SEPTEMBER 2021

EOS Storage Resource Monitor

AUTHOR(S):
Anna Pacanowska

CERN IT-ST-PDS Group

SUPERVISOR(S):
Jaroslav Guenther

PROJECT SPECIFICATION

EOS [1] is a distributed filesystem used at CERN experiments. The storage administrators often have to manage the resources for thousands of users, who store vast amounts of information in EOS – hundreds of petabytes of data. It is therefore imperative that limits are set on the resource consumption for specific users, groups, projects and data paths. In order to manage these limits, this project aims to develop a monitor with useful visualisations of different aspects of resource usage. The monitor shall be based on technologies having mid to long term support and requiring minimal maintenance from the administrators. The monitor shall show the resource usage, limits, and the evolution of these in time. This will also help the user community to manage the allocated resources and to spot potential operational issues faster.

ABSTRACT

This report describes how the EOS Storage Resource Monitor was created. It explains which tools were used and the reasoning behind these choices. It describes the deployment model and the visualisations that are displayed. For illustration, example pictures of the monitoring dashboard are included. In order to achieve resilience of the service to the potential unforeseen changes in licenses and support of the applied techologies at CERN, two independent solutions were initially adopted (Grafana + MetricTank, Kibana + ElasticSearch). Eventually, because of more robust access control, only the Kibana dashboards were exposed to the end users. Access to the monitoring data was given only to the authorized administrative personnel.

TABLE OF CONTENTS

TIMELINES

1. INTRODUCTION

EOS is a distributed filesystem used at CERN experiments. The aim of the project was to create a tool for monitoring EOS storage resources. The monitor periodically gathers data and visualises it in a dashboard. EOS allows the adminstrators to set limits on space and file count for users, groups and projects on a given path. These limits are called quotas and the paths for which they are set - quota nodes (Figure 1).

The main result of the project is a monitoring dashboard with various visualisations of the quotas and their usage. It allows the managers and administrators to easily view how many resources are used and how many are free. They are also able to see the evolution of these metrics through time. That way they are able to quickly spot important issues, such as exceeded quotas or sudden increases in the used resources.

Code repository for this project [2] and the documentation [3] can be found on gitlab.

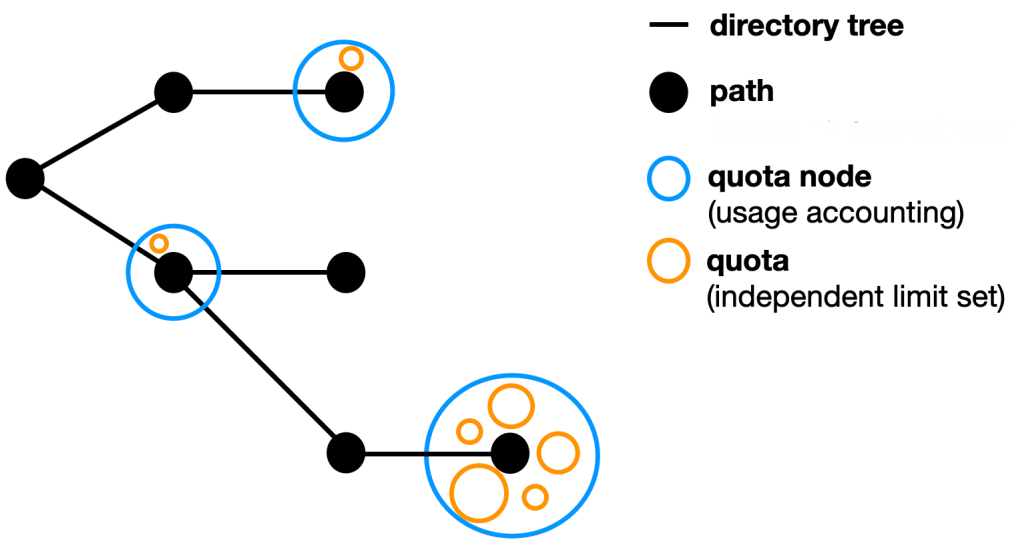

Figure 1. Diagram with an example role.

2. FRAMEWORK CONSIDERATIONS

The first step was to discuss with the team which technologies should be used. As the monitor should run for the next many years with minimal maintenance, the frameworks had to be supported by CERN IT in the mid to long term. They should also offer many possible visualisations to fit all of our use cases. Additionally, they had to allow access control, since we are displaying some sensitive data.

a. FRONTEND

The tools that best fitted our requirements were Grafana [4] and Kibana [5]. These are visualisation tools often used for monitoring at CERN, actively supported by CERN IT. They fit the requirement of easily customisable dashboard (offer many different plots and graphs) to fit the use cases of different experiments. They also provide access control.

