InfluxDB

(Process Metric Dashboards with InfluxDB)

(Group 8)

Aman Raj Singh, Nagapandu Potti,
Rakesh Dammalapati, and Tarun Gupta Akirala
University of Florida
Department of Computer & Information Science & Engineering
Gainesville, FL 32611, USA
{amansingh7, npotti, rdammala, takirala}@.ufl.edu

December 6, 2016

Abstract

Understanding the status and performance of your system's infrastructure is vital for the deployments in production environment[2], and for the users of applications to have a seamless experience. Even in the cases of performance related issues such as a user experiencing slowness with any of the applications, the root cause of the problem needs to be figured out immediately by following an easy and effective approach. Formulating such effective strategies requires a structured, well designed application that generates customizable metric dashboards and a detailed understanding on the rate at which the system metrics data such as cpu, memory, disk usage, network, IO, processes and services etc., is generated, format of the data, and in exploring the database that is capable of dealing and storing that data. This seems to be a problem of DevOps category, whose motive is to create a system environment reliable by taking preemptive steps and by continuously monitoring applications performance and end user experience. Realizing that the metrics data such as cpu usage, memory, network and processes etc., is generally time series data which is a sequence of data points typically consisting of measurements made from the same source over a time interval, and the data generation could be in the order of per milliseconds or seconds or hours etc., choosing the apt database is crucial instead of jumping to the conclusion of using traditional relational databases because the relational databases simply won't work in handling the time series data in the range of billions of data points and in meeting the read-write speeds[12] required in this context. Even the design needs to be schema less and flexible because of the variations in the data generated from different systems. One of the time series databases namely InfluxDB, of late has become very popular in dealing with problems of this kind, by overcoming the limitations posed by other such time series databases. Many companies have started using InfluxDB. One of the use cases says, "Mozilla uses InfluxDB to store important metrics about the performance of applications and devices running Firefox OS. By regularly testing their device builds and keeping that data in InfluxDB, they are able to run automated jobs which query the database and inform them when performance regressions occur. They heavily utilize the ability to aggregate data using mathematical functions to determine if builds have decreasing performance" [10]. Now that the right database is chosen for our purpose, the motive of this paper is to implement a web based application for visualizing the desired summaries and dashboards of the system metrics to spot negative trends over time and generate alerts [21] by just able to select the options from the user friendly UI, thereby auto building the queries based on the users selections and hit the database to pull out the data to generate graphs and summaries, also to showcase the outstanding features of the database to prove that InfluxDB is tailor-made in the areas of custom DevOps monitoring for storing process metrics data, and for visualizing, analyzing and summarizing the time series data.

Contents

1 Introduction

Although a time series database might just sound like a regular database with a timestamp column and data is ordered by that column, there are certain unique features that make time series database (InfluxDB) fit specifically for this kind of data as opposed to a traditional sql database such as postgresql or mysql. InfluxDB is a time series database built to handle large amount of reads and writes parallely. InfluxDB has a more complex model that allows for tracking events such as measurements like cpu load, processes etc., InfluxDB overcomes the limitations of the other databases such as incompatibility with the raw event data, and the inefficiency of legacy sharding mechanisms. The common trait of such "metric" data is that events happen in real time and the users or administrators care about summarizing, visualizing and analyzing the data in real time. Such visualizations play with the data at different levels of precision and different levels of granularity, and such complex model requires billions of data points to scale horizontally.

InfluxDB provides an easy to use HTTP API to build tooling around. You can read and write data via HTTP, manage the cluster from HTTP and it also has a security model to read and write data directly from the browser. InfluxDB makes it efficient and easy to automatically clear out old data. It's very common in time series to have high precision data for a short period of time and lower precision or downsampled data for months or years. If you are using other databases, you actually may have to write application level code to deal with this. Continuous queries in InfluxDB are powerful and are useful for several different use-cases, such as to downsample the raw data, to view the real time data and to analyze the data efficiently. Often, this downsampled data may be needed many times in the future and repeatedly computing the same rollups is inefficient. Continuous queries precomputes these expensive query results and stores the downsampled data as another measurement in real time [6]. A lot of times people working with time series data have some other process running where it will either monitor the data in real time as it comes in and outputs down samples and summaries or they will do it in batch, and for big data there will be a mapreduce that runs once an hour or once a day to build these summarizations, but it's kind of extra effort to always write that code to accomplish this, but InfluxDB's data store model and its inbuilt tool makes its implementation easy. InfluxDB is distributed by design, and in fact it is built in Go due to which it has an intrinsic advantage when it comes to distribution. One of the important reasons to have distribution is for reliability. It prevents data loss by placing replicas of the data on different nodes in the cluster. The next important reason is that it offers performance improvement. If a server is being bombarded with 20000 writes per second, then splitting the data into two or more separate nodes will split the load and considerably improve both read and write performances. Sharding helps in scaling out the data in different nodes which improves query and write performance by balancing out the load. In general terms, sharding is the process of splitting data into chunks based on some field. In case of InfluxDB, sharding is done automatically on the timestamp field.

These interesting features and capabilities made influxDB find its applications in the areas of IoT because sensors and devices in IoT architectures emit time-series data, and it can be sent to InfluxDB. InfluxDB also finds its applications in custom DevOps monitoring, Real time analytics and Cloud & Openstack. The use case for custom DevOps monitoring and visualizations on a web based application is our area of interest for the context of this paper. Monitoring means collecting metrics from disparate systems, applications, networking and infrastructure components [12]. InfluxDB's extensible design and its ability to integrate with telegraf collector tool that supports more than 30 types of inputs and 10 types of outputs can be easily extended to support many different sources of data. Telegraf collects the data in a format InfluxDB can consume [12].

Process Metric Dashboards and Monitoring systems help detect failures and identify the root cause of the events that caused the failure [2]. Although there are few such systems that are already deployed, often these systems cannot keep up with the humongous data, or sometimes are simply incompatible with the new applications and infrastructure that's being rolled out [12]. This motivates us to create custom process metric dashboards with visualizations and monitoring solutions using extensible open source technologies such as

InfluxDB and it's associated components. The challenges that were faced by the existing systems such as forced encapsulation, inability to integrate with other systems, not able to handle heterogenous data, fixed schema, scalable design for Big Data architectures, asymmetric metric granularity affecting the query, data compression and rollups etc., are proved to be solvable by the use of InfluxDB. This sounds realistic and feasible because many companies have started using InfluxDB and its inbuilt time-series platform to develop process metric dashboards. One such use case taken from ebay says "InfluxDB along with Grafana is used to measure various data quality metrics within the Experimentation platform that are monitored daily" [10].

This paper proposes a web based application for extending the features and capabilities of the existing process metric dashboards, visualizations and monitoring systems while making the fullest use of InfluxDB and some components of the InfluxDB's inbuilt TICK (Telegraf, InfluxDB, Chronograf, Kapacitor) stack to prove to be a best fit in terms of integration. The proposed solution is a web based application with a user interface where the user can select some options to view different types of system metrics in any time interval and perform analysis on it. The application would parse the user input, build the query itself (thus abstracting the query building process from the user), hits the database with the query, generates the summaries and shows the processed output to the user. It is that easy to use without the user having the knowledge of query language syntax unlike other traditional metric dashboards which force the user to write queries and requires manual intervention for fixing dashboards etc., Our solution is also expected to send the real time threshold alerts via email, for e.g., receive an alert when CPU has maintained an average usage higher than 70 for more than 15 minutes [13]. Our application is expected to be scalable to take metrics from different systems when configured.

2 Related Work

From pre-stage flat-file system[27], to relational and object-relational systems, database technology has gone through several generations and its history that is spread over more than 40 years now. The relational database systems have seen a unparalleled growth and is the widely used database system. Continued research on enhancing the capabilities of the database pertaining to specific use cases lead to the strategic development of many other databases, out of which InfluxDB is our area of interest. A software system that is optimized for handling time series data[1], arrays of numbers indexed by time is a time series database. The evolution of such time series databases is driven by the fact that the software with complex logic or business rules and high transaction volume[16] for the time series data is not practical with relational database management systems. Even the flat file databases were not a viable option when the data and transaction volume reaches a maximum threshold that is set by the capacity of the individual servers. Also the queries for historical data, replete with time ranges and roll ups[17] and arbitrary time zone conversions are difficult in a relational database. Time series database systems provides a specific model to such use cases. Relational data models grow "downwards" by adding rows, whereas the time series data is different [20]. An appropriate model to handle such data would be if that rows are identified by a primary key, and they grow "sideways" with the wide rows model.

There are a few existing time-series databases, OneTick - a commercial time series database, OpenTSDB [14], Graphite, RRD - open source time series database. All the above time series databases have some limitations. For example, OneTick fails to track when there are hundreds of unique time series. Also a specific ability that was missing is to store irregular time series. That is, series created by specific events, like trades in a market.

So a new time-series based database is built called InfluxDB. Thousands of organizations around the globe already use InfluxDB. They are using it to monitor their network infrastructure, and in the fields of security, container infrastructure, solar panels, agriculture, scientific experiments, user analytics, business intelligence, home automation, and countless other specific use cases. InfluxDB is built on top of Cassandra

and uses Redis for real time indexing. It is written in Scala and is formulated as a set of web services [8].

The challenges faced in developing the InfluxDB are to scale storage of time series data or the elasticity factor, where every company is focusing, and another important challenge is to build distributed systems for the time series workload. The future of InfluxDB is driven on solving the limitations of the database that arise due to high cardinality there by reduce the consumption of memory when indexing millions of records. They are also working on making the system more intelligent for automatic rollups, querying of lower precision data and a good visual representation of data over the time frames ranging from a very short span to years. Also adding in query language functionality like performing calculations across multiple measurements, combining series together through different calculations, sub-queries, ranking queries and much more are the challenges ahead to solve [8].

