# CS 387 - Project Report

Github Link:
https://github.com/sohammistri/DB-Project

Atin Bainada — 190050024
Likhith — 190050043
Shabnam Sahay — 190050111
Soham Mistri — 190050116

## Contents

# 1 Requirements

## 1.1 Problem Description

The aim of the project was to build a web application to track the running applications on one's system and display the appropriate time-series metrics associated with them. This application allows the user to track the resources usage of a specified application or device. The user can choose to see interactive, live graphs to better visualise the usage of resources. In case of certain resource usage values crossing defined threshold, the application issues a warning notification to the user.

## 1.2 Database Utilised

InfluxDB, a time-series database, was a good choice for working with such data for the following reasons: i) it is well-suited to store dynamic time-series data, ii) it allows fast and convenient querying using the declarative Flux language, and iii) it has inbuilt methods for maintaining data consistency.

The domain of resource-usage data that we modelled includes multiple systems, where multiple processes may be running on each system, and each process is associated with a user. For a single process, we tracked quantities such as percent of CPU usage, memory consumption, and so on.

We have used the ProcStat plugin (https://www.influxdata.com/integration/procstat/) to target processes of our choice and extract relevant data from them.

## 1.3 Web Tools Utilised

`express-js`
: We used `express-js` with `ejs`(embedded JavaScript) as template engine for creating a web server, handling the http requests, routing different urls and displaying the web pages.

`@influxdata/influxdb-client`
: For connecting to Influx DB and querying data from Influx DB we used their official library `@influxdata/influxdb-client`.

`@influxdata/giraffe`
: We used `@influxdata/giraffe` for plotting the data fetched from Influx DB using the flux queries.

`dotenv`
: We used `dotenv` for loading environment variables(eg influx token, bucket etc) from `.env` file.

# 2 Design

## 2.1 User Classes and Use Cases

We have a single user class, who can access information about multiple processes across multiple systems.

The use case implemented in our is application performance management, where the user specifies a process/set of processes on a single or multiple systems, and obtains analytics and metrics for CPU usage, network traffic, memory usage.

## 2.2 User Interface

The user interface of this application is as follows. A list of possible metrics to monitor is displayed on the landing page of the application. The user can choose which metric they want to monitor, and click on the corresponding link. On the respective monitoring page that opens, the user can specify the starting time from which they want to monitor the metric, the host name of the system they wish to monitor, along with additional parameters where required such as process name, disk name, and so on. The user is then shown real-time information through a time-series graph about the usage level of the metric they desired, constantly updated to reflect the present state.

## 2.3 Unrequired Parts

InfluxDB handles the definition and updation of relations containing the application-related data, hence there is no need for separately specifying an ER diagram and schema. Similarly, InfluxDB handles the definition and checking of integrity constraints as well.

No materialised views are required for our application. Since the data is time-series data, creating materialised views is pointless as the view will have to anyway be updated at each passing time interval, in order to correctly display live statistics to the user.

There is no requirement for defining indexes in this case, since InfluxDB automatically takes care of this optimization while storing such time-series data.

## 2.4 Transaction description for use cases

**Memory monitoring**
: Select the `memory_available_percent` and `memory_used_percent` fields for a specified hostnamem.

**Disk capacity monitoring**
: Select the `disk_used_percent` field for a specified hostname.

**Disk IO monitoring**
: Select the `diskio_iops_in_progress`, `diskio_read_time`, and `diskio_write_time` fields for a specified hostname.

**CPU usage monitoring**
: Select the `cpu_usage_user`, `cpu_usage_system`, and `cpu_usage_idle` fields for a specified hostname.

**Network traffic monitoring**
: Select the `packets_sent` and `packets_recv` fields for a specified hostname.

**System-level process monitoring**
Select the `processes_running`, `processes_blocked`, `processes_sleeping` and `processes_total` fields for a specified hostname.

**Individual process monitoring**
Select the `cpu_usage`, `memory_data`, `memory_usage`, and `num_threads` fields for a specified process name and hostname.

## 2.5 Telegraf configuration

```
[agent]
  interval = "10s"
  round_interval = true
  metric_batch_size = 1000
  metric_buffer_limit = 10000
  collection_jitter = "0s"
  flush_interval = "10s"
  flush_jitter = "0s"
  precision = ""
  hostname = ""
  omit_hostname = false

[[outputs.influxdb_v2]]
  urls = ["http://localhost:8086"]
  token = "$INFLUX_TOKEN"
  organization = "$INFLUX_ORG"
  bucket = "$INFLUX_BUCKET"

[[inputs.cpu]]
  percpu = true
  totalcpu = true
  collect_cpu_time = false
  report_active = false

[[inputs.disk]]
  ignore_fs = ["tmpfs", "devtmpfs", "devfs", "iso9660",
  "overlay", "aufs", "squashfs"]

[[inputs.diskio]]
[[inputs.kernel]]
[[inputs.mem]]
[[inputs.processes]]
[[inputs.swap]]
[[inputs.system]]
[[inputs.net]]

[[inputs.procstat]]
```

```
    user = "root"

  [[inputs.procstat]]
    pattern = "firefox"

  [[inputs.procstat]]
    pattern = "chrome"

  [[inputs.procstat]]
    pattern = "docker"

  [[inputs.procstat]]
    pid_file = "/var/run/docker.pid"
```

## 2.6 Flux Queries

The Flux queries used to fetch data from Influxdb for the use cases defined in previously.