Storage group (CERN IT-ST), which our team is a part of, runs an independent, custom Grafana instance, which gives us full control over its management. IT-ST Grafana [6] was chosen to provide the EOS monitoring service for the experiements. There is also a Kibana instance being supported by the monitoring group (CERN IT-CM). This tool was at first supposed to be a backup, but eventually became the primary solution. CERN IT is currently moving towards Open Distro for ElasticSearch [7] and there is a good belief that Kibana will stay supported in the years to follow.

An alternative was using a Python framework such as Flask or Django, but this solution was not implemented. Compared to the frontend frameworks mentioned above, these solutions would require more work to implement. Their flexibility would stay limited to the code interventions followed by the necessary deployment. The dashboards would be prone to stay static with little possibility for customisation. Adding more advanced features like timelines, advanced automated access authorization etc. might not be as straightforward either. Additionally, these frameworks are not actively used within the storage team. This might be a problem in the future, as making any major modifications would require someone with an active experience with these techonologies.

b. BACKEND

The frameworks that were chosen are MetricTank [8] and ElasticSearch [9]. Graphite [10] is a simple tool for storing and querying metrics. It has less features and settings than ElasticSearch. It is therefore easier to set up and maintain, but offers less flexible queries. MetricTank is an improved version of Graphite. It retains its minimalistic design, but offers faster performance. Similarly to Grafana, MetricTank instance is fully supported and maintained by the IT-ST team.

ElasticSearch on the other hand is a more elaborate solution which gives us more analytic features. It also supports advanced access control – it is even possible to have unique user permissions per data entry ('document') on only one dashboard. The main disadavantage of this solution are the larger maintenance requirements – currently provided by the CERN IT team. In the recent versions, the ElasticSearch licence has changed. Because of that the IT team is moving towards OpenSearch [11] – a project based on the previous ElasticSearch version.

One of the other possibilites that were considered was simply storing the data as files on EOS. It was decided against it, as the files might get large with time and take long time to query.

c. CONCLUSIONS

Finally, after several framework combinations were considered, two were selected for the implementation of our project: MetrictTank + Grafana and ElasticSearch + Kibana.

The MetricTank backend instance used by IT-ST does not have very strong protection in place and anyone from CERN can query it. In Grafana, anyone with viewer privileges can view all the data of the backend by default, but it is possible to use Grafana Teams in order to restrict the access to the dashboards. If we were to choose such solution, every dashboard already existing on the IT-ST Grafana instance would need to implement such protection. In order for the administrators to see only the data for the instance they manage, there would need be a separate dashboard created for each instance and assigned to a Grafana Team. In addition, the Grafana Teams are not compatible with the CERN egroup access control. Memebership updates would need to happen per person and be managed by the IT-ST team. We could achieve acceptable level of data protection only by creating dedicated MetricTank and Grafana instances per experiment, but the operational overhead would be too large for this solution to fit our minimal maintanence requirement.

On the other hand, in Kibana new users will not see any data unless they are explicitly allowed to. The ElasticSearch backend is password protected. There are two mechanisms of access control in Kibana.

Tenants [12] can allow different users to access different dashboards, visualisations etc. Roles [13] can allow the users to see only some of the data. They can be mapped to an egroup (Figure 2). We decided the roles were more suitable for our use case. We have one set of dashboards (one showing current status and one showing timelines) instead of many copies. For each instance we created a role that allows the users to see only the metrics for this instance and is mapped it to an egroup. The egroups were defined in the groups portal [14]. Each of them allows its members to access the information for only one instance. Therefore the users see different data on the same dashboard, depending on their egroup membership.

Taking all these considerations into account, we decided to expose Kibana + ElasticSearch as the final solution for the quota managers.