3 System Description

InfluxDB is basically an open source database that is designed particularly to deal with time series data. Working with time series data means that the data store demands high performance and high availability. The core component of the proposed web application i.e., InfluxDB has no external dependencies, so it installs within minutes. In spite of its less memory footprint, InfluxDB is scalable and flexible for complex deployments. Our advanced database system has characteristic features such as schema-less design with support for one million values per second, a native HTTP API to bypass server side code to manage, time-centric functions and an easy to use SQL-like query language (InfluxQL), data that can be tagged to allow efficient queries, and more importantly in answering queries in real-time by indexing every data point as it comes in and that data point becomes available within 100 milli second. The proposed web application that does both Real-Time Analysis and Historical Analysis on metrics data makes fullest of all the key features mentioned. As the metrics data is generated continuously every millisecond, we need some sort of queries that runs automatically and periodically on real time data and store the result in a specified measurement (Similar to a table in SQL). Continuous queries serve that purpose. So this process of pre-computing and storing the aggregated query results so that they are ready when you need them would significantly speed up the summary queries in the proposed web application. Well, after a particular point of time the data needs to be purged. InfluxDB provides a mechanism to do so, i.e., Retention policy. The proposed web application uses the default retention policy defined by the InfluxDB because it can efficiently auto-expire stale data.

While InfluxDB is schema-less, an entry into the InfluxDB database (measurement) will have some format. All the data in InfluxDB have column called time. Time stores the timestamps associated with the data. Other columns are fields (mandatory) and tags (optional). Fields are made up of field keys and field values. Field values are the data which can be strings, floats, booleans or integers. Field values are always associated with a timestamp. Tags record the metadata about the measurement. Tags are made up of tag keys and tag values. Understand that tags can be indexed while fields are not.

3.1 Query Language and Operations

InfluxDB Query Language (InfluxQL) syntax is similar to SQL query language tailored for querying ranges of time. Objects in the database can be accessed using user defined references, identifiers. They are in fact strings and single quoted values. These are the references with identifiers: usernames, database names, measurement names, field keys, tag keys, retention policy names.

To interact and operate on advanced database system, HTTP API, Command Line Interface or the built-web administration GUI can be used. The InfluxDB's command line interface is influx, an interactive shell for the database, whereas the Admin UI is available by default at port 8083. To Query the data, you can enter any valid InfluxQL in the Query box. These below queries and figures illustrate the system aspects of our advanced database system and the basis for database interaction of the proposed web application. As

our web application is supposed to collect metrics data, and those metrics are cpu, disk, memory, network statistics, processes, swap and system. Executing the query ”show measurements” on the system will be as shown in the figure 1.

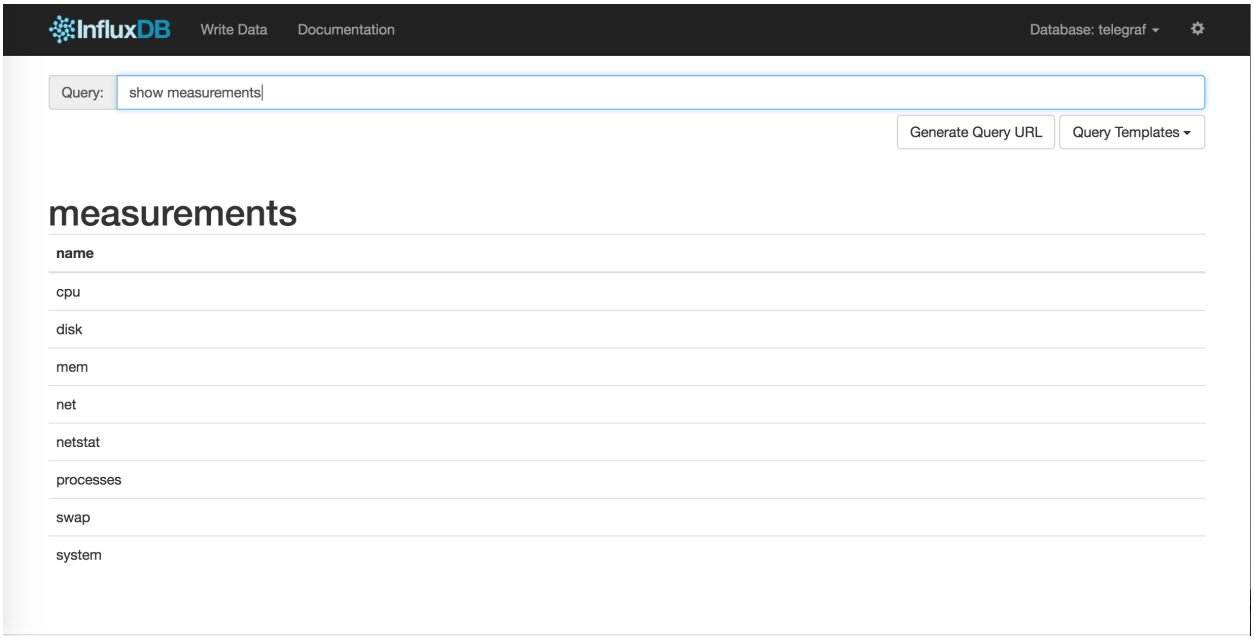
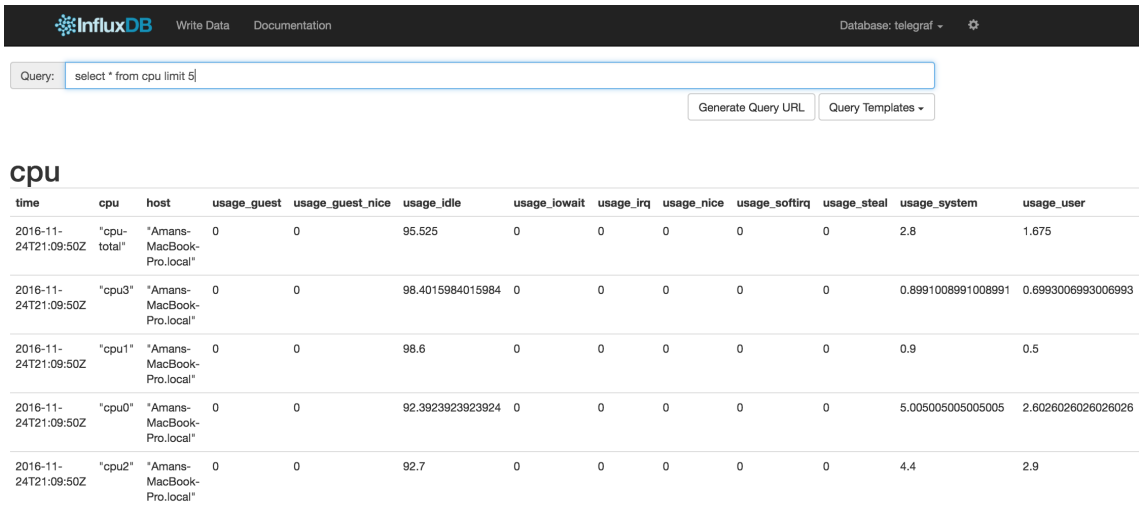


Figure 1: Query Operation: Show measurements

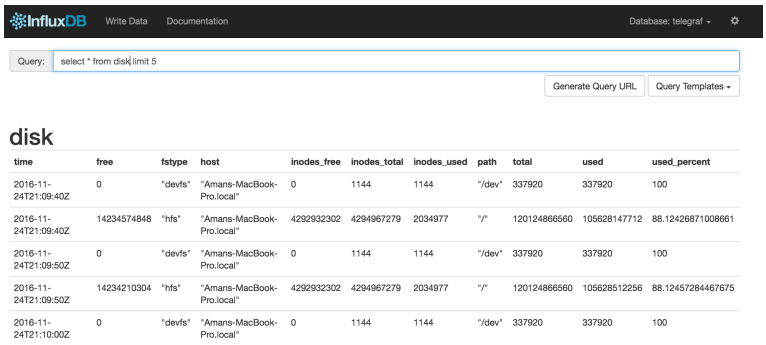
```
Select * from cpu limit 5
```



time	cpu	host	usage_guest	usage_guest_nice	usage_idle	usage_iowait	usage_irq	usage_nice	usage_softirq	usage_steal	usage_system	usage_user
2016-11-24T21:09:50Z	"cpu-total"	"Amans-MacBook-Pro.local"	0	0	95.525	0	0	0	0	0	2.8	1.675
2016-11-24T21:09:50Z	"cpu3"	"Amans-MacBook-Pro.local"	0	0	98.4015984015984	0	0	0	0	0	0.8991008991008991	0.6993006993006993
2016-11-24T21:09:50Z	"cpu1"	"Amans-MacBook-Pro.local"	0	0	98.6	0	0	0	0	0	0.9	0.5
2016-11-24T21:09:50Z	"cpu0"	"Amans-MacBook-Pro.local"	0	0	92.3923923923924	0	0	0	0	0	5.005005005005005	2.6026026026026026
2016-11-24T21:09:50Z	"cpu2"	"Amans-MacBook-Pro.local"	0	0	92.7	0	0	0	0	0	4.4	2.9

Figure 2: Query Operation: Select query

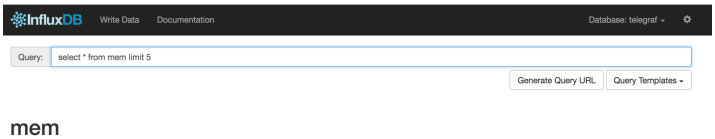
```
Select * from disk limit 5
```



time	free	fstype	host	inodes_free	inodes_total	inodes_used	path	total	used	used_percent
2016-11-24T21:09:40Z	0	"devfs"	"Amans-MacBook-Pro.local"	0	1144	1144	"/dev"	337920	337920	100
2016-11-24T21:09:40Z	14234574848	"hfs"	"Amans-MacBook-Pro.local"	4292932302	4294967279	2034977	"/"	120124866560	105628147712	88.12426871008661
2016-11-24T21:09:50Z	0	"devfs"	"Amans-MacBook-Pro.local"	0	1144	1144	"/dev"	337920	337920	100
2016-11-24T21:09:50Z	14234210304	"hfs"	"Amans-MacBook-Pro.local"	4292932302	4294967279	2034977	"/"	120124866560	105628512256	88.12457284467675
2016-11-24T21:10:00Z	0	"devfs"	"Amans-MacBook-Pro.local"	0	1144	1144	"/dev"	337920	337920	100

