Designing and Implementing a Migration Strategy to Move Archived Data from Graphite to InfluxDB

Abhishek Kapoor
Supervisor: Herve Rousseau

August, 2018
Geneva, Switzerland

Abstract

The EOS operations team has been using Graphite for many years in order to monitor and manage large-scale storage systems which are getting data from different CERN experiments, be it LHC experiments such as ATLAS, CMS, ALICE, etc. or non-LHC experiments such as AEGIS, DIRAC, etc. As data rate increases and as clusters are getting bigger, EOS is reaching the limits of a single Graphite node instance to monitor their infrastructure. So, the viable solution is to move to a different database such as InfluxDB. Moreover, InfluxDB is the standard time series metrics storage system provided and supported by the IT department at CERN. In this report, We will be discussing the designing and implementation of such a migration strategy to move archived data from Graphite to InfluxDB effectively.

Keywords: Migration, InfluxDB, Graphite, Whisper, Grafana

1. Introduction

EOS is a low-latency, multi-PB disk-based storage service [3], which is used by different experiments at CERN, be it LHC experiments such as ATLAS, CMS, etc. or non-LHC experiments such as AEGIS, DIRAC, etc. [4] for storing the experiments data. EOS have a highly-scalable hierarchical name space and data can be accessed by using XROOT protocol, EOS now serves the physics use cases and the regular user use cases as well [3]. EOS services are divided into two different data centers, one at CERN, Geneva, Switzerland and other at WIGNER center in Budapest, Hungary [1].

As EOS is such a large storage service and as many big research projects are using it for storing data for analysis, monitoring is an important aspect here to maintain a reliable and secure service. Until recently, EOS team at CERN was using Graphite [6] as a monitoring tool to store and visualize different metrics, but as data rate increases and as clusters are getting bigger, EOS is reaching the limits of a single Graphite node instance to monitor their infrastructure. So, the viable solution is to move to a different database such as InfluxDB [8]. Moreover, InfluxDB is the standard time series metrics storage system provided and supported by the IT department at CERN. The project involved here is to evolve the existing monitoring scripts to write to InfluxDB, as well as design and implement a migration strategy to move the archived data from Graphite whisper files to InfluxDB database. Once this is done monitoring data can be visualized and analyzed on Grafana [5]. We will first be covering the workflow of previous monitoring strategy i.e. using Graphite, and then will explain the new monitoring solution i.e. using InfluxDB. We will then cover and explain the migration strategy, and the tool developed to migrate the archived data from Graphite to InfluxDB.

2. Graphite Monitoring WorkFlow

Figure 1: **Graphite Monitoring WorkFlow**: We are depicting the Graphite monitoring solution workflow that is being used at CERN for monitoring EOS servers. Python scripts are being used to filter and parse the metrics from the EOS servers and are sending the data to Graphite instance, which in turn sending the data to Grafana to create dashboards and visualize the metrics.

Until Recently, CERN IT Storage Department was using Graphite as a monitoring tool, to visualize and monitor EOS metrics. The normal workflow (see figure 1) of this monitoring solution was, firstly, python was used as the main scripting language to filter and parse the relevant metrics that are needed. For example, these metrics were IO, IoFuse, NS_OPS, NS, uptime,

Mgmerr, Fsck, FD, Group, Space, Node, Versions, etc. The python script was running on EOS instances and were sending these metrics/data to the single master Graphite instance, which was used to store and visualize these metrics. Now, these metrics were again filtered and queried using Grafana to get the relevant Graphs and were monitored by creating dashboards on Grafana. The problem occurred when the cluster size got increased and the Graphite instance was not able to handle such a load. As a result, the performance declined and InfluxDB comes into picture.

3. InfluxDB Monitoring WorkFlow

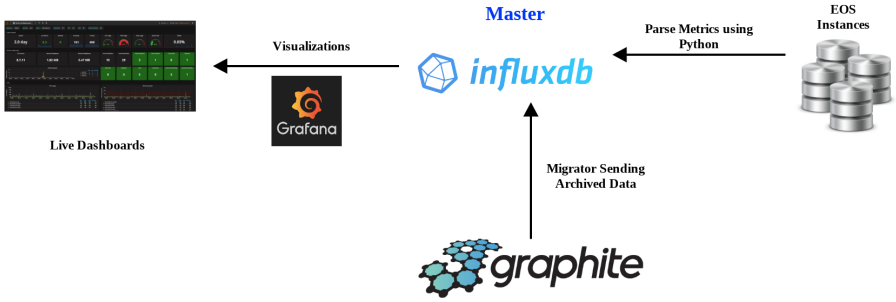

Figure 2: **InfluxDB Monitoring Workflow**: We are depicting the InfluxDb monitoring solution workflow that is being used at CERN for monitoring EOS. As can be seen, python scripts are being used to filter and parse the metrics from the EOS servers and are sending the data to InfluxDB central instance, which in turn sending the data to Grafana to create dashboards and visualize the metrics. We are also sending the archived graphite data from Graphite instance to the InfluxDB database, using the migrator.