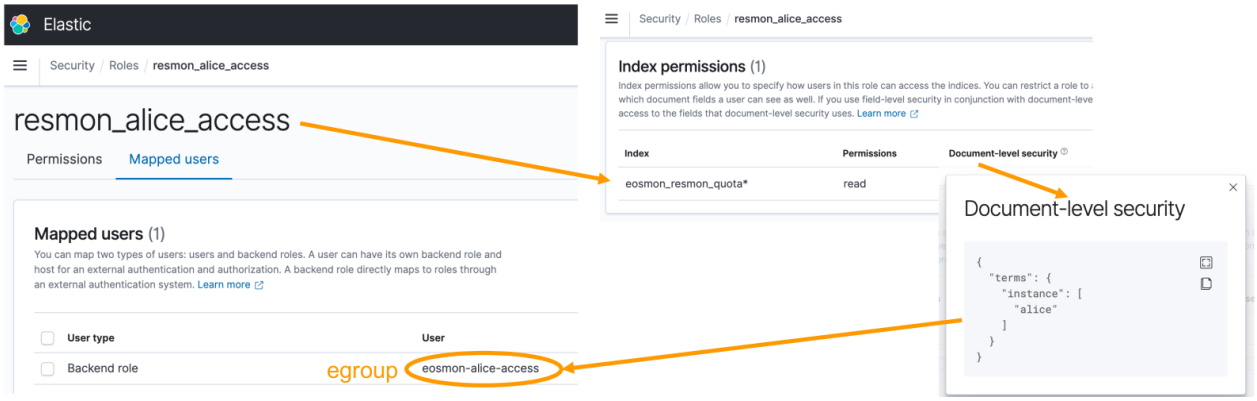

Figure 2. Diagram with an example role.

3. DEPLOYMENT MODEL

A Python script collects the data about the quota nodes using `eos quota ls -m` command. Then it transforms the output into the appropriate format for MetricTank and ElasticSearch. Finally, the metrics are pushed to the `metrictank-carbon.cern.ch` and `es-eosmon.cern.ch` instances. The metrics are collected every day from the EOS headnodes. For this purpose, we have configured CERN IT-ST puppet management tool to ensure a cron job is set up for every EOS instance headnode.

The script queries the backend after pushing the data in order to check if the metrics were sent properly. In case this validation fails or some other error arises during the execution, the administrators of the service will be informed by email. The script saves the logs to the file on the headnode, so it is possible to find out where it failed and for what reason.

Kibana basic administrative page can be found in [15].

4. METRICTANK + GRAFANA

a. DATA FORMAT

MetricTank is used to store metrics - numerical values which can change over time. Each metric has a name, which consists of a path and optionally some tags. Path is a sequence of words, called components, separated with dots. The tags are pairs of key and value. Unlike path, they are not

hierarchical. Tag values can include some special characters, such as '/' and '.', which are not allowed in the path components.

Our metrics have a path in the following format: `eos.<instance>.resmon.<metric_type>`. Prefix `eos.<instance>` is often used for metrics related to the EOS filesystem. Namespace `resmon` differentiates our metrics from other metrics starting with the same prefix. Next part designates the type of the metric. It can be one of `usedbytes`, `maxbytes`, `usedlogicalbytes`, `maxlogicalbytes`, `usedfiles`, `maxfiles`, `statusbytes`, `statusfiles` and `percentageusedbytes`. The metric value can be only numeric, which is why we need to encode all other information in the name. In order to describe the quota node we use tags: path to the node, id type (user/group/project), user or group id and whether the quota is set (for space and for files). This is how an example metric name looks like: `eos.pilot.resmon.percentageusedbytes;id=apacanow;id_type=user;path=/eos/pilot/opstest/eosquotamon/dir/;bytes_quota=yes;files_quota=yes`

b. VISUALISATIONS

The dashboard is divided in two parts - one showing the current status of the quota nodes and the other one showing the changes on the timeline. It is possible to filter the graphs by `path`, `id_type` and `id`. The user can also select a time window. Graphs we show fall into the following categories:

i. OVERVIEW OF THE INSTANCE

On the top of the dashboard, the user can see general information about the instance - total quota on the space (Figure 3) or the files and its usage.

