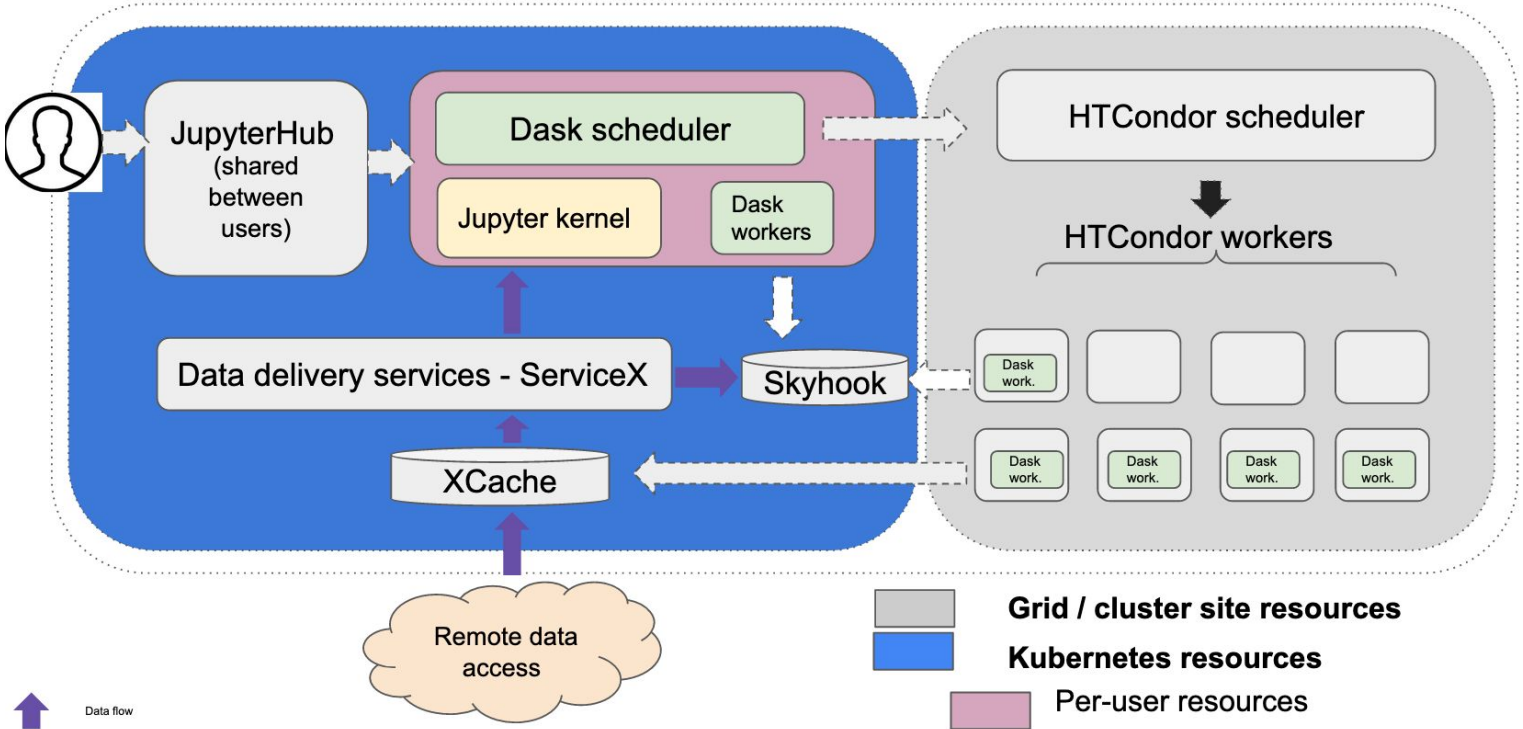
Metrics to define user-engagement activities on Coffea-Casa Analysis Facility Deployments

- Durbar Chakraborty, NIT Durgapur
  Mentors: Oksana Shadura, Ken Bloom

Current Infrastructure of Coffea Casa Facility:



Matous Adamec, Garhan Attebury, Brian Bockelman, Ken Bloom, Carl Lundstedt, John Thiltges, Oksana Shadura, Andrew Wightman

Our Objective:

coffea is a prototype package for pulling together all the typical needs of a high-energy collider physics (HEP) experiment analysis using the scientific python ecosystem. It is a HEP community project collaborating with iris-hep.

