DEVELOPING AN EOS DATA TRANSFER MONITORING TOOL

AUGUST 2022

AUTHOR(S):
Srobona Ghosh

SUPERVISOR(S):
Cédric Caffy

PROJECT SPECIFICATION

The aim of this project is to create an EOS online traffic monitoring tool to extract the amount of files/bytes read/written during a specific amount of time, like a particular day, month or year, per protocol 'sec.app'=xxxx and per eos instance (like eoslhcb or eosalice) in the EOS report log.

ABSTRACT

The EOS (EOS Open Storage) is a distributed disk-based storage system that provides a service for storing large amounts of physics data and user files. Today, EOS manages more than 700 PB of raw storage, storing around 7 billion files and delivering more than 2.5 ExaBytes per year to CERN experiments (LHC and non-LHC). Experiments can store (write) and access (read) their data through several different protocols (XrootD, GridFTP, HTTP, directly mounted file-system (FUSE)). The history of all the transfers of a day is stored in a report file, where each row corresponds to a transfer. The project consists in analyzing, implementing, and putting in place an online monitoring tool that will allow the operators to get the transfer information they are interested in with one click and look. This will also help in automating actions based on the current system load and can significantly facilitate the operations of this service.

TABLE OF CONTENTS

CONCLUSION

REFERENCES

1. INTRODUCTION

EOS is a software solution that aims to provide fast and reliable multi-PB disk-only storage technology for both LHC and non-LHC use-cases at CERN. The core of the implementation is the XRootD framework which provides a feature-rich remote access protocol. The CERN storage system, EOS, was created for the extreme LHC computing requirements. EOS instances at CERN exceed seven billion files (as of June 2022), matching the LHC machine's and experiments' exceptional performances. EOS is now expanding for other data storage needs beyond high-energy physics, adopting it for their big-data systems.

This report contains the general overview of the project, what has existed so far, the technologies that were being used for this project, the problems that were solved, the solutions proposed, and the challenges that were faced and their workarounds.

The project is divided into three parts:

- Creating the entire project workflow starting from the eosreport file to building Grafana dashboards
- Replaying the history for all eosreports from 1st January 2022
- Deploying in production an automated task that will run the statistics every dayby an automatic process

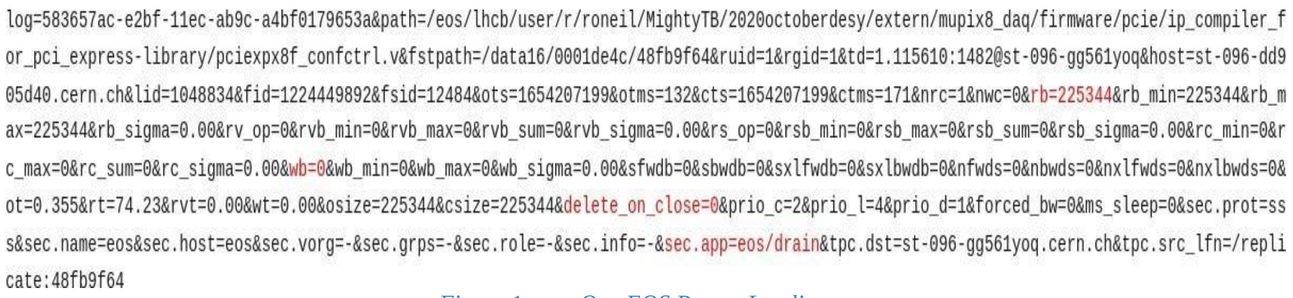
2. WHAT EXISTED SO FAR

a. EOS log reports:

EOS Log reports provide information about the files/ bytes read/ written/ created/ updated/deleted. They are used for data analysis, problem investigation, and security forensics. The EOS MGM writes report log files under /var/eos/report/<YEAR>/<MONTH>/<YEAR><MONTH><DAY>.eosreport.

Size has a directcorrelation with each EOS instance usage (10 to 30 GB/day)

Each line of the report logs looks like this:



```
log=583657ac-e2bf-11ec-ab9c-a4bf0179653a&path=/eos/lhcb/user/r/roneil/MightyTB/2020octoberdesy/extern/mupix8_daq/firmware/pcie/ip_compiler_f
or_pci_express-library/pciexpx8f_confctrl.v&fstpath=/data16/0001de4c/48fb9f64&ruid=1&rgid=1&td=1.115610:1482@st-096-gg561yoq&host=st-096-dd9
05d40.cern.ch&lid=1048834&fid=1224449892&fsid=12484&ots=1654207199&otms=132&cts=1654207199&ctms=171&nrc=1&nwc=0&rb=225344&rb_min=225344&rb_m
ax=225344&rb_sigma=0.00&rv_op=0&rvb_min=0&rvb_max=0&rvb_sum=0&rvb_sigma=0.00&rs_op=0&rsb_min=0&rsb_max=0&rsb_sum=0&rsb_sigma=0.00&rc_min=0&r
c_max=0&rc_sum=0&rc_sigma=0.00&wb=0&wb_min=0&wb_max=0&wb_sigma=0.00&sfwdb=0&sbwdb=0&sxlfwdb=0&sxlbwdb=0&nfwds=0&nbwds=0&nxlfwds=0&nxlbwds=0&
ot=0.355&rt=74.23&rvt=0.00&wt=0.00&osize=225344&csize=225344&delete_on_close=0&prio_c=2&prio_l=4&prio_d=1&forced_bw=0&ms_sleep=0&sec.prot=ss
s&sec.name=eos&sec.host=eos&sec.vorg=-&sec.grps=-&sec.role=-&sec.info=-&sec.app=eos/drain&tpc.dst=st-096-gg561yoq.cern.ch&tpc.src_lfn=/repli
cate:48fb9f64
```