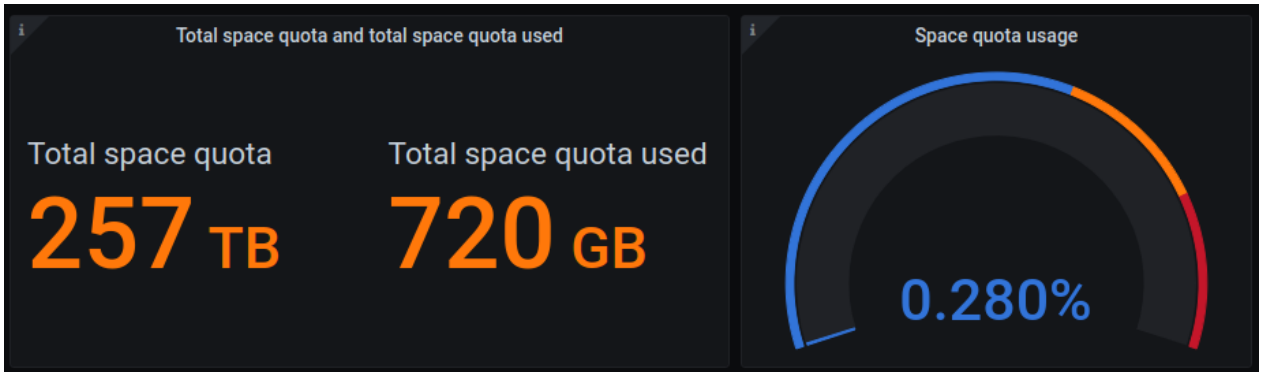

Figure 3. Panel showing space usage and limit for the whole instance.

ii. CURRENT QUOTA STATUS

Here, the visualisations display only the users, groups and projects for which the limit (quota) is set.

The first graph shows the usage of the quotas in percentange, sorted in the descending order. This would immediately alert the managers that some quotas were close to being filled. The next graphs show the details - number of used, free and maximum bytes for each quota (Figure 4). Then a similar set of graphs is presented for the file quotas.

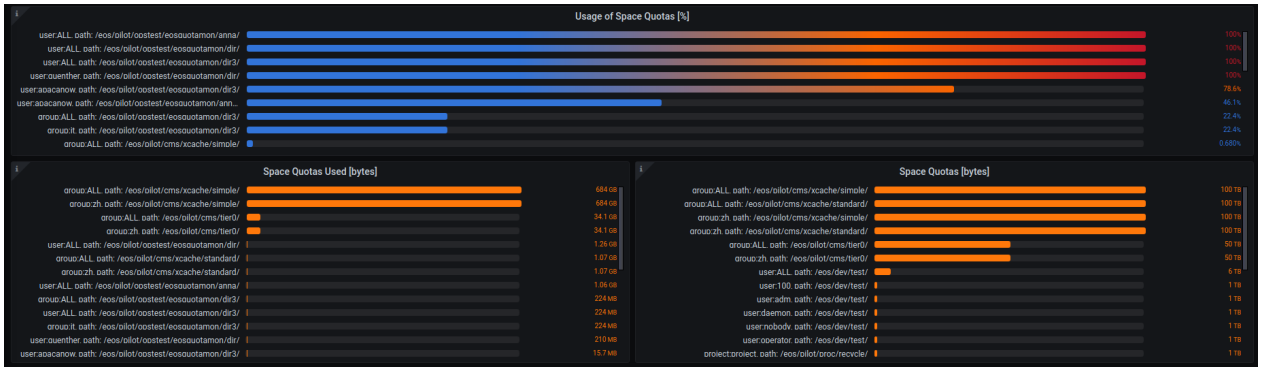

Figure 4. Panels showing space usage and space limit per quota.

iii. QUOTA CHANGES IN TIME

In this category the same statistics are presented as in the previous one, but on the timeline (Figure 5). This would allow the administrators to see when some problem started and to quickly spot operational issues, such as sudden increases in used resources.

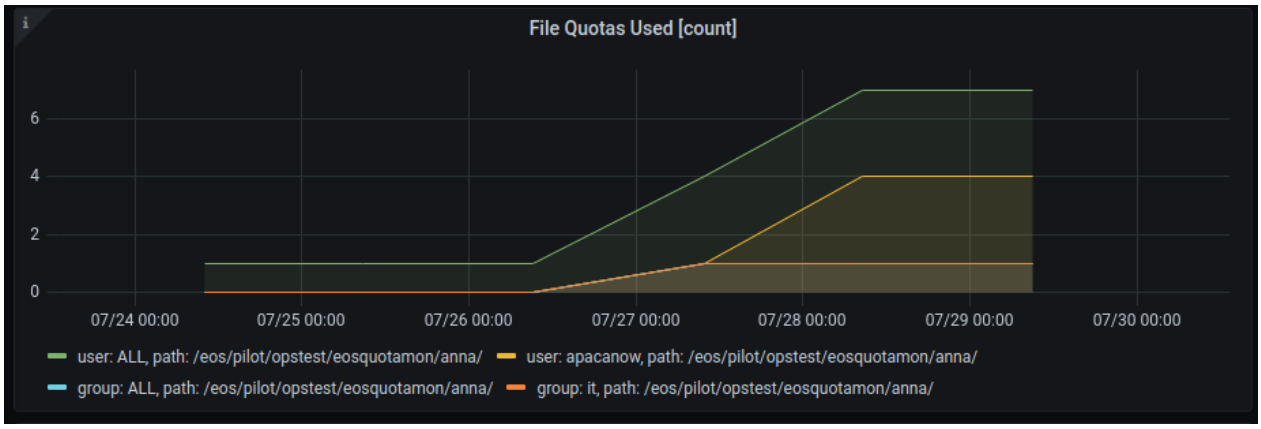

Figure 5. Panel showing the changes of the number of used files.