Figure 3: Query Operation: Select query

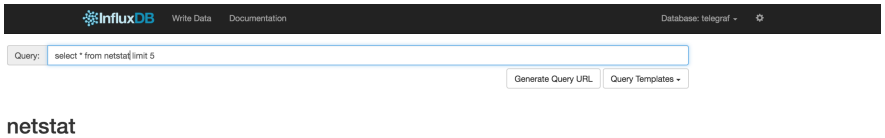
```
Select * from mem limit 5
```



time	active	available	available_percent	buffered	cached	free	host	inactive	total	used	used_percent
2016-11-24T21:09:40Z	813764608	806944768	9.394073486328125	0	0	23408640	"Amans-MacBook-Pro.local"	783536128	8589934592	7782989824	90.60592651367188
2016-11-24T21:09:50Z	778698752	801300480	9.328365325927734	0	0	20066304	"Amans-MacBook-Pro.local"	781234176	8589934592	7788634112	90.67163467407227
2016-11-24T21:10:00Z	861642752	852340736	9.922552108764648	0	0	45916160	"Amans-MacBook-Pro.local"	806424576	8589934592	7737593856	90.07744789123535
2016-11-24T21:10:10Z	867061760	815058944	9.488534927368164	0	0	23597056	"Amans-MacBook-Pro.local"	791461888	8589934592	7774875648	90.51146507263184
2016-11-24T21:10:20Z	816910336	831307776	9.677696228027344	0	0	18620416	"Amans-MacBook-Pro.local"	812687360	8589934592	7758626816	90.32230377197266

Figure 4: Query Operation: Select query

```
Select * from netstat limit 5
```



time	host	tcp_close	tcp_close_wait	tcp_closing	tcp_established	tcp_fin_wait1	tcp_fin_wait2	tcp_last_ack	tcp_listen	tcp_none	tcp_syn_recv	tcp_syn_sent	tcp_time_wait	udp_socket
2016-11-24T21:30:30Z	"Amans-MacBook-Pro.local"	0	8	0	19	0	0	0	15	0	0	0	0	4
2016-11-24T21:30:40Z	"Amans-MacBook-Pro.local"	0	8	0	19	0	0	0	15	0	0	0	0	4
2016-11-24T21:30:50Z	"Amans-MacBook-Pro.local"	0	8	0	21	0	0	0	15	0	0	0	0	4
2016-11-24T21:31:00Z	"Amans-MacBook-Pro.local"	0	8	0	20	0	0	0	15	0	0	0	0	4
2016-11-24T21:31:10Z	"Amans-MacBook-Pro.local"	0	8	0	40	0	0	0	15	0	0	0	0	5

Figure 5: Query Operation: Select query

Series is basically collection of data in the InfluxDB's data structure that shares a measurement, tag set, and retention policy. However, in the deep sense, time structured merge is the underlying storage engine for InfluxDB.

```
Show series
```

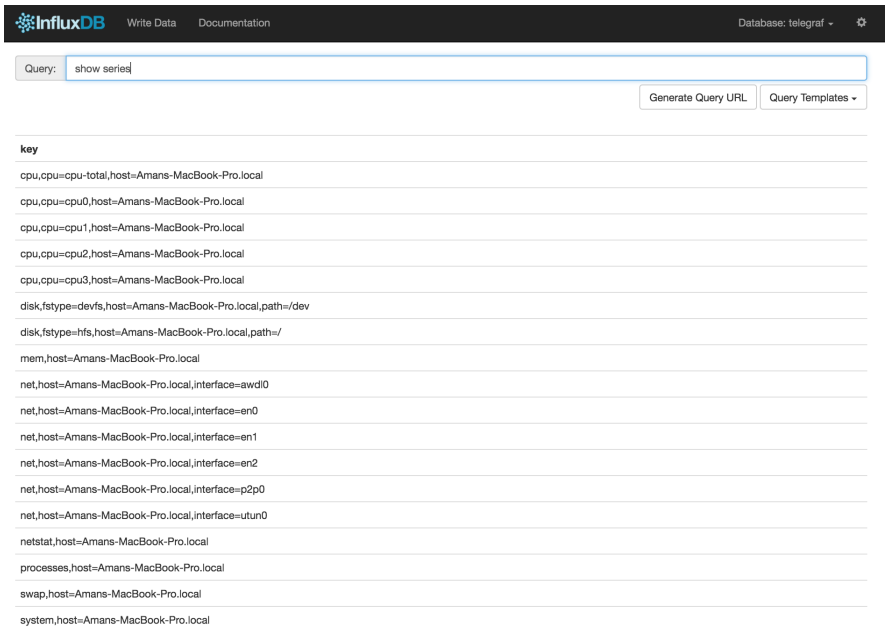


Figure 6: Query Operation: Series

3.2 System Architecture

InfluxDB is one of the components in TICK stack, and all these components can be easily integrated with each other. TICK stands for Telegraf, InfluxDB, Chronograf, Kapacitor. "Telegraf is an open source agent written in Go for collecting metrics and data on the system it's running on or from other services. Telegraf writes data it collects to InfluxDB in the correct format" [9]. The key features of Telegraf are minimal memory footprint, ability to collect metrics like CPU, I/O, memory etc., extensible design with support for 30 different types of inputs and outputs, and configurable connection to InfluxDB database. "Chronograf is a graphing and visualization application that you deploy behind your firewall to perform adhoc exploration of your Influx data. It includes support for templates and a library of intelligent, configurable dashboards for many types of data sets" [5]. The key features of Chronograf is a configurable connection to scalable number of InfluxDB servers, and a smart query builder designed to work with large datasets. "Kapacitor is InfluxDB's native data processing engine. It can process both stream and batch data from InfluxDB. Kapacitor lets you plug in your own custom logic or user defined functions to process alerts with dynamic thresholds, match metrics for patterns or compute statistical anomalies" [16]. The key features of InfluxDB are stream data from InfluxDB or query from InfluxDB, trigger events/alerts based on complex or dynamic criteria, and ability to add custom user defined function to detect anomalies. All the components of TICK stack combined for building a process metric dashboards is shown in figure 7.

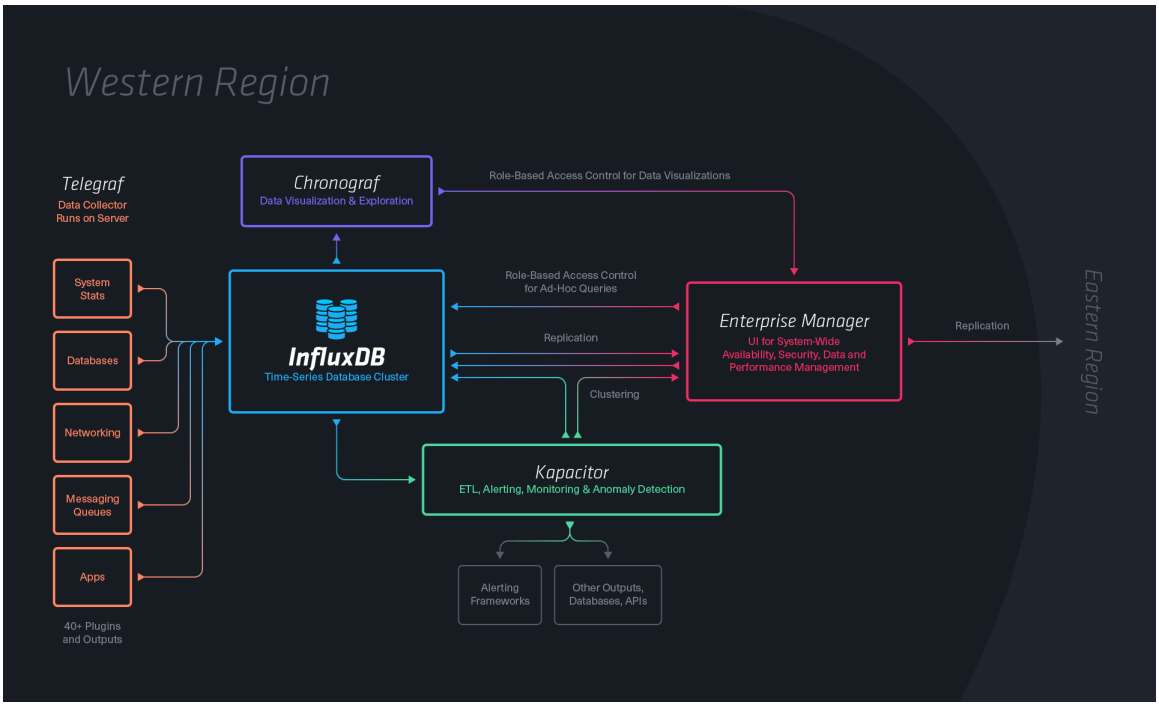


Figure 7: Architecture of InfluxDB TICK Stack [11]

3.2.1 Methodology 1: Visualization using Chronograf

The plan is to develop a web based application on top of the TICK stack shown in figure 1, with Node.js as the backend server. The user's UI selection (e.g., CPU utilization for the last 4 hours etc.,) is first sent to the server over HTTP, then the server makes a request to Chronograf's API which interacts with influxDB and generates a graph in response which is embedded on the user's dashboard. The user can also set custom monitoring alerts, which is handled by the Kapacitor API's in the backend. Note that Grafana is another visualization tool very similar to Chronograf. The application might use either Chronograf's APIs or Grafana's APIs and see whichever works out the best and efficient. "Node.js is an open-source, cross-platform JavaScript runtime environment for developing a diverse variety of tools and applications, and its design choices aim to optimize throughput and scalability in Web applications" [19]. The application will also be deployed on Google App Engine, a platform for building scalable web applications and mobile back-ends. App engine also scales the application automatically in response to the amount of traffic [18].