Figure 1. One EOS Report Log line

b. Generation of the statistics

The eosreports need to be parsed into multiple key-value pairs separated by '&'. Some key names might vary, and values might be empty or even anonymized.

However, the eosreport generated by an MGM node for each eos instance does not contain the name of the eos instance. The statistics from the eosreport should contain the parsed and aggregated data from these eosreports grouped by time, eos instance, and protocol.

The statistics about the data transfers were computed manually with bash scripts, and the plots were generated with OpenOffice. This process was tedious and computationally time-consuming. A code snippet on how the filters in bash scripts were used to generate the statistics:

```
regex_read_bytes_0='&
rb=0&'
regex_read_bytes='&rb
=[0-9]*'
regex_write_bytes_0='
&wb=0&'
regex_write_bytes='&wb=
[0-9]*'

read -r TOTAL_READ_FILES TOTAL_READ_BYTES <<< "$(zgrep -Ev
'sec.app=(eos/drain|eos/balancing|deletion|rm|recycle|eos/replication)'
$eosreportfile | grep -v $regex_read_bytes_0 | grep -o $regex_read_bytes
| cut -d'=' -f2 | awk -f $awk_files_bytes_prog)"

read -r TOTAL_WRITE_FILES TOTAL_WRITE_BYTES <<< "$(zgrep -Ev
'sec.app=(eos/drain|eos/balancing|deletion|rm|recycle)' $eosreportfile |
grep -v
$regex_write_bytes_0 | grep -o $regex_write_bytes | cut -d'=' -f2 | awk -f
$awk_files_bytes_prog)"
```

c. Visualization of the data transfers

The pie charts obtained from the eosreport statistics look like:

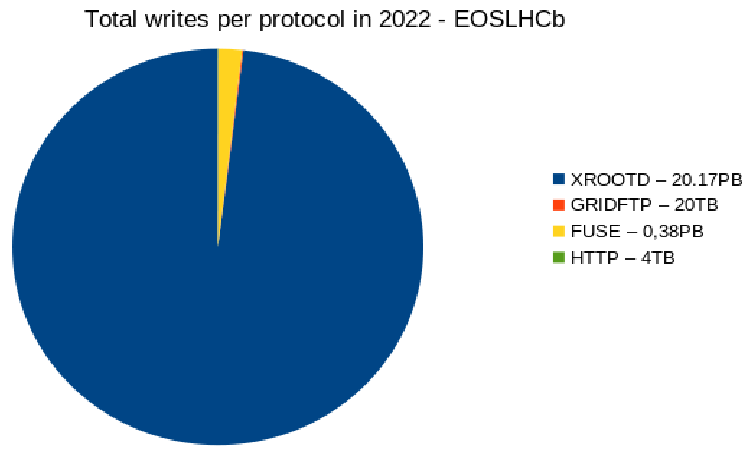


Figure 2. Total writes per protocol in 2022: EOSLHCb

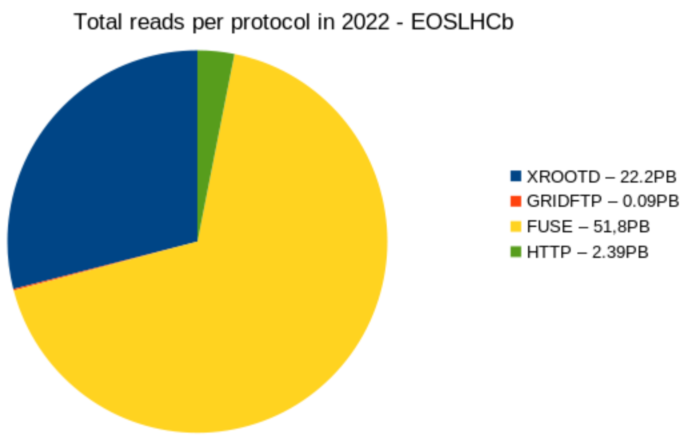


Figure 3. Total reads per protocol in 2022 EOSLHCb

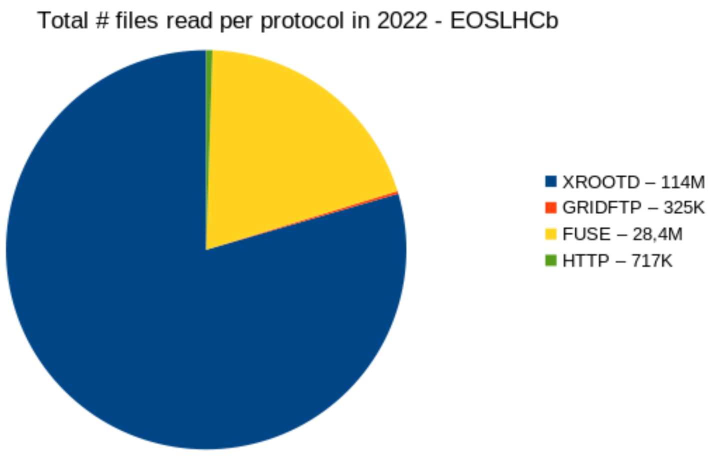


Figure 4. Total number of files read per protocol in 2022 EOSLHCb

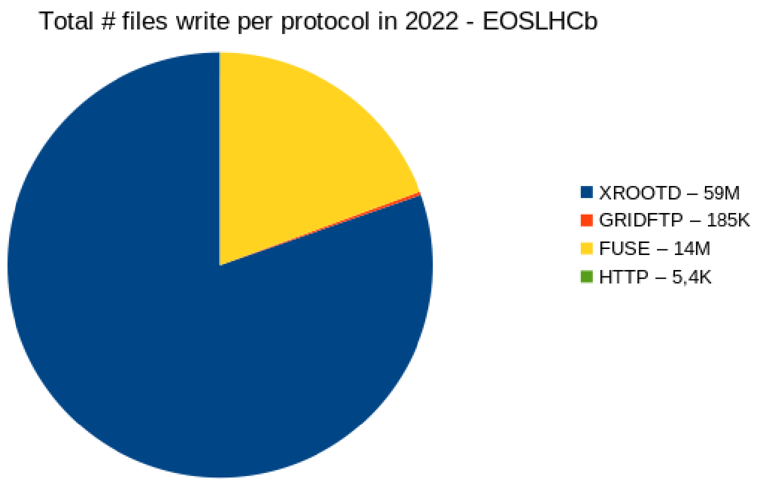
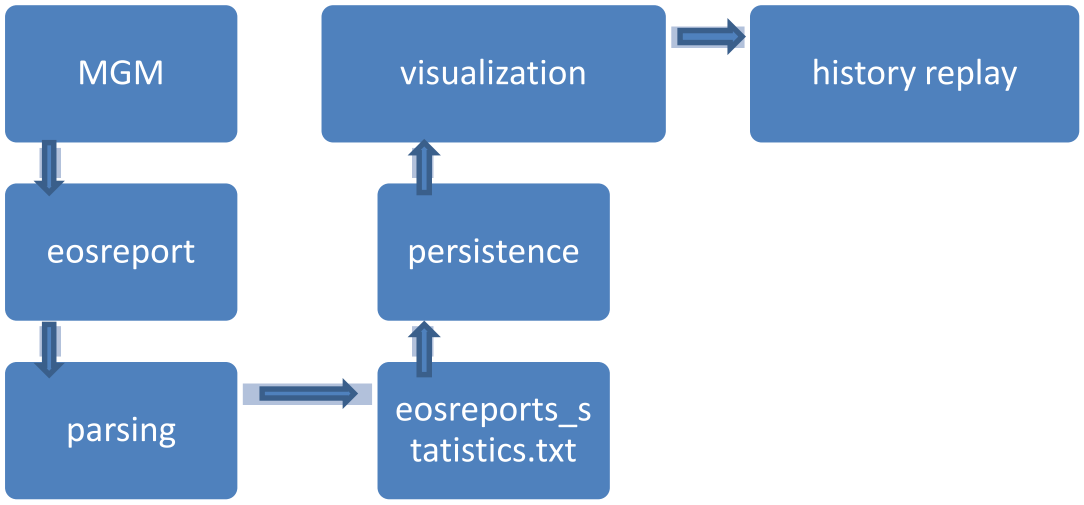


Figure 5. Total number of files written per protocol in 2022 EOSLHCb

3. GENERAL OVERVIEW OF THE PROJECT



4. EOS REPORT LOGS COLLECTING, PARSING, AGGREGATION, AND CREATION OF STATISTICS

a. Problem

How to read the EOS reports efficiently, how to generate the statistics (rb=x, wb=y, sec.app=zzzz, ...) and finally, aggregate them to have the amount of files/bytes read/written per protocol in one day/ month/ year without using the grep command as it was done previously?

b. Proposed Solution

The eosreport may contain some undesired data as a result of some EOS internal operations and FST deletions that are not required for the purpose of generating the statistics, storing the data, and visualizing them. So, it is necessary to filter out such records.

i. fluent-bit and stream processor

fluent-bit: fluent-bit is a fast and lightweight log processor, stream processor, and forwarder for Linux, OSX, Windows, and BSD family operating systems. Its focus on performance allows the collection of events from different sources and the shipping to multiple destinations without complexity. It is lightweight, allowing it to run on embedded systems as well as complex cloud-based virtual machines. It has in-built filters and pre-configured parsers and also allows backpressure handling and temporary buffer storage.