iv. RESOURCE USAGE BY ALL USERS, GROUPS AND PROJECTS

This part shows the statistics for all users, groups and projects - even the ones with no specific limit set. First visualisation shows users and groups that use the most space/files (Figure 6). The same metrics are presented on the timeline. Finally, there are graphs showing the changes through time of the used space and files for each quota node, user, group and project.

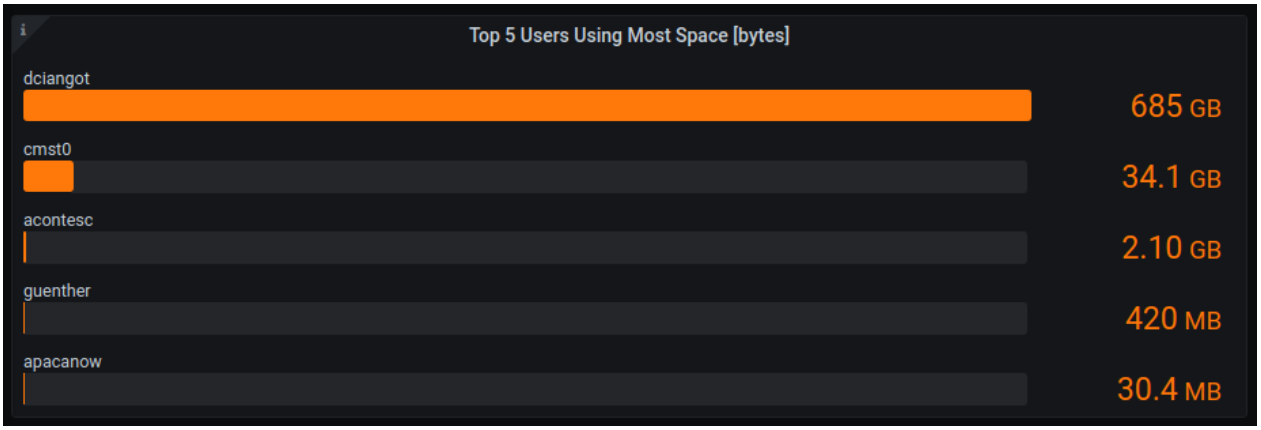

Figure 6. Panel showing the the users that use the most space.

5. ELASTICSEARCH + KIBANA

a. DATA FORMAT

A data entry in ElasticSearch is called a document. The documents are stored in an index. In our case, two different indices are used to store two different families of documents.

The first type of a document represents the quotas. Each entry contains the timestamp, the quota specification (path, instance, uid/gid), the metrics (used space, limit on the file count, status etc.), the instance metrics (instance logical capacity, space factor etc.) and the formatted fields (quota id, name).

On the dashboard there is a visualisation that displays the metrics about the entire instance, such as the nominal capacity. One way to achieve that could be to create a special document with this information. But then the user could add a filter that would not match this special document and the visualisation would show no data. A possible solution was to ignore the global filters, but then it would not be possible to filter by the instance, which is very useful. This data is therefore stored in every document in the instance. That way we can always display such general metrics if there is at least one document selected.

We also use some formatted fields that don't add any new information, but are necessary for some visualisations. For example, we often want to aggregate the documents for the individual quotas. To identify the quota we need `path`, `id_type` and `id`. An obvious solution would be to group by these fields in the visualisation. However, grouping by multiple fields is not supported in Kibana visualisations. It is possible to create buckets from one field and within each bucket group by another field, but for our use case a flat structure was required. To overcome this obstacle we added `quota_id` field composed of the path, type (user/group/project) and id, e.g. `/eos/pilot/opstest/eosquotamon/, daemon [u]`. This serves as the unique identifier of the quota and allows easy aggregating.

The second type of a document is used for the treemap visualisation (example treemap [16]). It represents the tree structure of the quota nodes. Each entry contains the timestamp, the quota node specification (path, instance etc.), the metrics (used space, space limit), the tree structure (parent path, leaf, etc.), the formatted fields (directory name, used space string etc.) and the fields for multi-index search (id, status bytes etc.).