As can be seen in figure 2, the InfluxDB monitoring solution workflow is somewhat same here, i.e. we are using python script to filter and parse the relevant metrics and sending these metrics to InfluxDB central instance. Here, we are using the migrator which will be sending the archived data from Graphite instance to the InfluxDb database over HTTP. Again, InfluxDB is used here as an optimal solution for storing time-series data and we are then sending the data to Grafana to create dashboard and monitor the EOS servers.

4. Migration Strategy and Tool

We now know what was the actual workflow we wanted to implement and how to migrate the archived data to InfluxDB. InfluxDB has many benefits over Graphite be it performance enhancement or flexible database schema [2]. We started by evolving our current python scripts which were sending metrics to the Graphite instance. The first issue was to generate and create the database schema for different metrics. Secondly, we have to modify our python scripts to send the metrics in the desired schema we created. Next, comes the migration of archived data from Graphite whisper files to InfluxDB database. We will now explain all these steps in the next subsections.

Designing Database Schema

InfluxDB has a flexible schema and a single database can have different schema for different measurements within that database. After looking into the Graphite database schema, we got to know that every measurement has a tag key associated with it, which is the instance name of the EOS server where the script is running. Keeping this mind, we created the schema depicted below as a JSON code. The schema contains name of the *"measurement"* which is actually the type of the metric, and contains tags such as *"instance"*, *"machine"* and so on. The field within a measurement name is the name of specific metric which we are filtering using the python script. Now, with this schema, we can add more relevant tags associated o certain type of metrics and can have a flexible schema which can be modified accordingly in the future without hampering the previous or archived data.

```
{
  "measurement": "measurement_name",        # Measurement
  "tags":{"instance":instance_name, ...},  # Tags
  "time": int(time.time()),                 # Time
  "fields": {metric_name:float(value)}      # Field
}
```

Evolving Python Scripts

The previous python script was used to filter the metrics and send it to Graphite. We kept the scripts incline to its previous version, just adding the desired filtering and schema pattern which will be used for InfluxDB. The

python client needs to connect to the InfluxDB database over HTTPS to send the metrics, so for this we are using the official InfluxDB python client [7]. As to have no external dependency, we have are using a definite version of InfluxDB Python client which is installed as a RPM package [11]. The full code of the python script is stored in the CERN Gitlab repository [12]. Here, we will just show how we are connecting to the InfluxDB database using InfluxDB python client. For full code please refer the repository.

```
from influxdb import InfluxDBClient
client = InfluxDBClient('host', 8080, 'user', 'passwd',
'dbname', ssl=True, verify_ssl=True)
```

We have to specify the *"host", "port number", database name", "username", "password"* to connect to a certain InfluxDB database running at a remote InfluxDB instance. As can be seen from figure 3, displays a Grafana dashboard where we are using InfluxDB database as the source and querying different metrics to get the desired graphs. Thus, our InfluxDB workflow is established and we are able to send the metrics to the database and are able to read or query it using Grafana as well.

Figure 3: **Grafana Dashboard**: Using InfluxDB database as a source here, we can query the metrics and create the relevant graphs which we need. The dashboard here shows few graphs related to EOS metrics such as *Free Space, Used Bytes, Write Throughput, Read Throughput, Current Readers, Current Writers, and Number of Directories.*

Migration Tool for Archived Data

We have established the workflow we wanted to achieve. The only thing left is migrating the archived data from Graphite whisper files (.wsp) to the InfluxDB instance, so that we can visualize and analyze the data over the past few years.

We are using the official Whisper migrator [9] as the base, but InfluxDB does not maintain it anymore and the current version of the official migrator doesn't work as well. The last commit was in June 2016. Still we are using it as the base and refined the code according to CERN use case, managed to make it run for the desired purposes. Also, edited and improved the code accordingly. The tool is using InfluxDB official golang ClientV2 library. It can be used in two modes:

```
# Get Whisper file information: This option displays,
# number of points in the file and oldest
# timestamp in the file.

> go migration.go -wspinfo -wspPath=whisper_folder_path

# Write to InfluxDB: It uses InfluxDB go library
# (clientV2) and migrates data by calling HTTP APIs.

> go migration.go -option=ClientV2 -wspPath=folder_path
-from=<2015-11-01> -until=<2015-12-30> -dbname=migrated
-host=host_adress -port=8086, -retentionPolicy=default
-tagconfig=config.json
```

Now, as EOS official Graphite whisper files directory has many subfolders (different metrics), and every subfolder (different metrics) has a different schema, directory structure and number of whisper files, We are using a workaround here to provide the "measurement", "tags", and "field" in the InfluxDB database directly from the Graphite directory structure rather than providing these parameters manually for every measurement. This means that the tool will read the directory path and based on the config file will assign the required parameters i.e. "measurement", "tags", and "field". All information regarding the usage of this tool can be found at the CERN Gitlab repository [10].