- Parser: Structure input into messages
- Filter: Enrich, modify, and store records
- Storage (Local Buffer): Increase reliability and performance

To filter, parse, aggregate and export the statistics from the eosreport file, fluent-bit was initially chosen. The parsing into key-value pairs was done by a Lua script while the aggregation of the data to extract the statistics was done by the stream processor. The output of this fluent-bit configuration file is in JSON.

fluent-bit-parsing: The fluent-bit configuration file applies all the necessary filters to drop the records from the eosreport file that are not desired.

Lua filter: The Lua script parses the log records into key-value pairs.

Stream Processor: The stream processor does aggregation(sum of files/bytes read/written per protocol) every 900s, output to the eosreports_statistics.txt text file.

However, a problem was encountered while using this approach. The parsed output from the fluent-bit was sent to the fluent-bit Stream Processor for aggregating which in turn was to be sent to the InfluxDB database for storing and running queries. The Stream Processor processes “streams” of data. It was configured so that it aggregates and output chunks of data received in a time interval of every 15 minutes of eosreport reading with the help of the “WINDOW tumbling” clause. Let us imagine that the Stream Processor starts its process at 23:55. As it will process and aggregate data every 15 minutes, the files that were read/written between 23:55 and 00:00 will be accounted for the next day. So, the data generated in this time interval for one particular day will have a closing timestamp of the next day resulting in incorrect data. Therefore fluent-bit and stream processor cannot be used to parse and aggregate the data coming from the eosreport file.

c. Solution used

This problem was tackled by the use of a combination of a python script and a fluent-bit process. To parse, aggregate and output the aggregated data, the Python script is used. Fluent-bit is used to add the eos instance to the aggregated data and to push these data to influxDB.

i. Python Script and fluent-bit

Python script: The Python script that reads the eosreport file, applies the necessary filters, do the parsing into key-value pairs, aggregates the data by performing sums and output the aggregated data to the eosreports_statistics.txt file.

```
summary = kpi_summary.get(record['protocol'], {"cts_timestamp": 0, "total_read_bytes": 0,
"total_vector_read_bytes": 0, "total_write_bytes": 0, "protocol": record['protocol'],
"total_read_files": 0, "total_write_files": 0})

    summary["cts_timestamp"]=max(summary['cts_timestamp'],record['cts'])

    summary["total_read_files"] += int(record['rb'] != 0)

    summary["total_write_files"] += int(record['wb'] != 0)

    summary["total_read_bytes"] += record['rb']

    summary["total_write_bytes"] += record['wb']

    summary["total_vector_read_bytes"] += record['rvb_sum']

    kpi_summary[record["protocol"]] = summary
```

eosreports_statistics.txt: The output file to the Python script or the fluent-bit parsing configuration file which contains the data in JSON format. This file will then be read by a fluent-bit process (fluent-bit-data-sending) in order to add the eos instance name to the data and store it to influxDB. A few lines from this output file for the lhcb eos instance look like:

```
{"cts_timestamp": 1641077999, "total_read_bytes": 190859441198562,
"total_vector_read_bytes": 28565352533548, "total_write_bytes": 156806252658577,
"protocol": "None", "total_read_files": 368062, "total_write_files": 522513}
{"cts_timestamp": 1641077987, "total_read_bytes": 466858363207063,
"total_vector_read_bytes": 0, "total_write_bytes": 5013016311118, "protocol": "fuse",
"total_read_files": 64085, "total_write_files": 67288} {"cts_timestamp": 1641077996,
"total_read_bytes": 147065116057895, "total_vector_read_bytes": 0, "total_write_bytes":
126926809804, "protocol": "fuse::lxplus", "total_read_files": 15931, "total_write_files":
582} {"cts_timestamp": 1641077988, "total_read_bytes": 1732189187516,
"total_vector_read_bytes": 0, "total_write_bytes": 189548812061, "protocol": "eos/gridftp",
"total_read_files": 3993, "total_write_files": 1591}
```

fluent-bit-data-sending: The fluent-bit-data-sending fluent-bit process is used to add the EOS instance to the data coming from the eosreports_statistics.txt file and send that data to the buffer influxDB database for persistence.

The python script exactly replicates the data processing job done by the earlier fluent-bit configuration file. It applies all the filters, parses the records into key-value pairs, and lastly, performs the aggregation to generate an summary of the required metrics in each eosreport for a whole day and for each protocol instead of doing that on streams of data received at a time interval of 15 minutes. Thus, after the execution of the python script, we get the eosreport statistics of a full day's report instead of the statistics being appended to the output file every 900 seconds as it was happening earlier with the fluent-bit configuration file. The records of the output file of the python script are JSON objects.

Along with the python script, a fluent-bit process is used whose primary job is to add the "eos_instance" tag to the output of the Python script since the eosreport produced by the MGM node for each instance does not contain the instance name.