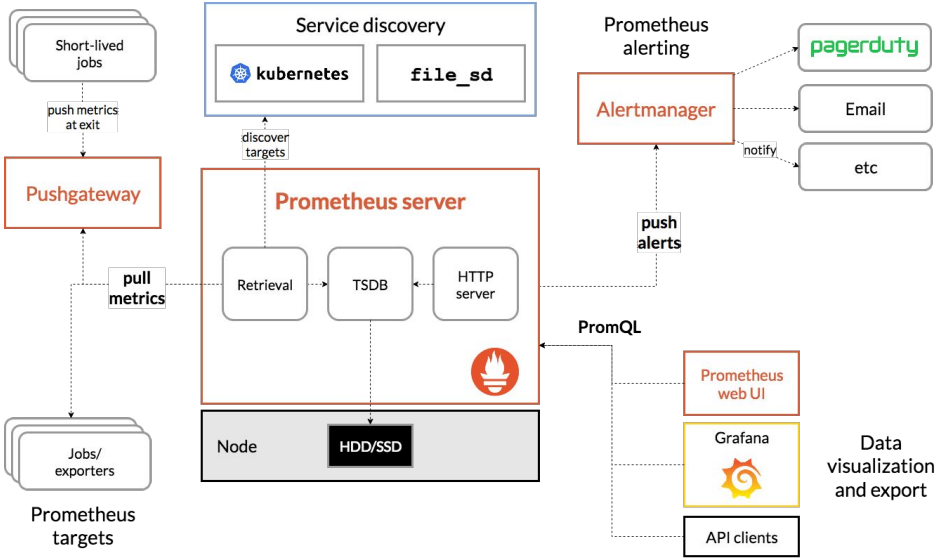
Our project defines some useful metrics on the facility, based on the data collected from various platforms which form a part of the underlying infrastructure, including *Jupyterhub*, *Dask and Kubernetes* clusters among others. Based on these metrics, we can monitor the analysis facility efficiently and get an idea of the user-engagement of the Coffea-Casa analysis facility. Some of the metrics are system metrics while others give us a more deeper insight into the activity of the individual users.

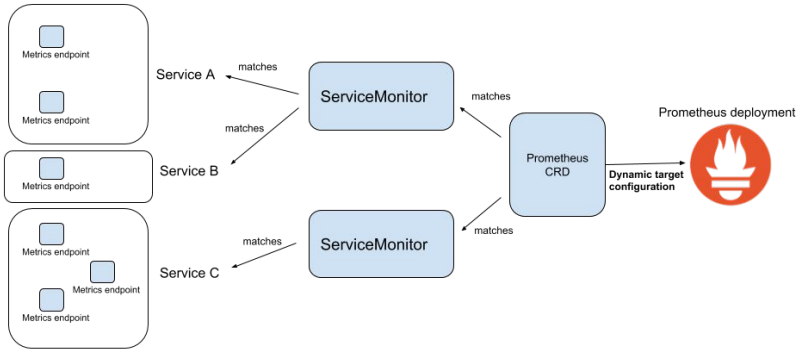
We also define a package 'af-metrics' which will automate the monitoring and data collection process for the process.

Tools we use:

- We use Prometheus monitoring tool to monitor the various metrics that we have defined on the Jupyterhub and Dask servers in the Coffea-Casa infrastructure.
- The Prometheus server runs at a specified port on the device and the user can visualise the list of active target labels from where the metrics are being scraped by Prometheus.
- Once the Prometheus server is running, we connect it to Grafana for generating easy-to-visualise dashboards using the scraped metrics.
- These dashboards give us a clear understanding of the metrics and how they help us get an idea of user-engagement of the Coffea-Casa Analysis facility.

Prometheus infrastructure





The architecture of Prometheus and some of its ecosystem components (very complex!)

We are using the similar way of the target discovery for services for prometheus

Possibilities for monitoring of coffea-casa AF installation

Coffea-casa specific endpoint (to be able to monitor coffea analysis performance)