Tag Config File

The config file is required to specify the "measurement", "tags", and "field" name based on a given pattern. Please see the desired sample config file shown below. We have specified the name of the config file based on the metrics/measurements it is being used for.

```
[
  {
    "pattern": "home.user.Desktop.#TEXT1.#TEXT2.#TEXT3",
    "measurement": "#TEXT2",
    "tags": [
      {
        "tagkey": "instance",
        "tagvalue": "#TEXT1"
      }
    ],
    "field": "2"
  }
]
```

The **pattern** in the config file is based on the directory path that contains the whisper files. For example, the directory path for IO metrics is

```
/home/user/desktop/alice/io/filename.wsp
# So, for this the defined Pattern is
home.user.desktop.#TEXT1.#TEXT2.#TEXT3
```

As we are interested only in the subpath "alice/io/whisperfiles.wsp". Here, as per the pattern #TEXT1 will take the value as "alice", #TEXT2 will take the value as "io", and #TEXT3 will take the value as "filename". Now, we can use these values, rather than specifying manually (hardcode for each measurement), in the config file. For example, for the IO metrics:

```
measurement: "#TEXT2" (As measurement name should be io)
tags: {tagkey: "instance", tagvalue: "#TEXT3"}
(As tag key instance should be equal to alice here.)
field: "2"
```

The tool will automatically take these values and replace the actual values from the path mentioned initially.

Field Value

We are storing the # part of the pattern as an array here, separated by #. So for the above case, the array will be [TEXT1, TEXT2, TEXT3] i.e. [alice, io, filename]. Now, the field number signifies the values we don't want, i.e. the field value of "2" will ignore the first two indexes of the array and will join the rest indexes by "." separator. Here, we have only one value remaining in the array which is the filename thus we are not using "." here and the field name will go directly as the filename.

By this we can migrate the archived data stored in Graphite whisper files and send it to the InfluxDB database effectively, approximately 20 mins is taken by this tool to migrate the archived data for one instance.

5. Code Repositories

The relevant code repositories can be accessed at CERN Gitlab pages. Please contact Herve Rousseau as currently the repositories are private. Although the documentation is complete, if you still aren't able to understand anything then please contact: *kapoorabhishek24@gmail.com*.

- Python Script - `https://gitlab.cern.ch/abkapoor/eos-influx`
- RPM - `https://gitlab.cern.ch/abkapoor/python-influxdb-rpm`
- Migrator - `https://gitlab.cern.ch/abkapoor/whisper-influx`

6. Conclusion

Our primary goal of this project was to use InfluxDB as a database for storing the time series data and visualize and analyze it later on Grafana. The process involved was, firstly to design the database schema for the InfluxDB instance and then use python to filter and parse the relevant metrics, and send them to InfluxDB database using InfluxDB python Client Library. After this was done, we had to migrate the archived data stored in Graphite whisper files and for this we created a tool to read and migrate data from whisper files to InfluxDB based on our new schema. We were ultimately able to migrate the monitoring solution from Graphite to InfluxDB, and now this solution

can be used to monitor EOS servers. Also, the schema is flexible so the same InfluxDB instance can be used in future to scale and add more number of metrics.

Acknowledgements

I would like to thank my Supervisor Herve Rousseau and the leader of the Disks Operations section at CERN, Massimo Lamanna for providing me the great opportunity to work on this project and for providing the desired resources whenever I asked for it. I am highly obliged for giving the time to learn and work on this project by myself, and for providing me the necessary guidance whenever I was facing some issue.

References

[1] AJ Peters, EA Sindrilaru, G. A. Eos as the present and future solution for data storage at cern. http://iopscience.iop.org/article/10.1088/17426596/664/4/042042/pdf.

[2] DB-Engines. System properties comparison graphite vs influxdb. https://db-engines.com/en/system/Graphite%3BInfluxDB.

[3] EOS. Eos introduction. http://information-technology.web.cern.ch/services/eos-service.

[4] Experiments, C. Experiments and facilities. https://home.cern/about/experiments.

[5] Grafana. Grafana website. https://grafana.com/.

[6] Graphite. Graphite website. https://graphiteapp.org/.

[7] InfluxDB. Influxdb-python. https://github.com/influxdata/influxdb-python.

[8] InfluxDB. Influxdb website. https://www.influxdata.com/.

[9] InfluxDB. Whisper-migrator. https://github.com/influxdata/whisper-migrator.

[10] Kapoor, A. Graphite whisper migrator to influxdb. https://gitlab.cern.ch/abkapoor/whisper-influx.

[11] Kapoor, A. Python influxdb client rpm package. https://gitlab.cern.ch/abkapoor/python-influxdb-rpm.

[12] Kapoor, A. Python script for parsing metrics. https://gitlab.cern.ch/abkapoor/eos-influx.