Memory monitoring

```
from(bucket: "DB Project")
  |> range(start: timeRangeStart)
  |> filter(fn: (r) => r["_measurement"] == "mem")
  |> filter(fn: (r) => r["_field"] == "available_percent"
    or r["_field"] == "used_percent")
  |> filter(fn: (r) => r["host"] == "<host_name>")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

Disk capacity monitoring

```
from(bucket: "DB Project")
  |> range(start: timeRangeStart)
  |> filter(fn: (r) => r["_measurement"] == "disk")
  |> filter(fn: (r) => r["_field"] == "used_percent")
  |> filter(fn: (r) => r["host"] == "<host_name>")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

Disk IO monitoring

```
from(bucket: "DB Project")
  |> range(start: timeRangeStart)
  |> filter(fn: (r) => r["_measurement"] == "diskio")
  |> filter(fn: (r) => r["_field"] == "iops_in_progress"
    or r["_field"] == "read_time" or r["_field"] == "write_time")
  |> filter(fn: (r) => r["host"] == "<host_name>")
  |> aggregateWindow(every: v.windowPeriod, fn: mean,
    createEmpty: false)
  |> yield(name: "mean")
```

CPU usage monitoring

```
from(bucket: "DB Project")
  |> range(start: timeRangeStart)
  |> filter(fn: (r) => r["_measurement"] == "cpu")
  |> filter(fn: (r) => r["_field"] == "usage_system"
    or r["_field"] == "usage_user" or r["_field"] == "usage_idle")
  |> filter(fn: (r) => r["cpu"] == "cpu-total")
  |> filter(fn: (r) => r["host"] == "<host_name>")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

Network Traffic Monitoring

```
from(bucket: "DB Project")
  |> range(start: timeRangeStart)
  |> filter(fn: (r) => r["_measurement"] == "net")
  |> filter(fn: (r) => r["_field"] == "packets_recv"
    or r["_field"] == "packets_sent")
  |> filter(fn: (r) => r["host"] == "<host_name>")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

System-level process monitoring

```
from(bucket: "DB Project")
  |> range(start: timeRangeStart)
  |> filter(fn: (r) => r["_measurement"] == "processes")
  |> filter(fn: (r) => r["_field"] == "running" or r["_field"] == "sleeping"
    or r["_field"] == "blocked" or r["_field"] == "total")
  |> filter(fn: (r) => r["host"] == "<host_name>")
  |> aggregateWindow(every: v.windowPeriod, fn: mean, createEmpty: false)
  |> yield(name: "mean")
```

Individual process monitoring

```
from(bucket: "DB Project")
  |> range(start: timeRangeStart)
  |> filter(fn: (r) => r["_measurement"] == "procstat")
  |> filter(fn: (r) => r["pidfile"] == "<pid_file>")
  |> filter(fn: (r) => r["_field"] == "cpu_usage" or
    r["_field"] == "memory_data"
    or r["_field"] == "memory_usage" or
    r["_field"] == "num_threads"
  |> aggregateWindow(every: v.windowPeriod, fn: mean,
    createEmpty: false)
  |> yield(name: "mean")
```

### 2.7 Business Logic Controller

For each of the use cases, we have the front end interfaces described previously, where the user can choose the attributes to be displayed. Then, based on the attributes selected, the backend will generate the appropriate query and call the underlying Influx database to fetch the data to be displayed. The time-series database will be built and returned by InfluxDB and will contain the required host or process-related data, and the Giraffe library will be used to interactively visualize this data in the webpage.

## 3 Testing

### 3.1 Test plan

For each use case, the plan involved specifying the input parameters with either all of them being valid, or some subset of them being invalid. In any case where at leasy one of the required input parameters is invalid, an error message should be displayed by the webpage and the corresponding graph should fail to plot.

### 3.2 Functional testing

For each use case, if the combination of required parameters taken as input (e.g. host name + process name) does not correspond to valid data in the Influx database, then the message 'Empty Table Received' is displayed as an error in place of the graph that would have been plotted.

Also, if data does not exist onwards from the start time specified by the user as input, then this same error occurs.

In all other cases, the required data is retrieved from the database by the backend, and plotted correctly according to the attributes specified.

### 3.3 Load testing

Our load testing plan was as follows. To test the application, we would get some pre-loaded data obtained from various systems (lab machines or systems of team members) using InfluxDB. Once the data is obtained over a certain time frame (say, around a day), we would load it into the time-series database which is connected with our web application, to display the results.

We would do load testing via running specified use cases on the loaded data, especially the ones involving the process monitoring interface form and that take up a lot of resources or have a lot of disk operations, etc. The metrics we would measure during this load testing include average response time, peak response time, error rate, requests per second, throughput.

Our user interface was well-suited to do this testing, since we have taken the desired start time and end time for each use case query as parameters entered by the user. So we could theoretically try querying each use case with different, large time ranges and observe the results.

However, our website loads the data directly from InfluxDB and not from a saved and stored csv. Due to time constraints, we were unable to leave InfluxDB open long enough to perform the required load testing directly from the database.

## 3.4 Analysis of unachieved goals

- Alert creation and notifications:
  We were planning to use Influx DB itself for handing alerts and `monitor` package from flux's standard library for creating checks on certain fields, and sending notification to our web server through a `POST` request using `monitor.notify` function but we were unable to send the notification using the function due to an error which we couldn't resolve.

- Visualizing multiple metrics together in a single graph where suitable:
  We were unable to do this due to the giraffe library used for plotting presenting some issues in displaying multiple line plots together. When multiple fields are filtered into the result in a query, all of them get concatenated into the same column of the table, making it difficult to separate and plot them directly using giraffe.

- Quality of service use case:
  This was not implemented due to lack of time to understand the Flux query syntax required for this particular query (selecting the three process names with the highest cpu/memory usage and then running the query to retrieve their metrics).