The metrics are calculated per quota node by recursively summing all its children (i.e. all quotas on all quota nodes down the quota node path). The formatted fields are necessary for the treemap visualisation. For example `dirname` is a directory name without the rest of the path, which is displayed as the quota

name. The `usedspacestring` field contains human readable value of the used space, as this format is not supported in this visualisation.

The last category are the fields for the multi-index search. The documents in different indices contain different fields. When the user filters for a field that exists in one index, but not the other, all documents from the second index will be filtered out. This should be avoided, because the treemap should be visible even if there is a filter on the quota documents. A few solutions were tried. The first was to simply ignore the filters, but they were essential to navigate the tree. The second idea was to use `courier:ignoreFilterIfFieldNotInIndex` setting, but after turning it on most custom filters did not work anymore. This is because of a bug in Kibana - it has not been fixed in the currently used version. We could also put all documents in one index, but we would be facing the same problems with no additional benefits. Eventually the most often used fields from the quota index, such as `id` or `statusbytes`, were added to the tree document with all possible values from the instance.

b. VISUALISATIONS

Clicking one of the visualisations will add a filter that matches only clicked quota. The user can also create a custom filter using a query language, such as KQL or Lucene. The panels below will then be adjusted to count only the quotas that match the filters.

Visualisations presented in Kibana fall into similar categories as the ones from Grafana.

i. OVERVIEW OF THE INSTANCE

On the top the user can see the most important information about the instance (Figure 7). First, there is a short description of the dashboard with a link to the more extensive documentation. Next to it there is the date of the last update and the information about the instance capacity and used space.

Below, there is a treemap visualisation. The size of each rectangle corresponds with the size of the quota. The ‘Toggle scale’ button changes the differences between the sizes – it is useful when some rectangles are too small to display the name. The visualisation allows the user to go down the tree by clicking one of the rectangles or up the tree by using the ‘Back’ button. When the user goes to some node, the filters matching only this subtree are automatically applied to the other visualisations. This visualisation was much more difficult to create the the others. It required many custom features that were not available in other Kibana visualisations, so it had to be written in Vega, a visualisation grammar.

Next, there are the overall statistics for the instance, such as the sum of all quotas and how much of them is currently used. The pie chart shows the sum of the quotas – broken down by the instance in the inner circle, then by the type of the quota (user/group/project) and finally by the id in the outer circle. This allows the user to instantly see which quotas are the largest. Any of the pieces can be clicked to easily apply the appropriate filter.

ii. CURRENT QUOTA STATUS

The panels in this category focus on the individual quotas. They show the most filled quotas (Figure 8), the largest ones etc.