5. PERSISTENCE OF THE STATISTICS: STORING THE AGGREGATED DATA IN THE DATABASE

a. Problem

How can the statistics and the aggregated data be stored in the best possible way?Should a database be used? What types of databases can be used?

b. Solution

InfluxDB was chosen because it is purpose-built for time series data. Relational databases *can* handle time series data but are not optimized for typical time series workloads.

InfluxDB is designed to store large volumes of time series data and quickly perform real-time analysis on that data. In InfluxDB, a timestamp identifies a single point in any given dataseries. This is like an SQL database table where the primary key is pre-set by the system and is always time. Two InfluxDB databases in the same instance were used in this project.

i. influxDB-datatransfer-buffer

This database contains every row coming from the fluent-bit-data-sending process. It can be considered as a "buffer database" from which the influxDB-datatransfer-prod database will be filled using the query SELECT … INTO (for backlog ingestion) and a continuous query running on it (for real-time data ingestion). Continuous Queries(CQ) are InfluxQL(SQL used in InfluxDB) queries that run automatically and periodically on time-series data and store the query results in a specified measurement.

The InfluxQL query to aggregate and send the data from the influxDB-datatransfer-buffer database to the influxDB-datatransfer-prod database:

```
SELECT sum_total_read_bytes AS strb, sum_total_read_files AS strf,

sum_total_vector_read_bytes AS stvrb, sum_total_write_bytes AS stwb, sum_total_write_files

AS stwf INTO “influxDB-datatransfer-prod”.”autogen”.”aggregated_data” FROM (SELECT sum(*)

FROM processed_input_file GROUP BY time(1d) fill(none) tz('Europe/Paris')) GROUP BY

protocol, eos_instance, *
```

Four rows of data in the influxDB-datatransfer-buffer database for each of the physics eos instances look like:

```
1659650396000000001 3621 cms http 116063063159898 92197 0 579451580825 2172

1659650398000000000 1585 alice None 51249050111160 5374245 838046180977115 58229348340

17295

1659650398000000000 2314 lhcb None 280108808908843 1259303 122416214749645 245388196277844

605277

1659650398000000000 3198 atlas http 170422307820007 170671 0 54732 2373
```

ii. influxDB-datatransfer-prod

This is the final production database containing thetotal amount of files/bytes read/written by an eos instance and by protocol each day. This database will be queried by Grafana to display the plots. A few rows of the grouped and aggregated data stored in the influxDB-datatransfer-prod database for the lhcb eos instance look like:

```
1659391200000000000 lhcb fuse::swan 3896284139 14 0 0 0

1659391200000000000 lhcb fuse::volhcb 0 0 0 14399435 172

1659391200000000000 lhcb http 22533040271 87 0 139226453583 292

1659391200000000000 lhcb tpc 4715014533 8 0 71558977600 12
```

c. Disadvantages

However, InfluxDB cannot handle duplicated data for different sequence tags for values that are incremented for consecutive, simultaneous events. Also, running continuous queries on an InfluxDB database during backlog ingestion of data requires the use of the 'FOR' and 'RESAMPLE' clauses (which tell us how far to look back in the past to get the data for a particular time period) which can result in duplicate data in the database for the same timestamp and tag set. This may be a problem in production.

6. VISUALIZATION OF THE STATISTICS

a. Problem

How can the data be visualized in a nice monitoring setup? What should the data visualization look like? Which visualization tool is to be preferred: Grafana or Kibana?

b. Solution

Grafana is used to accomplish this task as it is vastly used as a visualization tool for monitoring tasks at CERN.

c. Grafana

Grafana dashboards are created that will allow us to see the amount of files/bytes read/written, per eos instance, per day, and per protocol. We chose to do the visualization of the data transfers using pie charts and line graphs.

Grafana is an open-source analytics and interactive visualization tool. It provides charts, graphs, and alerts for the web when connected to supported data sources like InfluxDB, Elastic Search, Graphite, Prometheus. Thus, using Grafana makes it very convenient for use as we are using InfluxDB for the persistence of the data. Grafana also allows users to create complex monitoring dashboards using interactive query builders.

7. REPLAY OF HISTORICAL DATA FROM THE 1ST OF JANUARY 2022

The most important part of the project was to generate the statistics from all the eosreports from the beginning of the year 2022, that is, the backlog ingestion into the database and generating the dashboards for all the data from the beginning of 2022 until the current date. The bash script was used to automate this task of replaying the historical data. Instead of manually providing the eosreport file of each day of each MGM of each instance to the python script, this bash script will automate this process. The script takes an eos instance, a year, a month, and a day as inputs. If only a year is provided, the entire year will be replayed. If a month in addition to a year is provided, the data for the entire month of that year will be replayed. If a day is provided in addition to the month and the year, the specific day will be replayed.

Bash script: The bash script copies the zipped eosreport file of each day of each MGM of each instance from the directory */eosproject/project/e/eos/Report/{instance}/{MGM}/{YEAR}/{MONTH}*, copies it to the */var/tmp/{instance}* directory, extracts it (tar -xvf ./eosreport.XXXX.gz), provides it to the python script present in the same directory to execute over it in order to do the calculation and append it to the text file that is read by fluent-bit. Once the Python script execution has finished, the bash script deletes the copied unzipped eosreport file that the Python read.