3.2.2 Methodology 2: Visualization using D3.js

While retaining most of the architecture shown in the figure 1 and the application specification as the same, the visualization tool component (Chronograf or Grafana) can be replaced by advanced JavaScript Libraries like D3.js to generate visualizations and see whichever works out to be the best and efficient. Even in this methodology, the Telegraf collects the metrics data from different sources to InfluxDB, and the user's UI selection is passed to the backend server (Node.js) where the appropriate InfluxQL query is constructed based on the user selection to fetch stats from InfluxDB database. The response obtained is plotted as graphs or charts with the help of D3.js. "D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, SVG, and CSS" [4].

4 Application Description

After trying out the above mentioned methodologies, we have decided to go with the below explained approach on the roots of ease and efficiency that best suits our requirements while working with time series databases such as InfluxDB so as to play around with the stored data, and in fact to make meaningful use of data. In the final implementation of the proposed web application, these are the following set of technologies used. Ruby Sinatra as the Web Server, HTML and Bootstrap for the UI, JavaScript to program the behavior of web pages, AJAX to request, receive and send data to and from the InfluxDB database server over the HTTP API, and an open source JavaScript toolkit Rickshaw for creating interactive time series graphs based on the data stored in the time series database InfluxDB.

"Sinatra is an open source software web application library and domain-specific language written in Ruby" [26]. The motive behind choosing Sinatra is it's thin MVC architecture, and because Sinatra is great for micro-style. There is just a thin boundary between Model and Controller, and Sinatra works really well for the API, and the smaller sites [15]. "HyperText Markup Language (HTML) is the standard markup language for creating web pages and web applications" [25]. "Bootstrap is the most popular HTML, CSS and JS framework for developing responsive, mobile first projects on the web, and Bootstrap makes front-end development faster and easier" [3]. "JavaScript is the programming language of HTML and the web" [23]. "AJAX, asynchronous JavaScript and XML is a set of web development techniques using many web technologies on the client-side to create asynchronous web applications, and with Ajax, web applications can send data to and retrieve from a server asynchronously without interfering the display and behavior of the existing page" [24]. In the context of the proposed web application, Ajax requests are in the form of InfluxDB queries (Influx Query Language) that hit the InfluxDB database server. The response of the query is then passed to the Rickshaw, a JavaScript library for generating interactive graphs based on that response data. "Rickshaw provides the elements that are needed to create interactive graphs for visualization: renderers, legends, hovers, range selectors, etc." [22]. Range selectors feature that this graphing toolkit provides is exactly in line with how a user of the proposed web application selects the desired range options from the graphs, that in fact queries from the InfluxDB database. Rickshaw is built on d3.js, so that the graphs are drawn with standard SVG and styled with CSS. [22]

4.1 Implementation Details and System Configuration

The data sources for this application include cpu, memory, swap, processes and disk metrics, and as these data sources are generated continuously in real time, it's size will be in the range of billions of data points.

The Figure 8 below shows the list of some of the metrics that will be recorded when our solution is deployed in a windows based environment. These metrics will be analogous even if the solution is moved to linux distribution.

A sample of the configuration showing on how to start collecting cpu metrics into InfluxDB is shown below.

```
telegraf -sample-config -input-filter cpu:mem -output-filter \
        influxdb > telegraf.conf
```

Figure 8 shows CPU metrics, General Metrics, Memory Metrics and Disk Metrics.

CPU metrics

Metric	Fields	Description
winproc.cpu	type=user	The number of seconds the CPU has spent executing instructions in user space.
winproc.cpu	type=system	The number of seconds the CPU has spent executing instructions in kernel space.

General metrics

Metric	Description
winproc.uptime	The number of seconds since the process was created.
winproc.threads	The number of threads being used by the process.

Memory metrics

Metric	Fields	Description
winproc.mem.bytes	type=working_set	The number of bytes of physical memory used by the process's working set. This is the amount of memory that needs to be paged in for the process to execute.
winproc.mem.bytes	type=peak_working_set	The peak working set size for the process since creation time, in bytes.
winproc.mem.bytes	type=paged_pool	The paged-pool usage, in bytes. This is the amount of bytes of swappable memory in use.
winproc.mem.bytes	type=peak_paged_pool	The peak paged-pool usage, in bytes.
winproc.mem.bytes	type=nonpaged_pool	The nonpaged pool usage, in bytes. This is the amount of memory in use that cannot be swapped out to disk.
winproc.mem.bytes	type=peak_nonpaged_pool	The peak nonpaged pool usage, in bytes.
winproc.mem.bytes	type=pagefile	The current pagefile usage, in bytes. The is the total number of bytes the system has committed for this running process.
winproc.mem.bytes	type=peak_pagefile	The peak pagefile usage, in bytes.
winproc.mem.bytes	type=rss	The current resident size in bytes. This should be the same as the working set.
winproc.mem.bytes	type=vms	The current virtual memory size in bytes. This does not include shared pages.

Disk metrics

Metric	Fields	Description
winproc.disk.ops	type=read	The number of disk read requests issued by the process since creation time.
winproc.disk.ops	type=write	The number of disk write requests issued by the process since creation time.
winproc.disk.bytes	type=read	The number of bytes read from disk by the process.
winproc.disk.bytes	type=write	The number of bytes written to disk by the process.

Figure 8: Disk Metrics [19]

Starting the telegraf server, you could see the measurements of the metrics in the telegraf database created in InfluxDB as shown below.

```
> SHOW measurements
name: measurements
--------------------------
name
cpu
mem
```

The interaction diagram of all the components in the web application is as shown in the Figure 9.

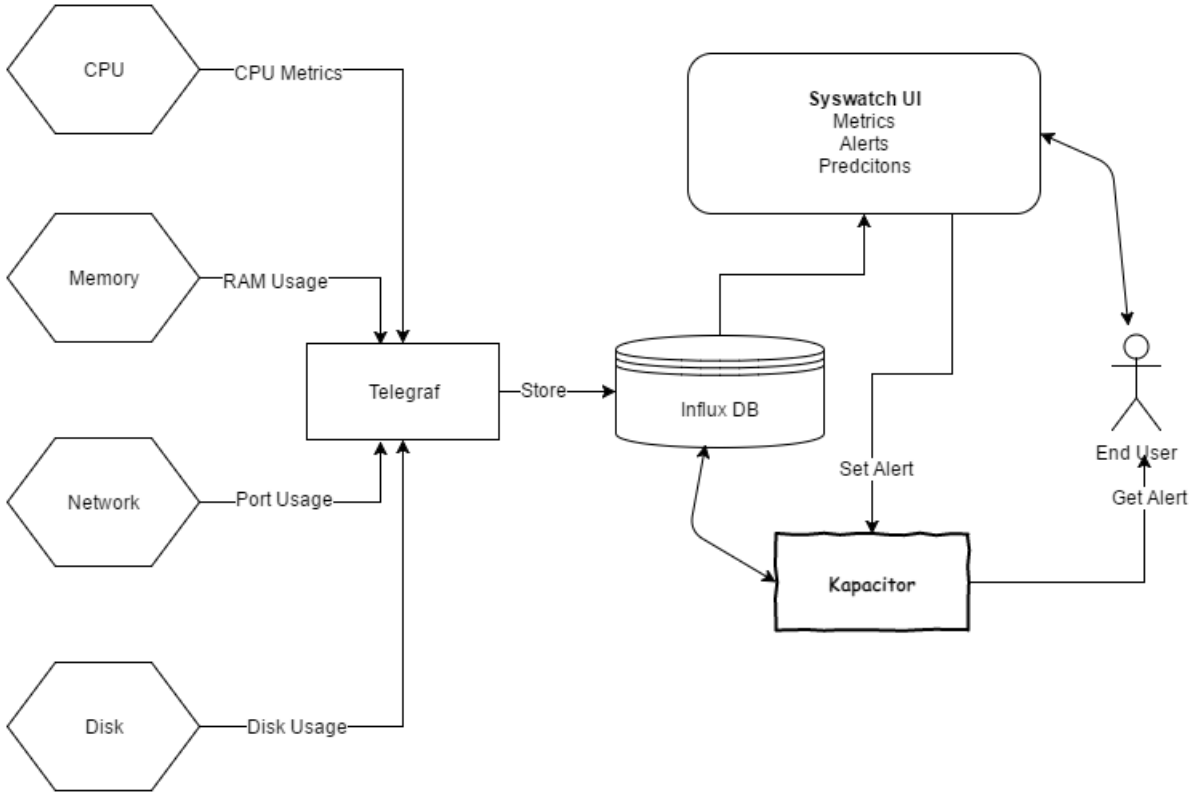


Figure 9: Interaction among web application components

The application can run on all the modern machines. The kind of the use case defines the system requirements. A low load application needs 2-4 core CPU and 2-4GB of RAM, a moderate load application needs 4-6 cores and 8-32GB of RAM and high load application might need as much as 8+ cores and >32GB of RAM. The load on the system depends on the number of writes and queries that comes in. An application can be classified as low load if the writes are <5K and queries of <5 per second, similarly writes of <100K and queries of <25 for moderate load and writes of >100K and queries >25 per second for high load applications [7]. The current development is being carried on Linux distribution on a quad core Intel - i7 processor aiming for a low to moderate load applications.

The following steps show how the system is implemented and the commands to make the application up and running.

Install InfluxDB

```
curl -sL https://repos.influxdata.com/influxdb.key | sudo apt-key add -
source /etc/lsb-release
echo "deb https://repos.influxdata.com/${DISTRIB_ID,,} ${DISTRIB_CODENAME} stable"
| sudo tee /etc/apt/sources.list.d/influxdb.list
sudo apt-get update && sudo apt-get install influxdb
```

Now that InfluxDB is installed, start influx server.

```
sudo service influxdb start
```

Influx GUI can be seen at http://localhost:8083/

As explained previously, Telegraf is data collector tool, and these are the steps to install, configure and run telegraf.

```
brew install telegraf
or, sudo apt-get install telegraf
telegraf.conf file is already present in the `telegraf` dir of the repo
telegraf -config telegraf.conf
```

By default, it will start collecting all the cpu metrics and will write that to influxdb in a database called telegraf. This is the list of measurements you should see, name cpu, disk, diskio, kernel, mem, processes, swap, and system.