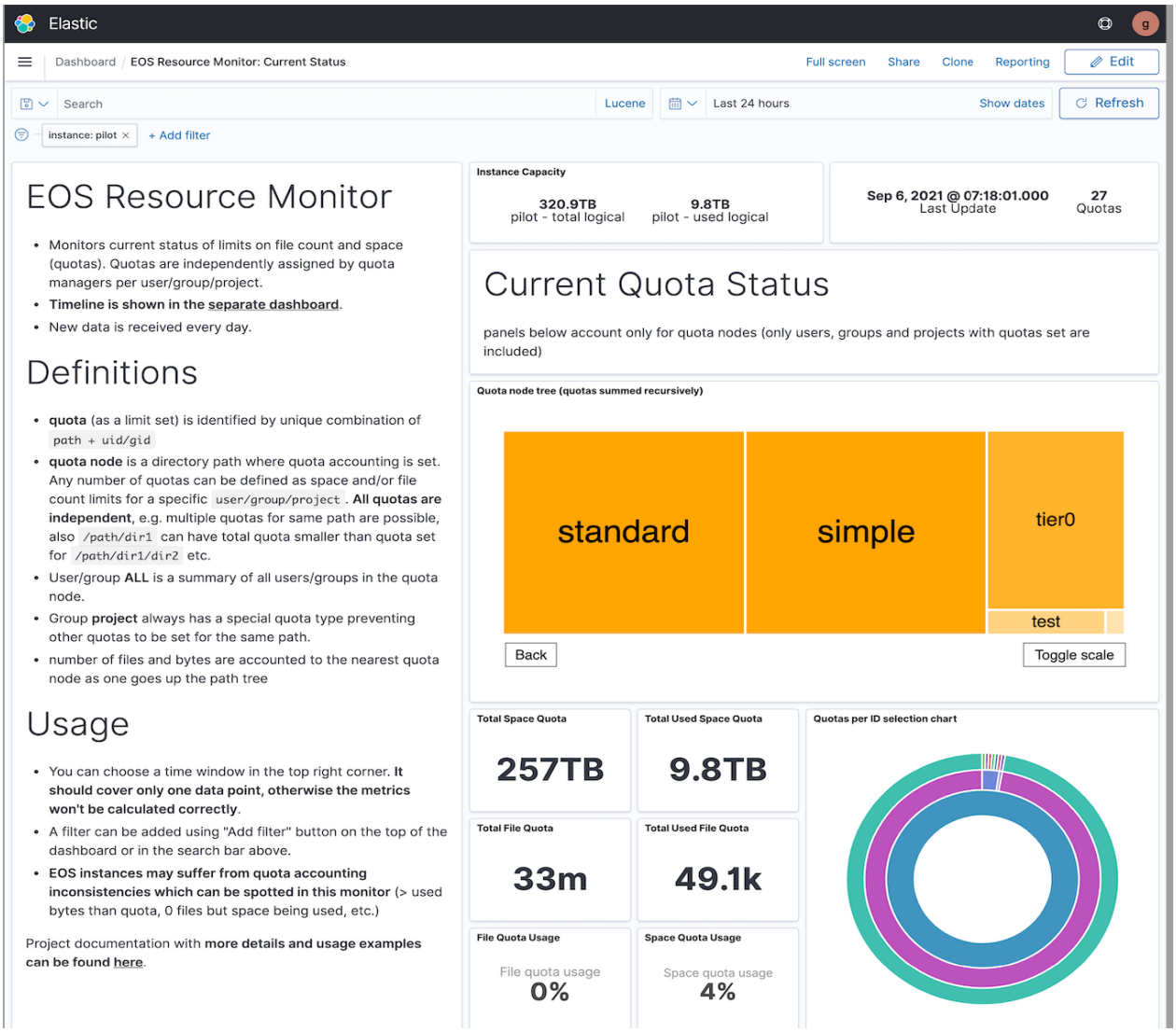

Figure 7. Overview of the instance.

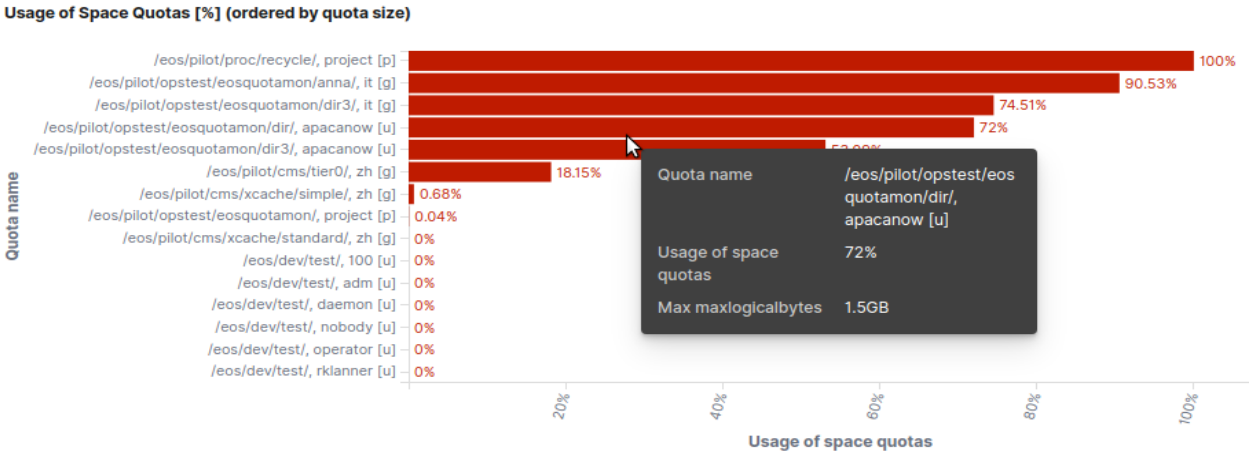

Figure 8 The most filled quotas.

iii. RESOURCE USAGE BY ALL USERS, GROUPS AND PROJECTS

These panels show all used resources, even if there is no specific limit set. The viewer can see which groups or users use the most space or files. There is also a table with the raw information, which the user can download as a csv file (Figure 9).