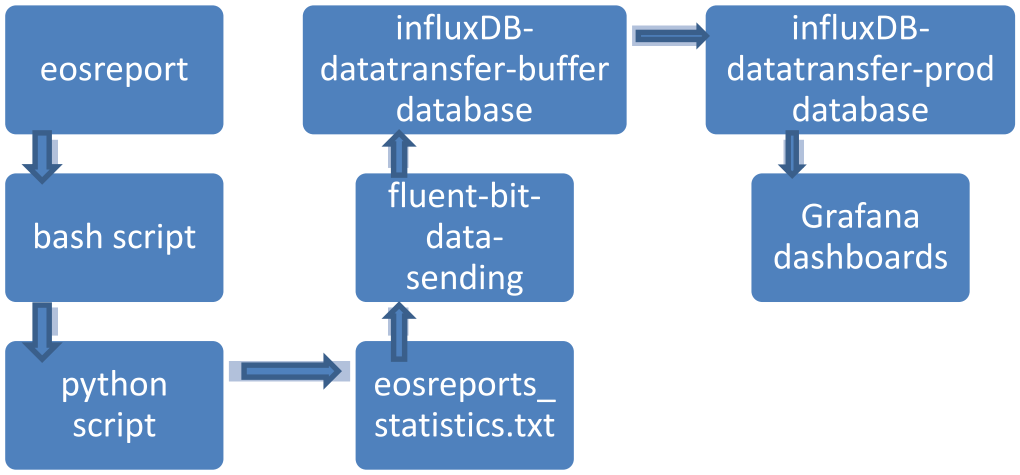
```
cp -v $file /var/tmp/eos-datatransfer-history-replay/"${INSTANCE}"/

gzip -dv /var/tmp/eos-datatransfer-history-replay/"${INSTANCE}"/${filename}

time python3 py_script.py --inputfile /var/tmp/eos-datatransfer-history-
replay/"${INSTANCE}"/${filename%.gz} --outputfile /var/tmp/eos-datatransfer-history-
replay/"${INSTANCE}"/eosreports_statistics.txt

rm -v /var/tmp/eos-datatransfer-history-replay/"${INSTANCE}"/${filename%.gz}
```

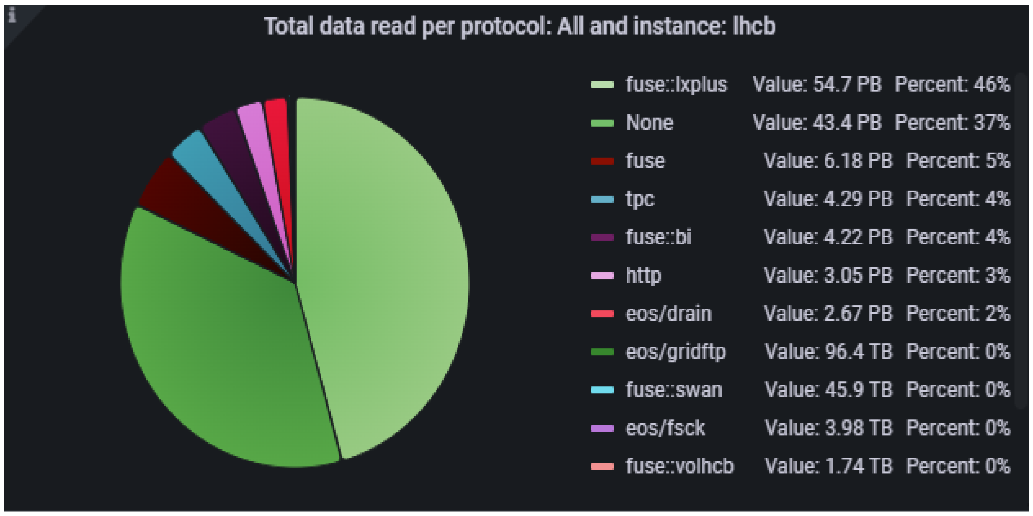
8. SUMMARY


eosreport
bash script
python script
eosreports_statistics.txt
fluent-bit-data-sending
influxDB-datatransfer-buffer database
influxDB-datatransfer-prod database
Grafana dashboards

9. RESULTS

This project's files can be found in this GitLab link: https://gitlab.cern.ch/eos/eos-datatransfer-monitoring.