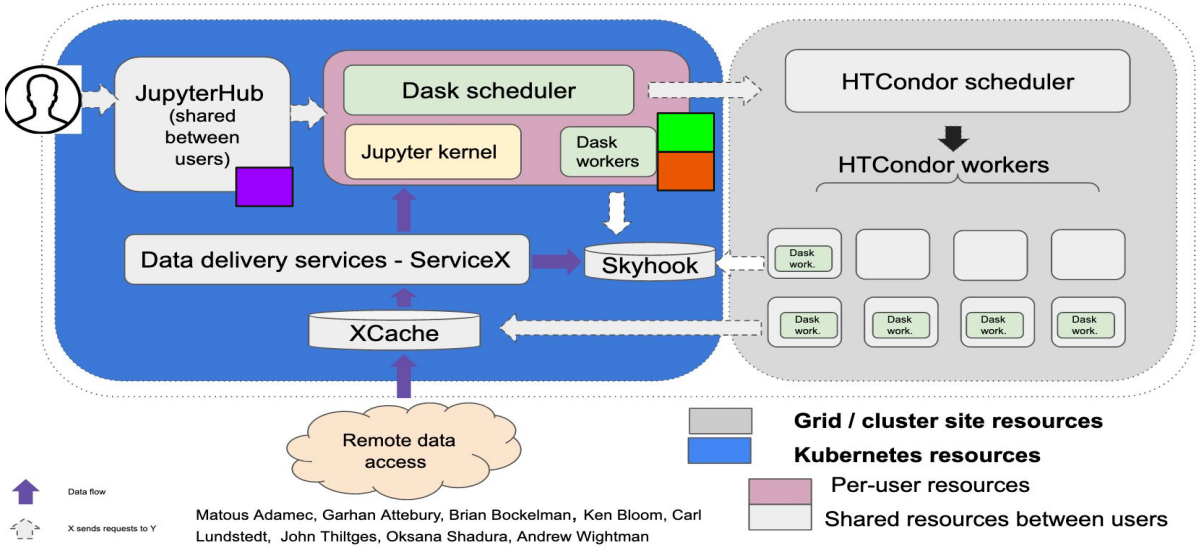
Dask.distributed exposes scheduler and worker at */metrics* endpoint

JupyterHub /metrics endpoint

Kubernetes specific endpoints available directly from Prometheus

Our Proposed Infrastructure:

Prometheus



Matous Adamec, Garhan Attebury, Brian Bockelman, Ken Bloom, Carl Lundstedt, John Thiltges, Oksana Shadura, Andrew Wightman

- JH endpoint: one for all users
- Dask scheduler endpoint: one per user
- Coffea-casa endpoint: one per users and available from af-metrics module

In the future we can add more endpoints for other services, such as ServiceX, XCache or HTCondor

Types of metrics we have looked into:

Resource utilisation metrics (for k8s, JH, Dask scheduler and workers)

Performance metrics (how fast user can run analysis?)

Community engagement metrics

(we are looking for other ideas, please help us!)

Some of the metrics we have scraped:

Jupyterhub Metrics:

Jupyterhub_total_users: Total number of users

Jupyterhub_running_servers: The number of user servers currently running

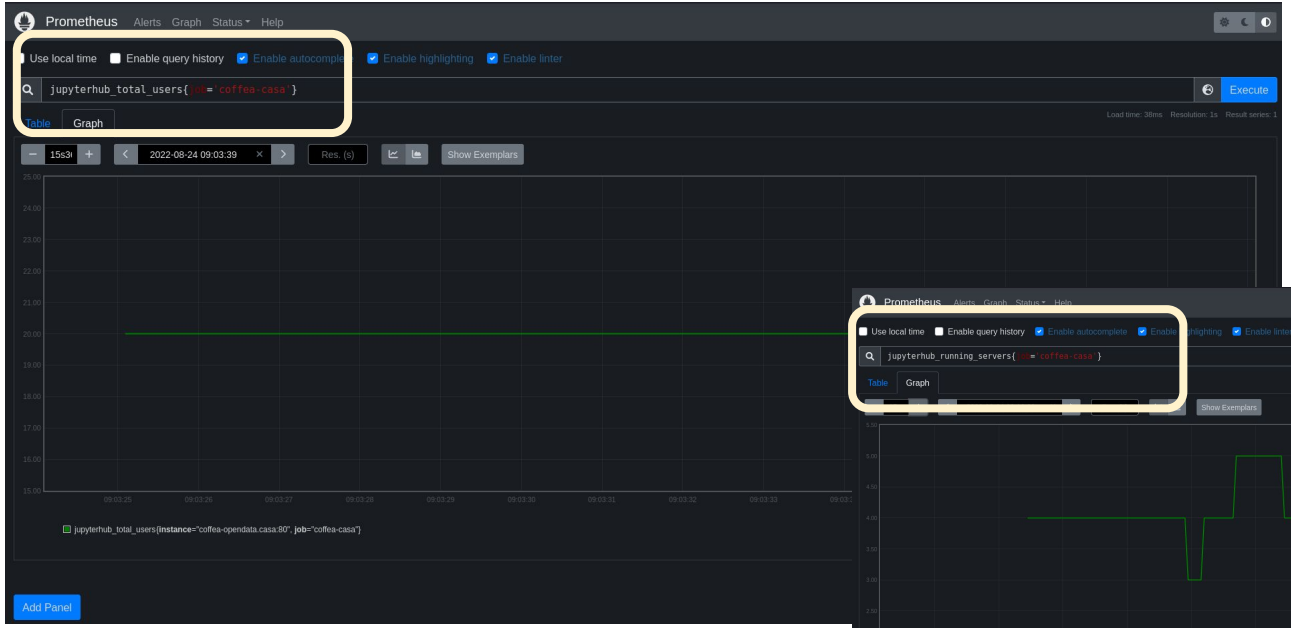
Jupyterhub_request_duration_seconds: request duration for all HTTP requests