Now that the database is set up, these steps will show on how to develop a web application that will interact with this data, analyse this data, and predict the future data and trends, and generate alerts.

Sinatra is the framework for serving the pages (routes) and controller for handling requests. Install rvm, ruby and start Sinatra server

```
curl -sSL https://get.rvm.io | bash
rvm install 2.2.1
cd influx-time-series/Project
gem install sinatra
gem install twilio-ruby
cd influx-time-series/Project
ruby app.rb
Go to http://localhost:4567/
```

As explained previously kapacitor is the InfluxDB's native data processing engine through which we can plug in custom logic to process alerts with user defined thresholds.

To install, configure, and start kapacitor.

```
brew install kapacitor
kapacitor.conf file is already present in the `kapacitor` dir of the repo
cd influx-time-series/kapacitor
kapacitord -config kapacitor.conf
```

To define and run a streaming task to trigger alerts.

```
Copy all .tick and .sh files in influx-time-series/kapacitor to /tmp dir
kapacitor define mem_alert -type stream -tick /tmp/mem_alert.tick -dbrp telegraf.autogen
kapacitor enable mem_alert
kapacitor define cpu_alert -type stream -tick /tmp/cpu_alert.tick -dbrp telegraf.autogen
kapacitor enable cpu_alert
```

Go to http://localhost:4567/alerts and set alert thresholds for mem and cpu. Alerts will start appearing on the /alerts page.

The following screenshots show the Web based user interface design and the corresponding advanced database system queries that resulted in the appropriate graphs. Note that the metrics data is generated continuously i.e., every milli second and hence the database size is over 10000 records within short amount of time the telegraf (data collector tool) is started. So the graphs shown here are based on the real data which are fetched based on Influx queries, and Rickshaw graph library is used for plotting the graphs on that data. See figure 10 for how a rendered graph will look like. This graph is updated as if it moves in real time by fetching graph data points by ajax at regular intervals and perform graph.update(). Range selector horizontal scroll bar is for the user to see the data for different periods of time, for e.g., cpu metric data for the last 3 hours, or for the last 5 minutes. Bootstrap is used for a neat view, headers and drop-downs that point to the other pages on selection. The web interface by default will show the query results grouped by 10s, however this can be easily adjusted by the sliding along the range selector present below every graph from the UI.

CPU:

```
select mean(usage_user) FROM cpu WHERE time > now() - 1h GROUP BY time(10s), cpu
```

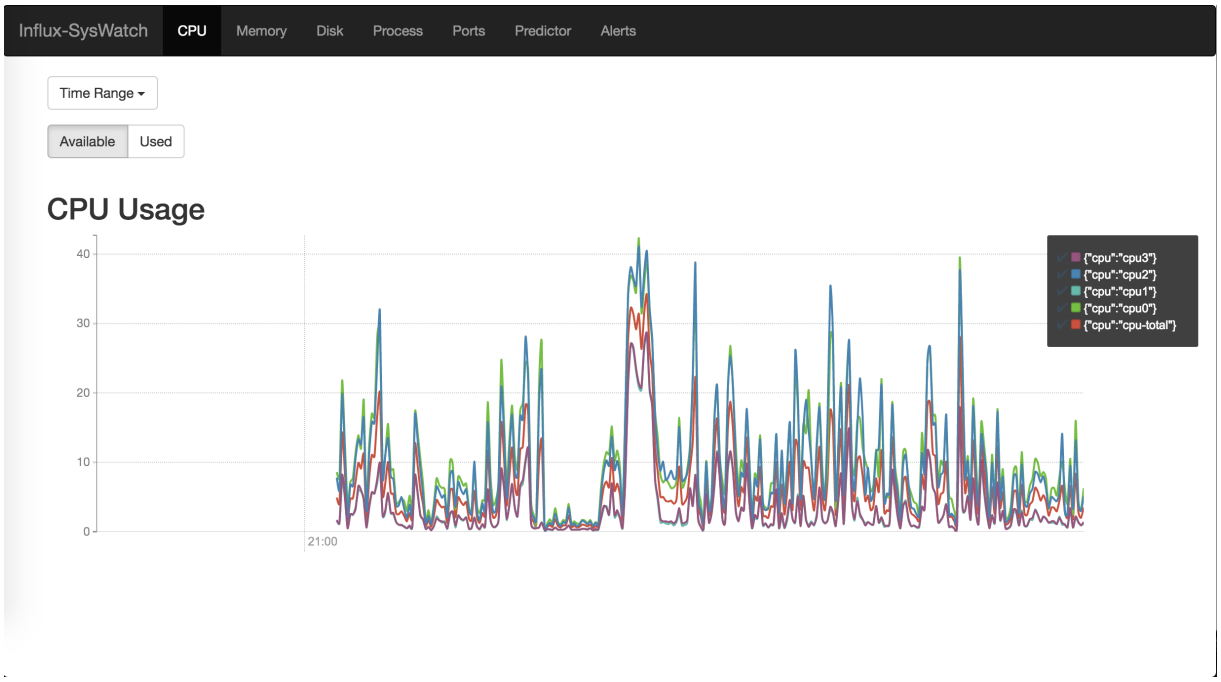


Figure 10: CPU metrics graph

```
Select mean(usage_idle) FROM cpu WHERE time > now() - 1h GROUP BY time(10s), cpu
```

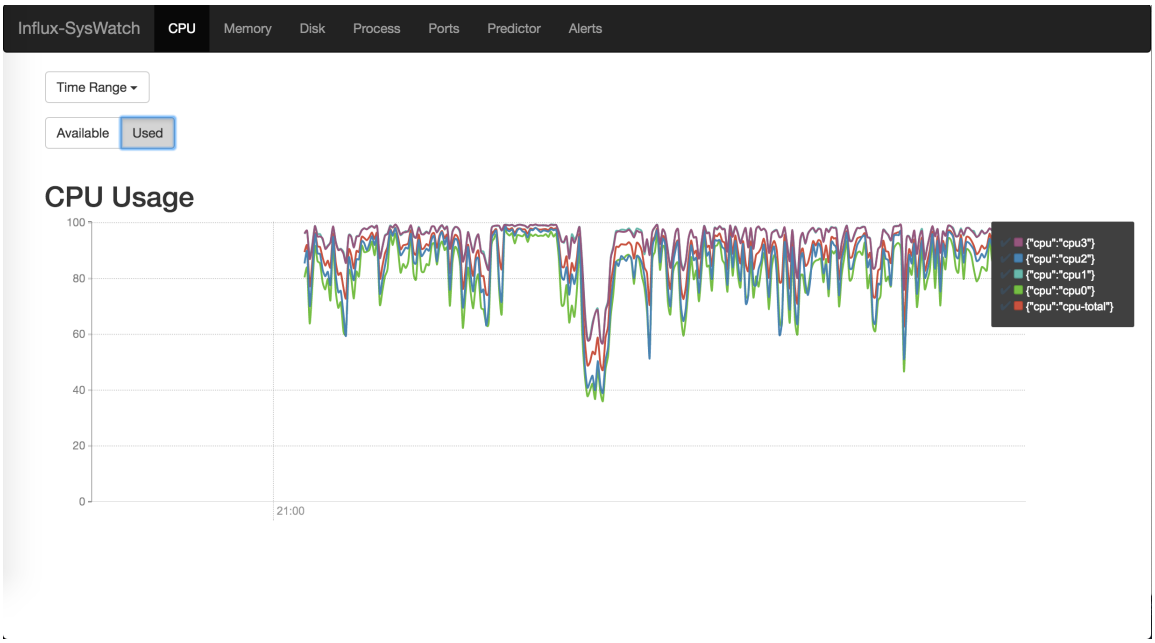


Figure 11: CPU metrics graph

Memory:

```
select mean(available) / 1000000000 from mem where time > now() - 1h GROUP BY time(10s);
```

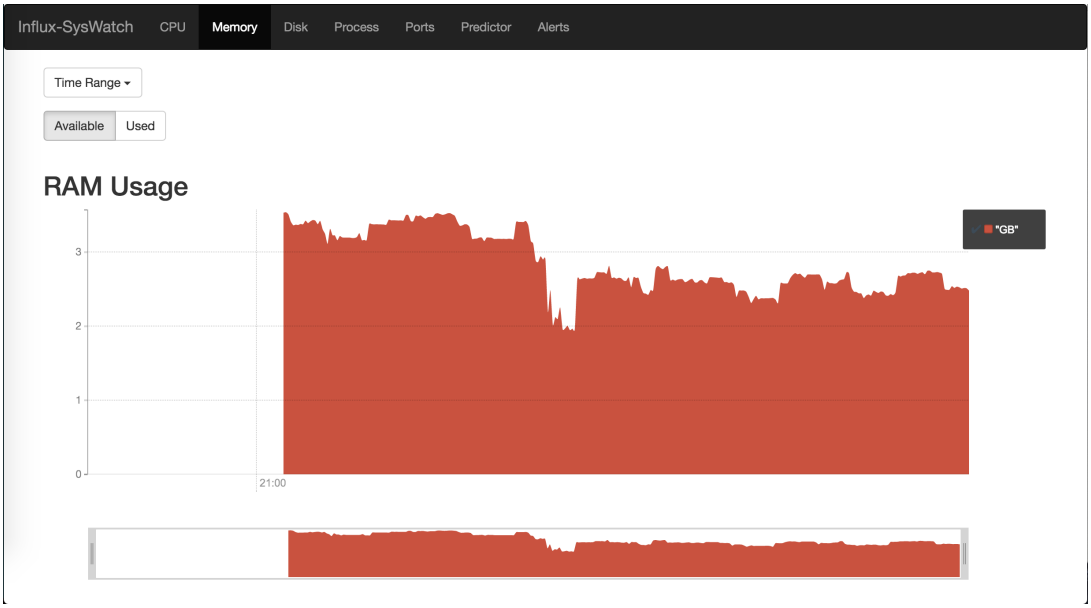


Figure 12: Memory metrics graph

```
select mean(used) / 1000000000 from mem where time > now() - 1h GROUP BY time(10s);
```