The entire workflow from an eosreport file to building the Grafana dashboards was implemented and the replay of the historical data from the beginning of 2022 was successfully done. As desired, the bash script collects each eosreport file and calls the Python script to generate the eosreports_statistics.txt file containing all the aggregated data. Next, a fluent-bit process adds the eos instance name and sends all the data to the buffer influxDB database on which queries are ran to store long-term production data into the production influxDB database. Grafana generates the dashboards allowing the users to select the EOS instance and the protocol they are interested in one click and look.
The pie-charts for each eos instance: lhcb and all protocols from the beginning of 2022 till the current date are provided below.



fuse::lxplus	Value: 54.7 PB	Percent: 46%
None	Value: 43.4 PB	Percent: 37%
fuse	Value: 6.18 PB	Percent: 5%
tpc	Value: 4.29 PB	Percent: 4%
fuse::bi	Value: 4.22 PB	Percent: 4%
http	Value: 3.05 PB	Percent: 3%
eos/drain	Value: 2.67 PB	Percent: 2%
eos/gridftp	Value: 96.4 TB	Percent: 0%
fuse::swan	Value: 45.9 TB	Percent: 0%
eos/fsck	Value: 3.98 TB	Percent: 0%
fuse::volhcb	Value: 1.74 TB	Percent: 0%