Some of the metrics we have scraped:

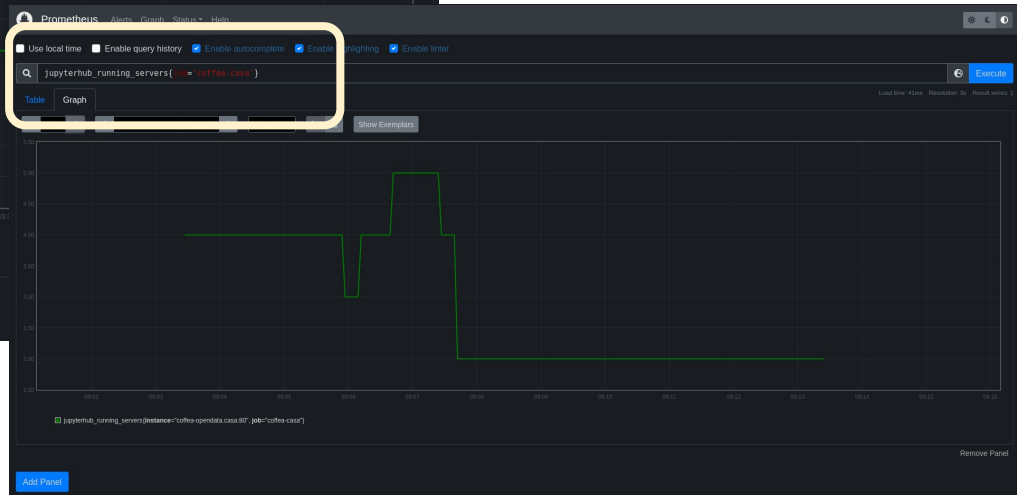
Dask Metrics:

- Dask workers
  - Idle
  - Active
  - Dead
- Total dask scheduler tasks
- Suspicious tasks
- Waiting tasks
- Errored tasks
- In memory tasks

Graph for scraped metrics in Jupyterhub for Coffea-casa OD facility



Total number of users over time = 20 (after migration to new hardware)



Total number of active users using facility (currently in this moment)

Graph for scraped metrics in Jupyterhub:

Testing request time for HTTP requests with Jupyterhub Auth OAuthAuthorizeHandler

Prometheus Alerts Graph Status Help

Use local time | Enable query history | Enable autocomplete | Enable highlighting | Enable linter

jupyterhub_request_duration_seconds_bucket{job="coffea-casa"} Execute

Load time: 55ms Resolution: 1s Result series: 600

1s5m 2022-09-12 12:26:15 Res. (s) Show Exemplars

Could be enabled for any type of handler (including custom)

35.00k
30.00k
25.00k
20.00k
15.00k
10.00k
5.00k
0.00
12:26:15

jupyterhub_request_duration_seconds_bucket{code="200", handler="jupyterhub.apihandlers.auth.OAuthAuthorizeHandler", instance="coffea-opendata.casa:80", job="coffea-casa", le="+Inf", method="GET"}
jupyterhub_request_duration_seconds_bucket{code="200", handler="jupyterhub.apihandlers.auth.OAuthAuthorizeHandler", instance="coffea-opendata.casa:80", job="coffea-casa", le="0.005", method="GET"}
jupyterhub_request_duration_seconds_bucket{code="200", handler="jupyterhub.apihandlers.auth.OAuthAuthorizeHandler", instance="coffea-opendata.casa:80", job="coffea-casa", le="0.01", method="GET"}
jupyterhub_request_duration_seconds_bucket{code="200", handler="jupyterhub.apihandlers.auth.OAuthAuthorizeHandler", instance="coffea-opendata.casa:80", job="coffea-casa", le="0.025", method="GET"}
jupyterhub_request_duration_seconds_bucket{code="200", handler="jupyterhub.apihandlers.auth.OAuthAuthorizeHandler", instance="coffea-opendata.casa:80", job="coffea-casa", le="0.05", method="GET"}
jupyterhub_request_duration_seconds_bucket{code="200", handler="jupyterhub.apihandlers.auth.OAuthAuthorizeHandler", instance="coffea-opendata.casa:80", job="coffea-casa", le="0.075", method="GET"}
jupyterhub_request_duration_seconds_bucket{code="200", handler="jupyterhub.apihandlers.auth.OAuthAuthorizeHandler", instance="coffea-opendata.casa:80", job="coffea-casa", le="0.1", method="GET"}
jupyterhub_request_duration_seconds_bucket{code="200", handler="jupyterhub.apihandlers.auth.OAuthAuthorizeHandler", instance="coffea-opendata.casa:80", job="coffea-casa", le="0.25", method="GET"}