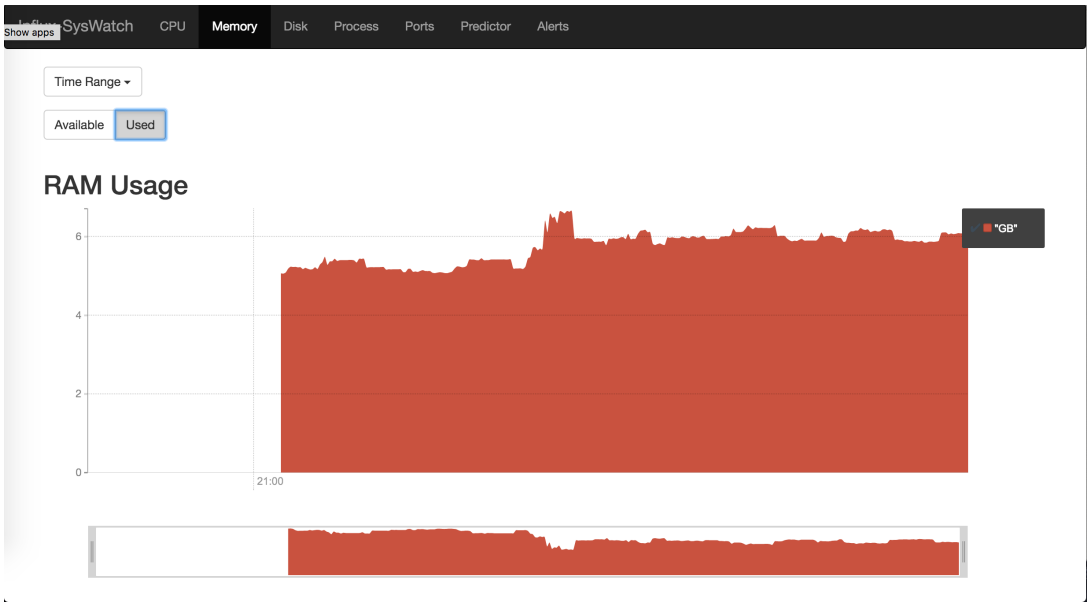


Figure 13: Memory metrics graph

Disk:

```
select DIFFERENCE(MEAN(used)) / 1000000 AS MB from disk where time > now() - 1h AND
fstype='hfs' GROUP BY time(10s)
```

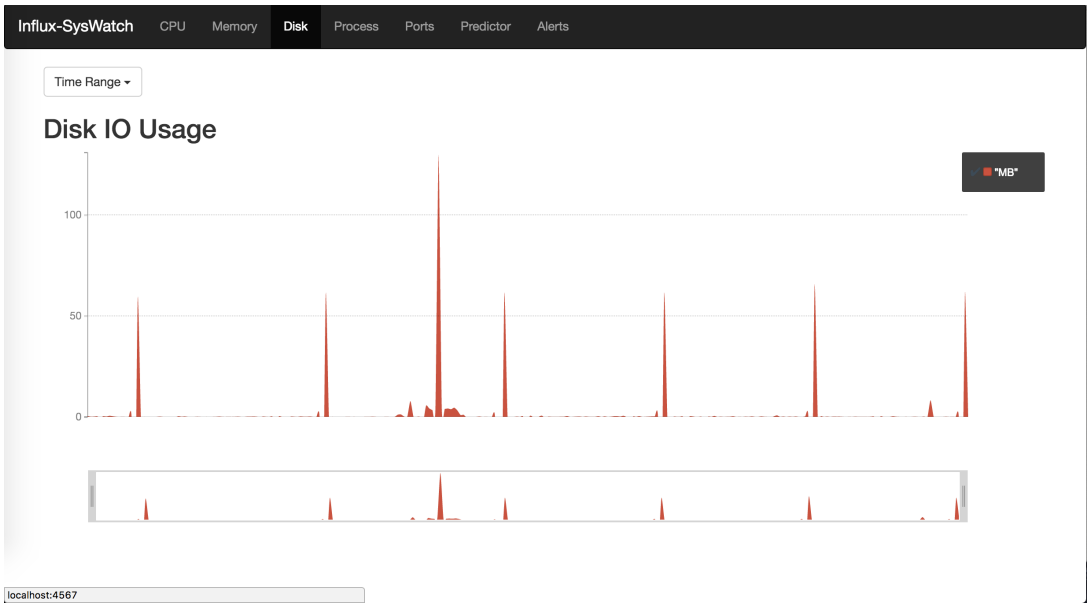


Figure 14: Disk metrics graph

Processes:

```
select MEAN(running) AS Running from processes where time > now() - 1h GROUP BY time(10s)
```

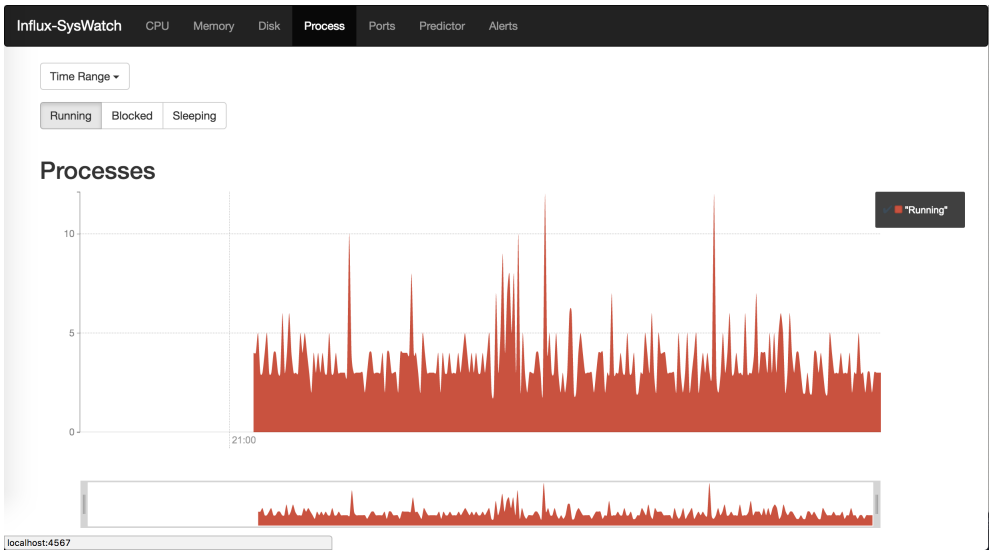


Figure 15: Process metrics graph

```
select MEAN(blocked) AS Blocked from processes where time > now() - 1h GROUP BY time(10s)
```

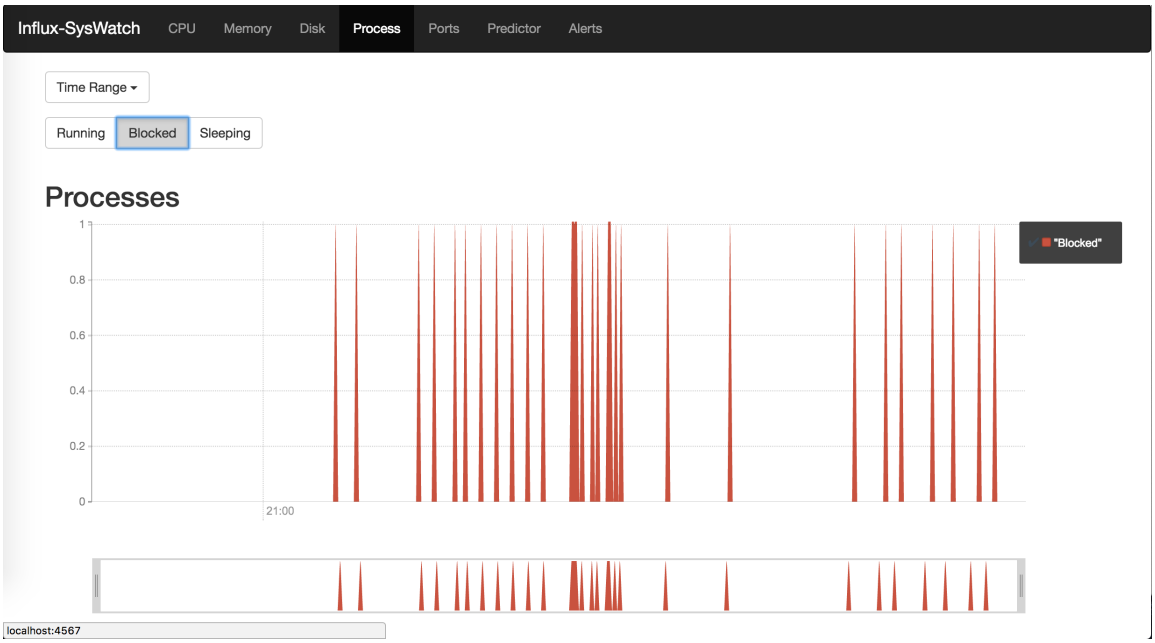


Figure 16: Process metrics graph

```
select MEAN(sleeping) AS Sleeping from processes where time > now() - 1h GROUP BY time(20s)
```