Figure 6. Total data read per protocol

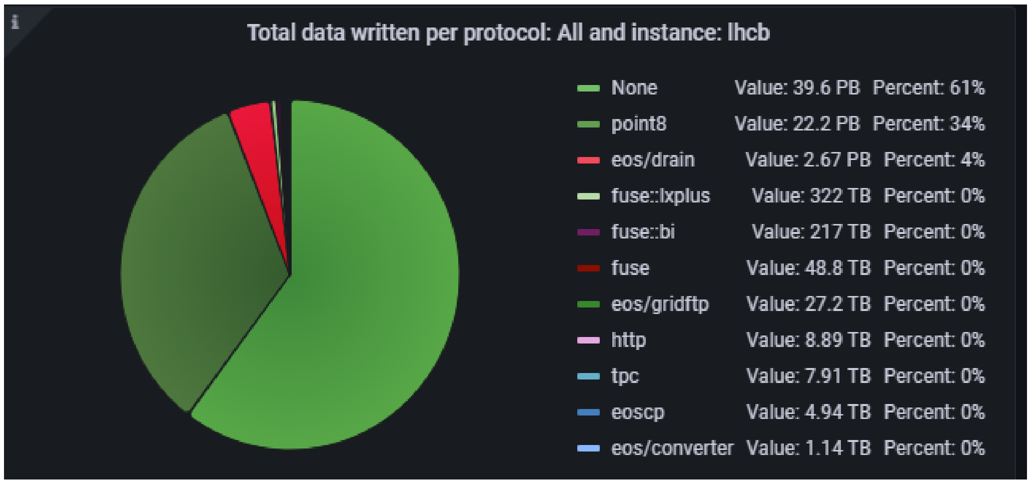


Figure 7. Total data written per protocol

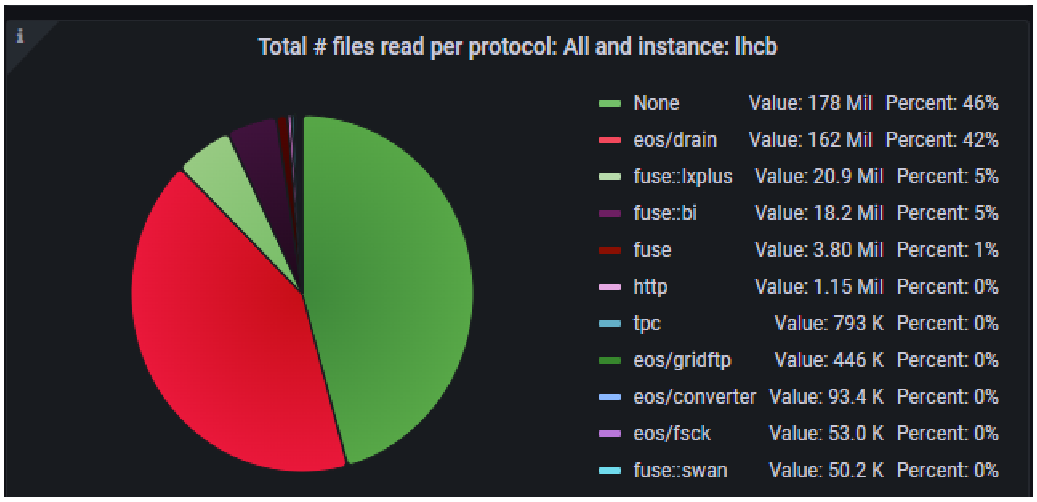


Figure 8. Total number of files read per protocol

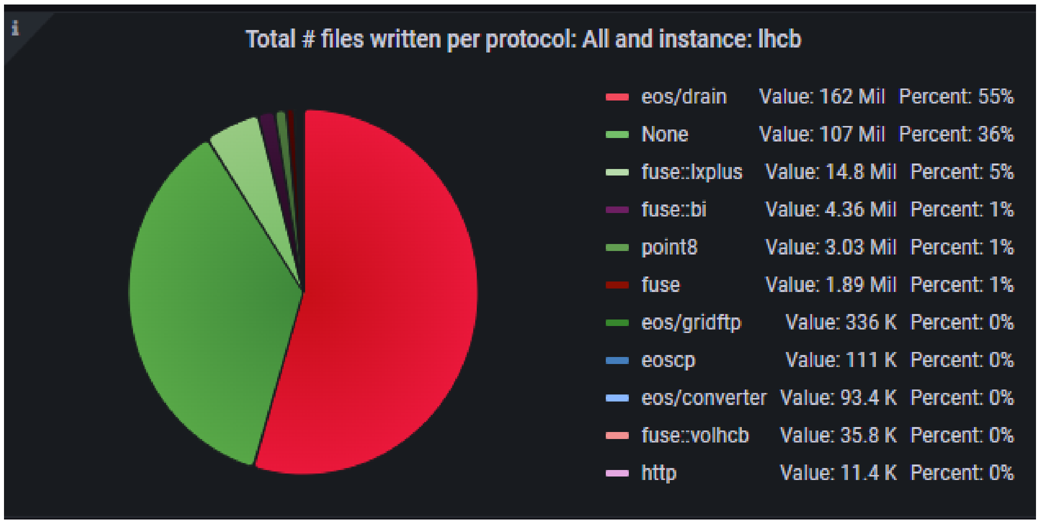


Figure 9. Total number of files written per protocol

10. FURTHER INVESTIGATION

While most of the plots verify the statistics about the number and amount of transfers, a discrepancy was observed while inspecting the time-series plot of the total amount of total data reads (for the sum of data reads and vector data reads), where vast amounts of data in order of exabytes(EB) were encountered. This was also reflected in the Grafana dashboard, where it can be seen in the form of the huge spikes in the amount of data reads between 30th June 2022 and 30th July 2022. The plot of which is provided below:

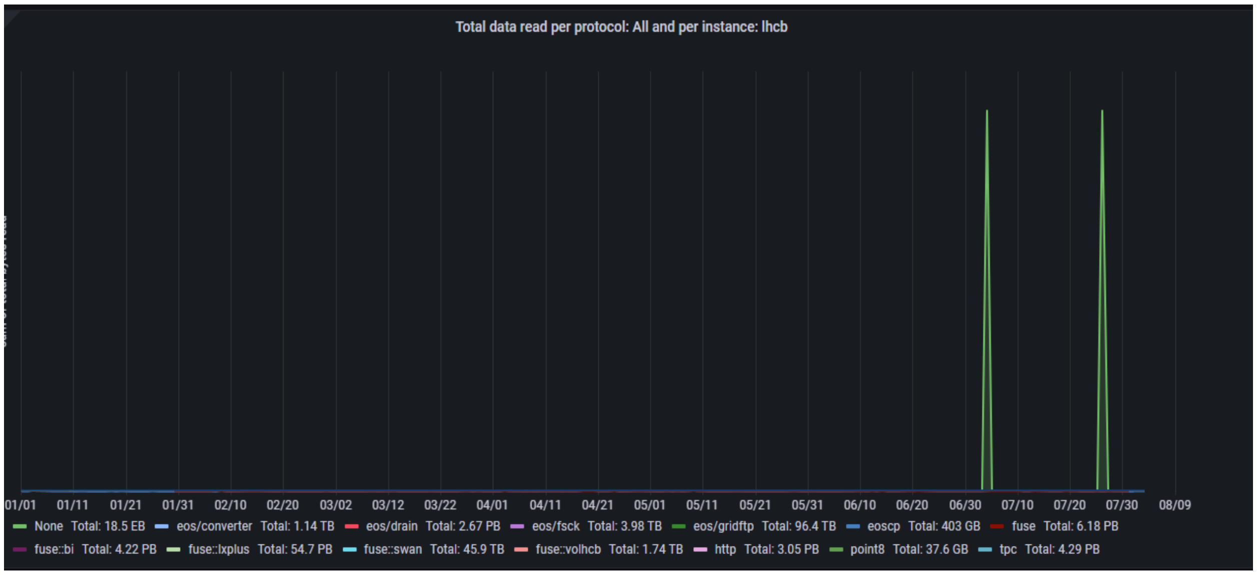


Figure 10. Discrepancy in Total data read per protocol for July 2022

Thus, further investigation is needed to determine the reason that results in these two huge spikes in the amount of data reads for July 2022.

11. CONCLUSION

The aim of that project was to create the EOS Data Transfer Monitoring tool that will allow the EOS operators to see, in one click and look, the amount of files/bytes read/written in an EOS instance within a specific period of time.
Python, Bash, fluent-bit, InfluxDB and Grafana were used to accomplish this task. This monitoring setup can be automated and deployed in production using a cron job to obtain the required statistics of the current day eosreports.
Finally, this monitoring tool can be extended to an RPM to deploy all the related scripts and configuration files in production on an EOS MGM using the Puppet configuration management tool.

12. REFERENCES

- https://indico.cern.ch/event/1040723/contributions/4371444/attachments/2259661/3839856/FluentBit-ASFD-10-06-2021.pdf
- https://en.wikipedia.org/wiki/Grafana
- https://docs.fluentbit.io/manual/
- https://docs.influxdata.com/influxdb/v2.0/
- https://grafana.com/docs/
- https://monit-grafana.cern.ch/d/000000036/eos-control-tower?orgId=22&refresh=5m
- https://home.cern/science/computing/storage
- https://github.com/cern-eos/eos