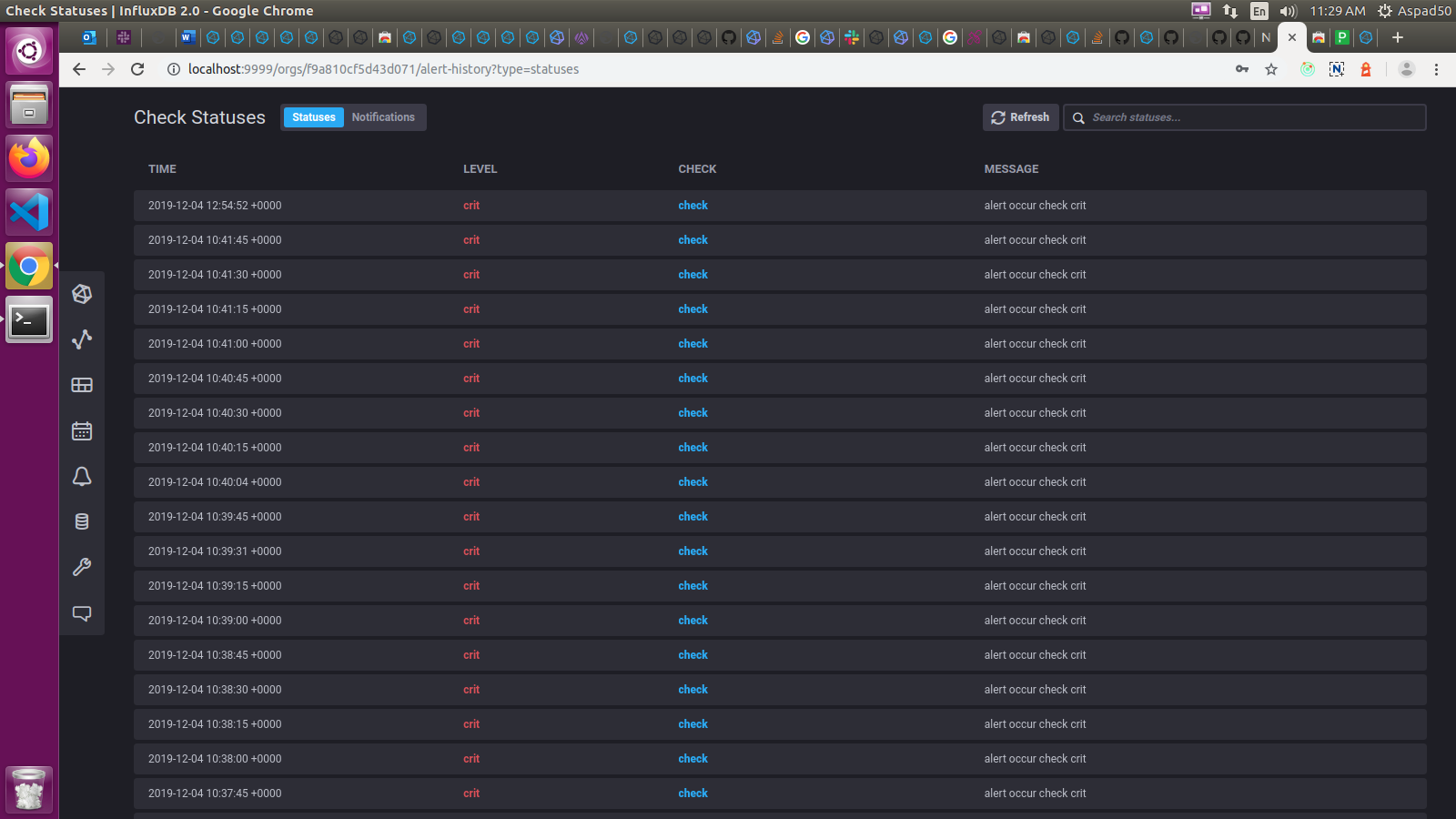
* + Checks
  + Notification Rules
  + Notification Endpoints

**Checks**

* Checks are part of queries used in monitoring to read input data and assign a [status](https://v2.docs.influxdata.com/v2.0/reference/glossary/#check-status) (\_level) based on specified conditions
* A [check](https://v2.docs.influxdata.com/v2.0/reference/glossary/#check) gets one of the following statuses (\_level): **crit, info, warn, or ok**
* Check statuses are written to a status measurement in the \_monitoring bucket

*Refer*: <https://v2.docs.influxdata.com/v2.0/reference/glossary/#check> to create a Check in InfluxData UI

The statuses of the check can be viewed by clicking **View History** option of check



**Notification Endpoints**

The notification endpoint specifies the **Slack, HTTP or PagerDuty** endpoint to send a notification and contains configuration details for connecting to the endpoint

*Refer*: <https://v2.docs.influxdata.com/v2.0/monitor-alert/notification-endpoints/create/> to create Notification end points

**Notification Rules**

A notification rule specifies a status level to alert on, the notification message to send for the specified status level, and the interval or schedule you want to check the status level. If conditions are met, the notification rule sends a message to the[**notification endpoint**](https://v2.docs.influxdata.com/v2.0/reference/glossary/#notification-endpoint) and stores a receipt in a notification measurement in the **\_monitoring bucket**

*Refer:*<https://v2.docs.influxdata.com/v2.0/monitor-alert/notification-rules/create/> to create Notification Rules