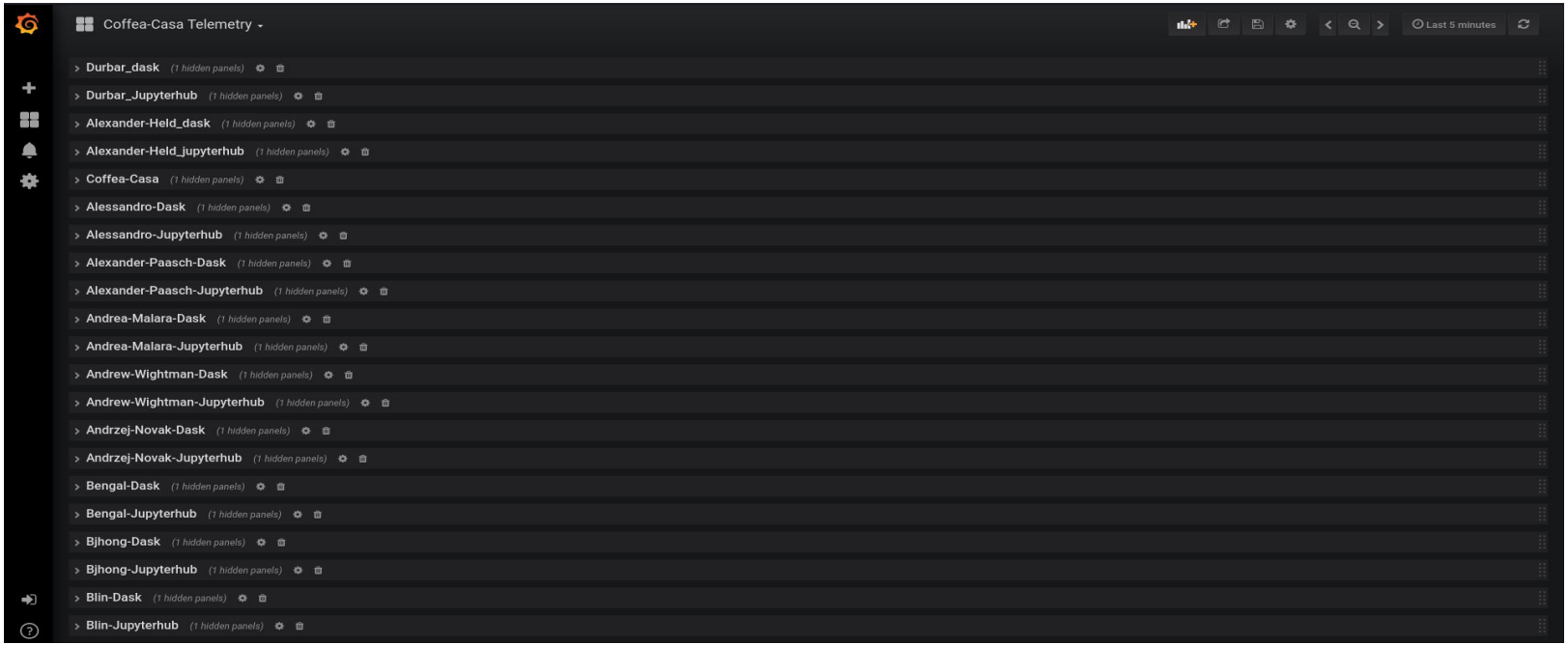
Output of endpoint(s) that collects data for all plots we will see after (available per user)

Prometheus metrics for Coffea Casa:

```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 3.801889e+06
python_gc_objects_collected_total{generation="1"} 389606.0
python_gc_objects_collected_total{generation="2"} 13074.0
# HELP python_gc_objects_uncollectable_total Uncollectable object found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 9137.0
python_gc_collections_total{generation="1"} 830.0
python_gc_collections_total{generation="2"} 27.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="8",patchlevel="10",version="3.8.10"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 5.99044096e+08
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 1.22474496e+08
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.66094773891e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 442.26
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 23.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 1.048576e+06
# HELP jupyterhub_request_duration_seconds request duration for all HTTP requests
# TYPE jupyterhub_request_duration_seconds histogram
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="0.005",method="GET"} 0.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="0.01",method="GET"} 0.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="0.025",method="GET"} 427.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="0.05",method="GET"} 525.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="0.075",method="GET"} 525.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="0.1",method="GET"} 525.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="0.25",method="GET"} 525.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="0.5",method="GET"} 525.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="0.75",method="GET"} 525.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="1.0",method="GET"} 525.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="2.5",method="GET"} 525.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="5.0",method="GET"} 525.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="7.5",method="GET"} 525.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="10.0",method="GET"} 525.0
jupyterhub_request_duration_seconds_bucket{code="200",handler="jupyterhub.apihandlers.users.UserListAPIHandler",le="+Inf",method="GET"} 525.0
```