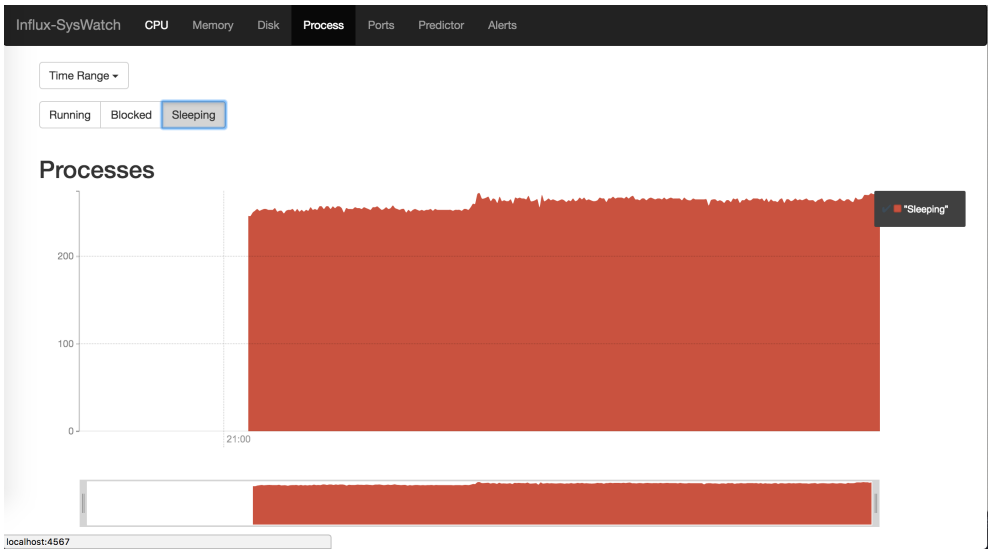


Figure 17: Process metrics graph

Ports:

```
select MEAN(tcp_established) AS TCPEst from netstat where time > now() - 1h GROUP BY time(10s);
```

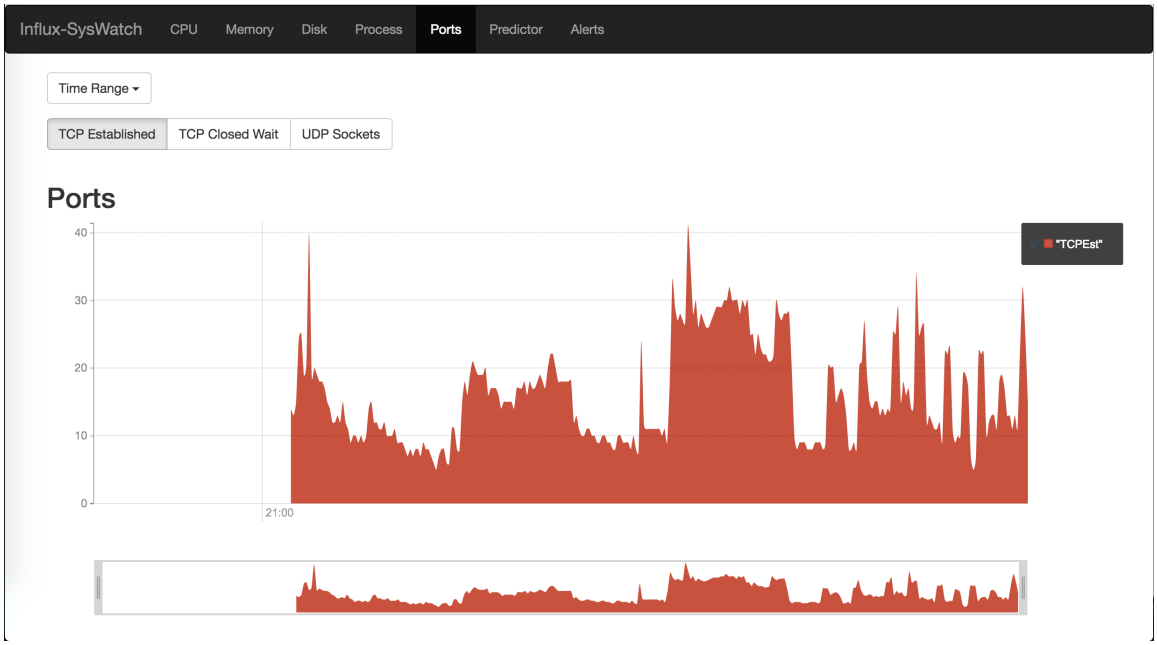


Figure 18: Port metrics graph

```
select MEAN(tcp_close_wait) AS TCPCloseWait from netstat where time > now() - 1h GROUP BY time(20s);
```

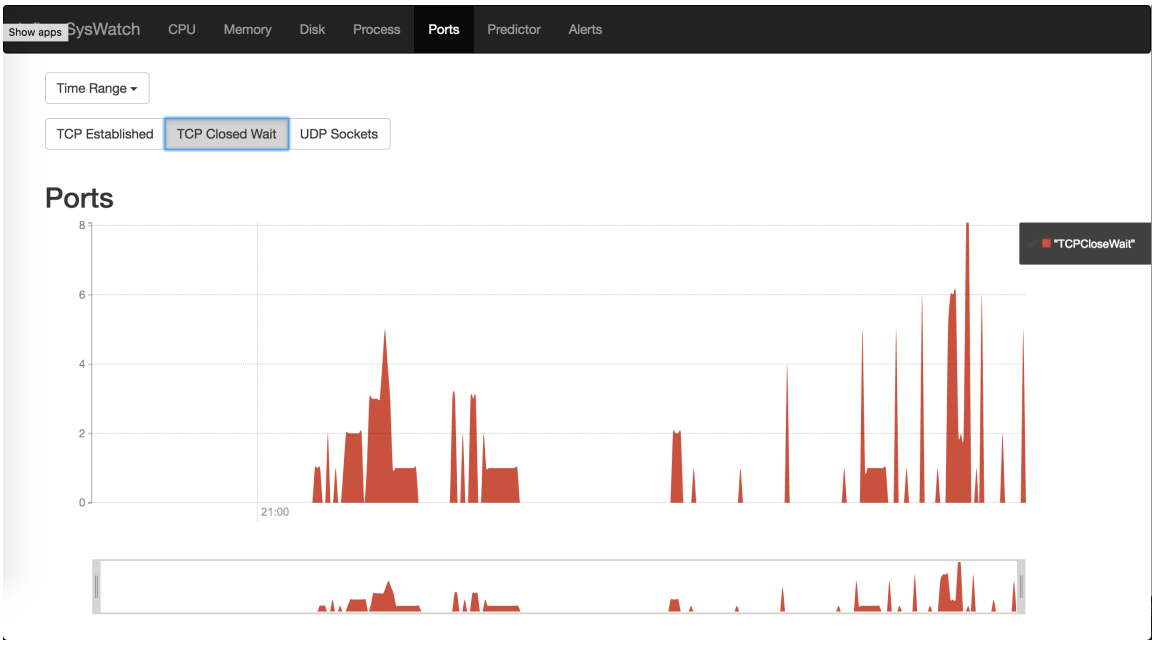


Figure 19: Port metrics graph

```
select MEAN(udp_socket) AS UDPSocket from netstat where time > now() - 1h GROUP BY time(20s);
```

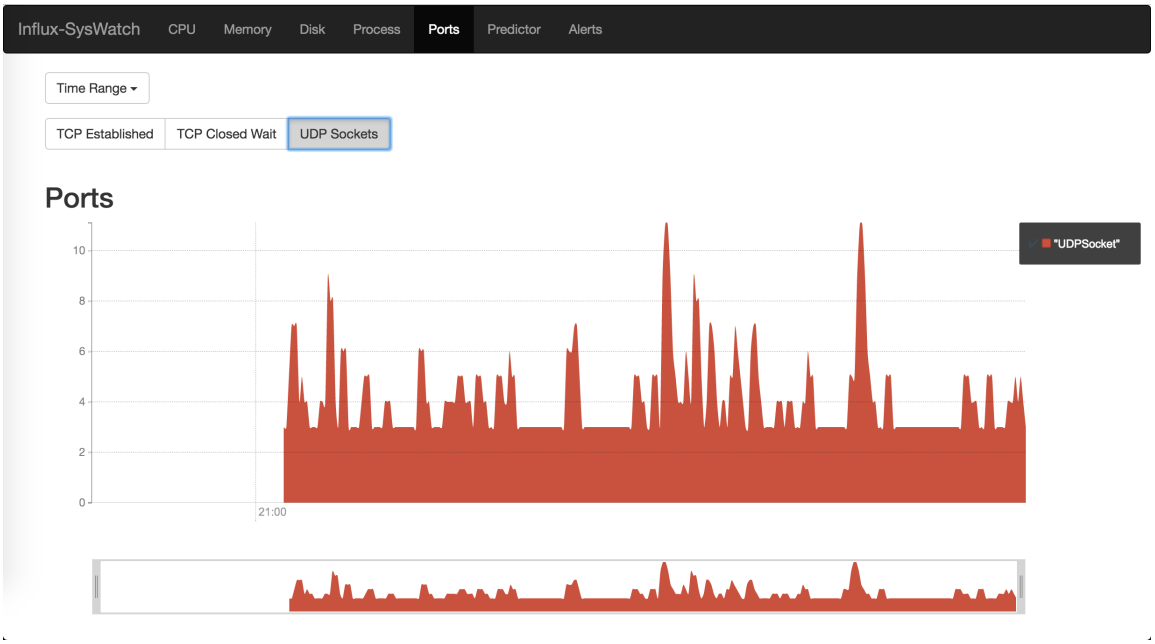


Figure 20: Port metrics graph

Another interesting feature implemented in the proposed web application is the usage of HOLT_WINTERS() function. It returns N number of predicted values for a single field, and this function's capabilities are used

to predict when data values will cross a given threshold. This allows the DevOps Monitoring use case to take a proactive action instead of reactive action. This function can also compare predicted values with actual values to detect anomalies in the data.

```
select holt_winters_with_fit(MEAN(udp_socket),1000,10)/1000000000 AS UDP FROM mem
where time > now() - 8h group by time(20s)
```