Path	ID type	ID	Space quota	Used space quota	File quota	Used file quota
/eos/pilot/cms/xcache/standard/	group	zh	100TB	1.1GB	1m	1
/eos/pilot/cms/xcache/standard/	group	ALL	100TB	1.1GB	1m	1
/eos/pilot/cms/xcache/simple/	group	zh	100TB	683.7GB	1m	615
/eos/pilot/cms/xcache/simple/	group	ALL	100TB	683.7GB	1m	615
/eos/pilot/opstest/	project	project	50TB	19.9GB	10m	2.8k
/eos/pilot/cms/tier0/	group	zh	50TB	11.3TB	10m	59.1k
/eos/pilot/cms/tier0/	group	ALL	50TB	11.3TB	10m	59.1k
/eos/dev/test/	user	ALL	6TB	11.6kB	6m	4
/eos/pilot/proc/recycle/	project	project	1TB	708B	1m	6
/eos/dev/test/	user	rklanner	1TB	0B	1m	0
			563TB	**48TB**	**75m**	**242k**

Export: Raw Formatted

1 2 3 4 5 … 6 »

Figure 9. Raw data table.

iv. TIMELINES

The panels shown in the timelines dashbard show similar statistics, but in different points in time – that way the user can see how they change through the time (Figure 10). The legend for specific quotas for readability displays `<dirname>, <id> [<id_type>]`, while the path is visible in the tooltip.

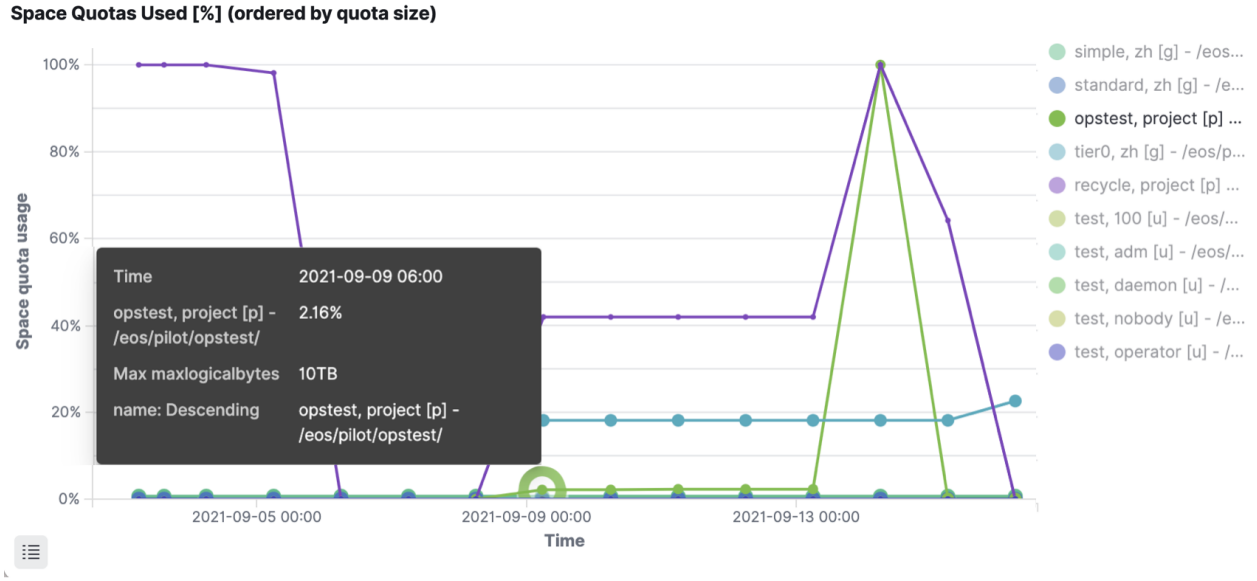

Figure 10. Total quota changes in time.

6. REFERENCES

[1] https://eos-web.web.cern.ch/eos-web/

[2] https://gitlab.cern.ch/eos/eos-resource-monitor/

[3] https://eosops-docs.web.cern.ch/tools/monitoring/eosresourcemonitor.html

[4] https://grafana.com/

[5] https://www.elastic.co/kibana/

[6] https://filer-carbon.cern.ch

[7] https://opendistro.github.io/for-elasticsearch/

[8] https://grafana.com/oss/metrictank/

[9] https://www.elastic.co/elasticsearch/

[10] https://graphiteapp.org/

[11] https://opensearch.org/

[12] https://opendistro.github.io/for-elasticsearch-docs/docs/security/access-control/multi-tenancy/

[13] https://opendistro.github.io/for-elasticsearch-docs/docs/security/access-control/users-roles/

[14] https://groups-portal.web.cern.ch/group/08d9625a-ecf7-473b-8a12-31790397f281/details

[15] https://application-portal.web.cern.ch/manage/08d8edd7-5fcf-460d-8f8a-ccb6006d54b6

[16] https://vega.github.io/vega/examples/treemap/