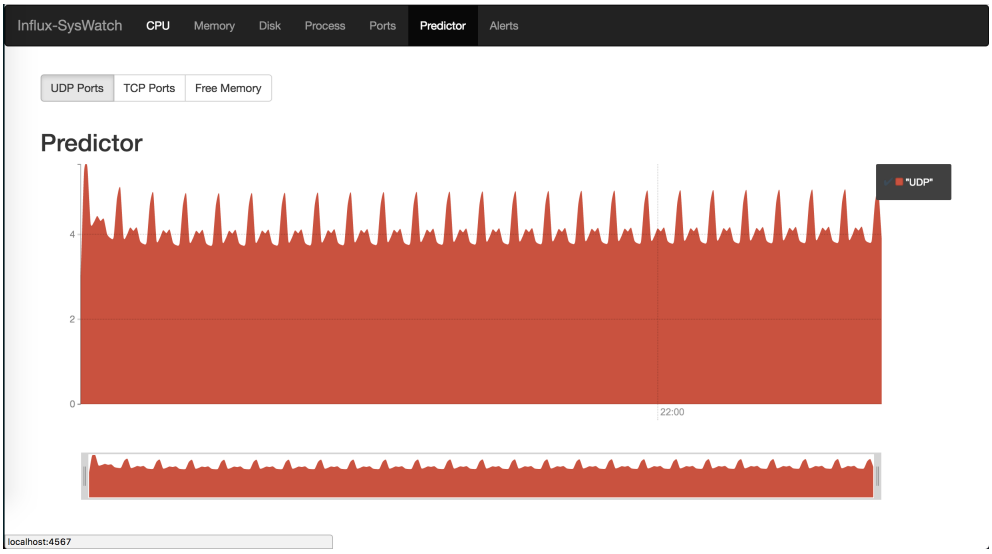


Figure 21: Holt Winters metric prediction graph

```
select holt_winters_with_fit(MEAN(tcp_established),1000,10)/1000000000 AS TCP FROM mem
where time > now() - 8h group by time(10s)
```

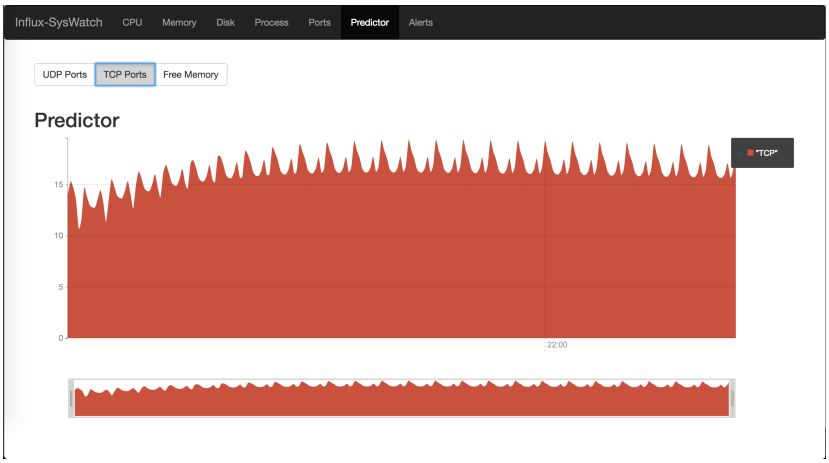


Figure 22: Holt Winters metric prediction graph

```
select holt_winters_with_fit(MEAN(mem),1000,10)/1000000000 AS GB FROM mem where
time > now() - 8h group by time(10s)
```

Figure 23: Holt Winters metric prediction graph

One of the important features is the ability for an user to set a desired threshold value for certain metric parameters so that the user would be able to get alerts when the metric values in real time exceed the specified threshold value. The following screenshot shows the alerts page on the UI that lists all the alerts. Users will be alerted via email as well.

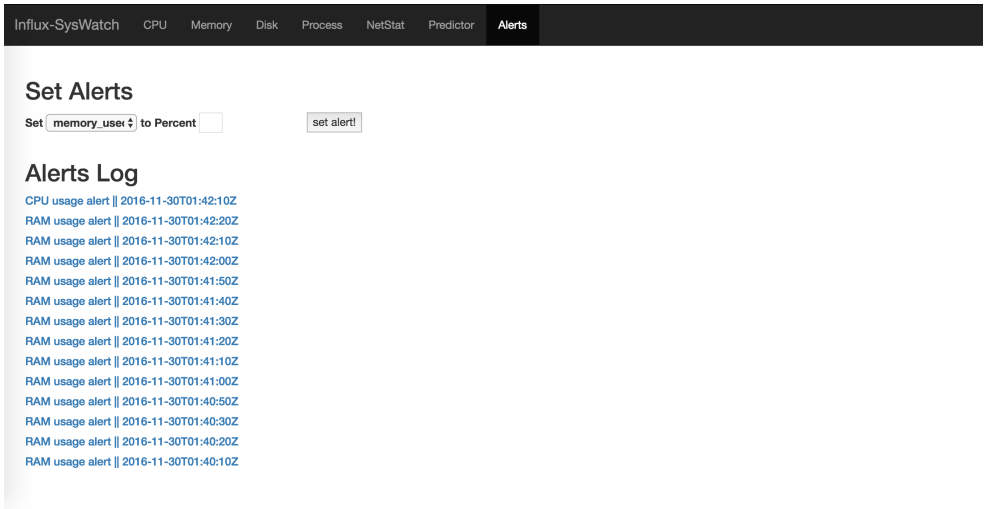


Figure 24: Alerts graph

5 Conclusion

While the traditional relational databases are suitable for most kind of applications, but at what cost? efficiency? maintenance? and reliability? It is understood that building those databases specific to type of data being stored, application specific databases based on the use cases and requirements of the application is very much essential in this era of technology to reduce maintenance costs, increase efficiency and improve the reliability and scalability. The implemented web application that this paper explained could have used relational database, but the above said factors might not be fulfilled if InfluxDB, the database specifically build to handle time series data had not been used. Well the time series data means continuous data in the range of billions of data points. This Big Data could have been handled by technologies like Hadoop etc., which might turn out to be quite complex, but InfluxDB provided a simple way to accomplish the required task with its features capable to serve both small scale and enterprise level applications. The scope of this web application is beyond the metric types collected in this system, and is extendable to collect many other metric types by specifying in the configuration file.

References

[1] Mikhail J. Atallah, Christos Berberidis, Walid G. Aref, Mikhail Atallah, Ioannis Vlahavas, and Ahmed K. Elmagarmid. Time series databases.

[2] Matthew Hodgkins Blog. Windows Metric Dashboards with InfluxDB and Grafana. `https://hodgkins.io/windows-metric-dashboards-with-influxdb-and-grafana`. 2016-03-29.

[3] Bootstrap. . `http://getbootstrap.com/`. 2016.

[4] D3. Data-Driven Documents. `https://d3js.org/`. 2015.

[5] Paul dix. Chronograf. `https://www.influxdata.com/time-series-platform/chronograf/`. 2016.

[6] Paul dix. Continuous Queries. `https://docs.influxdata.com/influxdb/v0.8/api/continuous_queries/`. 2016.

[7] Paul dix. Hardware Sizing Guidelines. `https://docs.influxdata.com/influxdb/v0.10/guides/hardware_sizing/`. 2016.

[8] Paul dix. InfluxDB 1.0 GA Released: A Retrospective and Whats Next. `https://www.influxdata.com/influxdb-1-0-ga-released-a-retrospective-and-whats-next/`. 2016-09-08.

[9] Paul dix. Telegraf. `https://www.influxdata.com/time-series-platform/telegraf/`. 2016.

[10] Paul Dix. Testimonials. `https://www.influxdata.com/testimonials/`.

[11] Paul dix. TICK stack architecture. `https://influxdata.com/wp-content/uploads/2015/09/TICK-Stack.png`.

[12] Paul Dix. Use Cases for Time-Series Data. `https://www.influxdata.com/use-cases/custom-devops-monitoring/`. 2016.

[13] dotcom monitor. Windows Performance Counter Monitoring. `https://www.dotcom-monitor.com/windows-performance-counter-monitoring/`. 2016.

[14] T. Goldschmidt, A. Jansen, H. Koziolek, J. Doppelhamer, and H. P. Breivold. Scalability and robustness of time-series databases for cloud-native monitoring of industrial processes. *2014 IEEE 7th International Conference on Cloud Computing*, pp. 602–609, June 2014.

[15] Darren Jones. Rails or Sinatra: The Best of Both Worlds? https://www.sitepoint.com/rails-or-sinatra-the-best-of-both-worlds/. 2016.

[16] Xiao-Ying Liu and Chuan-Lun Ren. Fast subsequence matching under time warping in time-series databases. *2013 International Conference on Machine Learning and Cybernetics*, volume 04, pp. 1584–1590, July 2013.

[17] U. Mori, A. Mendiburu, and J. A. Lozano. Similarity measure selection for clustering time series databases. *IEEE Transactions on Knowledge and Data Engineering*, 28(1):181–195, Jan 2016.

[18] Google Cloud Platform. App Engine. https://cloud.google.com/appengine/. 2016.

[19] Scalyr. Windows Process Metrics. https://www.scalyr.com/help/monitors/windows-process-metrics. 2015.

[20] Baron Schwartz. Time-Series Databases and InfluxDB. http://www.xaprb.com/blog/2014/03/02/time-series-databases-influxdb/. 2014-03-02.

[21] server density. Windows server monitoring. https://www.serverdensity.com/windows-server-monitoring/.

[22] Shutterstock. Rickshaw. http://code.shutterstock.com/rickshaw/. 2016.

[23] w3schools. JavaScript. http://www.w3schools.com/js/. 2016.

[24] Wikipedia. Ajax. https://en.wikipedia.org/wiki/Ajax_(programming). 2016.

[25] Wikipedia. HTML. https://en.wikipedia.org/wiki/HTML. 2016.

[26] Wikipedia. Sinatra. https://en.wikipedia.org/wiki/Sinatra_(software). 2016.

[27] Wikipedia. Time Series database. https://en.wikipedia.org/wiki/Time_series_database. 2016-09-16.