Grafana dashboards for users on Coffea Casa OD facility:

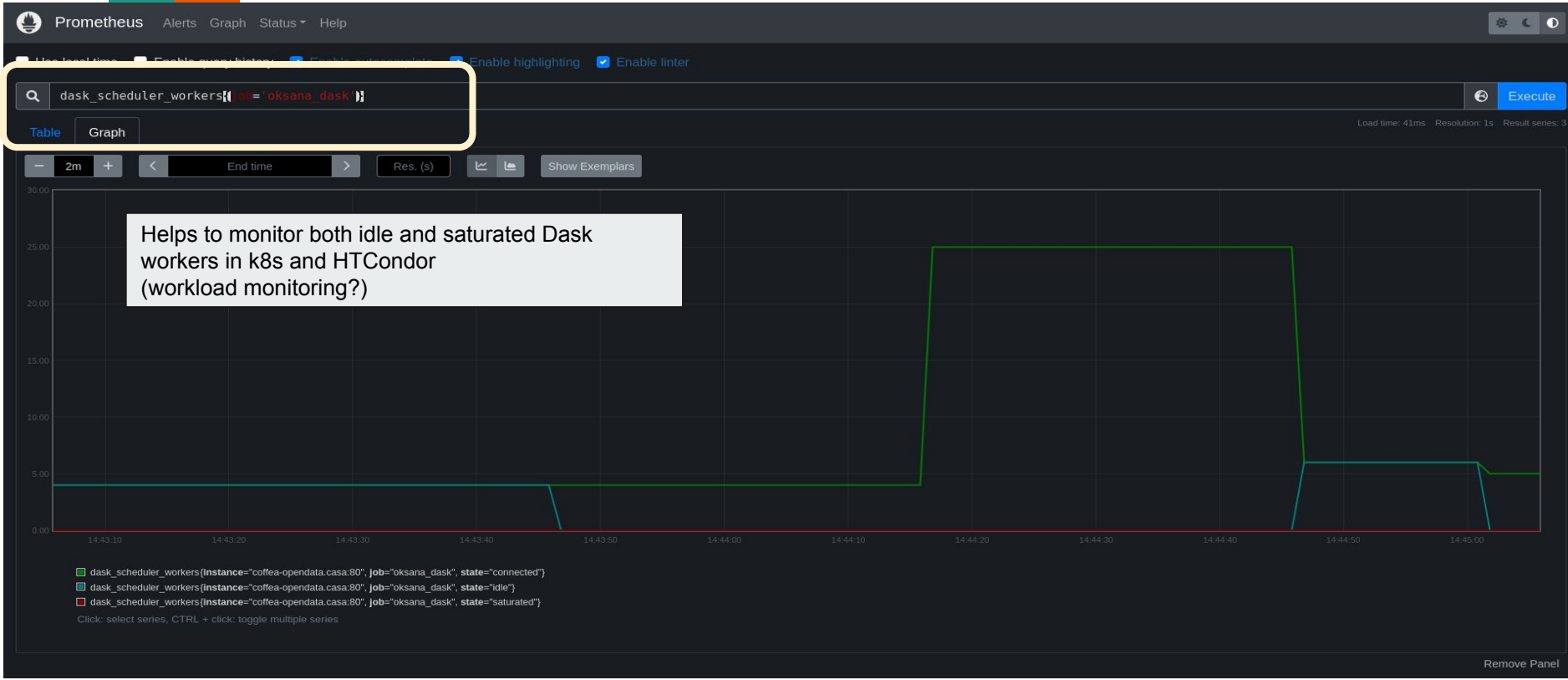
Dashboards generated per user



Graph for scraped metrics in Dask:

Testing **number of workers per user over** time

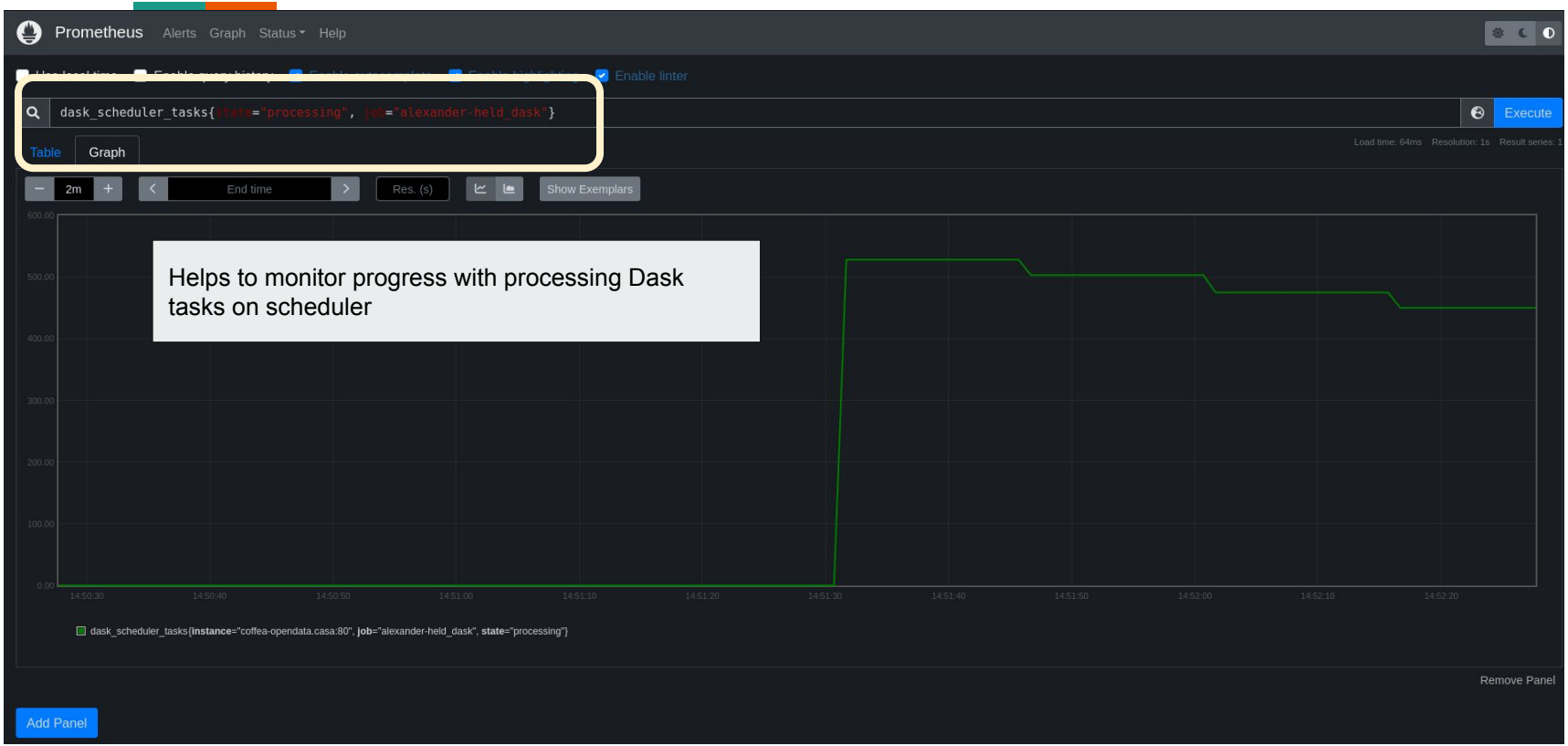
- Could be interesting for accounting?



Helps to monitor both idle and saturated Dask workers in k8s and HTCondor (workload monitoring?)

Graph for scraped metrics in Dask:

Available per user:
dashboard is shown for Alexander Held



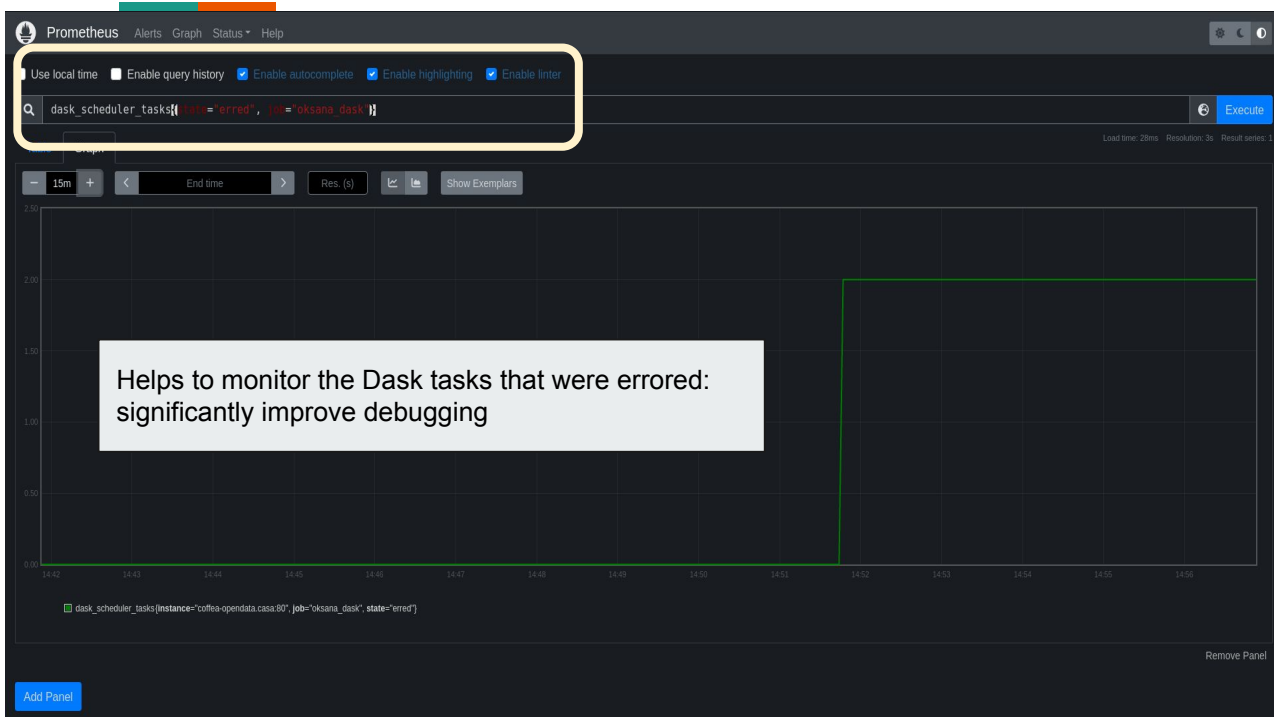
Helps to monitor progress with processing Dask tasks on scheduler

Graph for scraped metrics in Dask:

Available per user:
dashboard is shown for Oksana Shadura

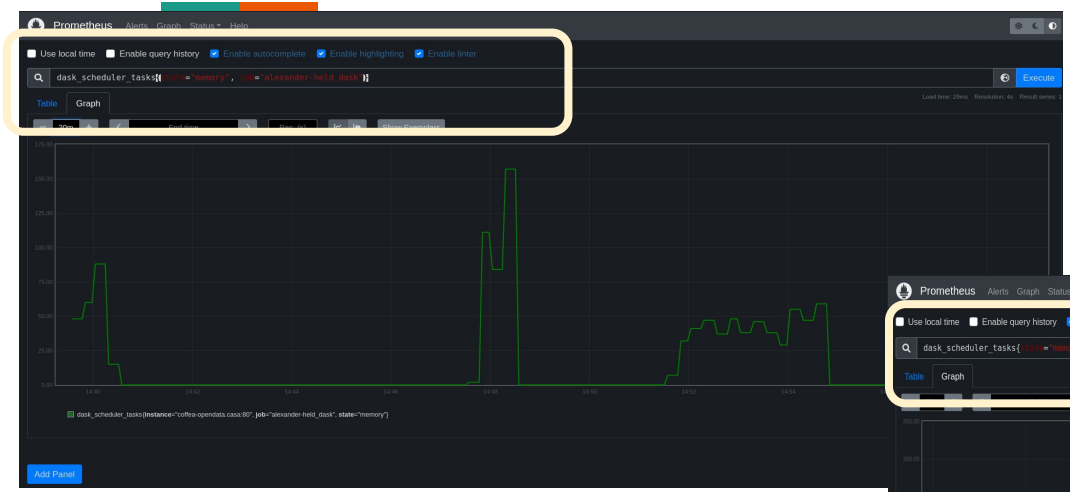
Helps to monitor the progress with processing Dask tasks on scheduler (no stuck workload)

Graph for scraped metrics in Dask:



Available per user:
dashboard is shown for Oksana Shadura

Graph for scraped metrics in Dask:



Helps to monitor the Dask tasks that were in memory: significantly improve debugging if for example we are running out of memory on workers

Dask.distributed stores the results of tasks in the distributed memory of the worker nodes. The central scheduler tracks all data on the cluster and determines when data should be freed. Completed results are usually cleared from memory as quickly as possible in order to make room for more computation

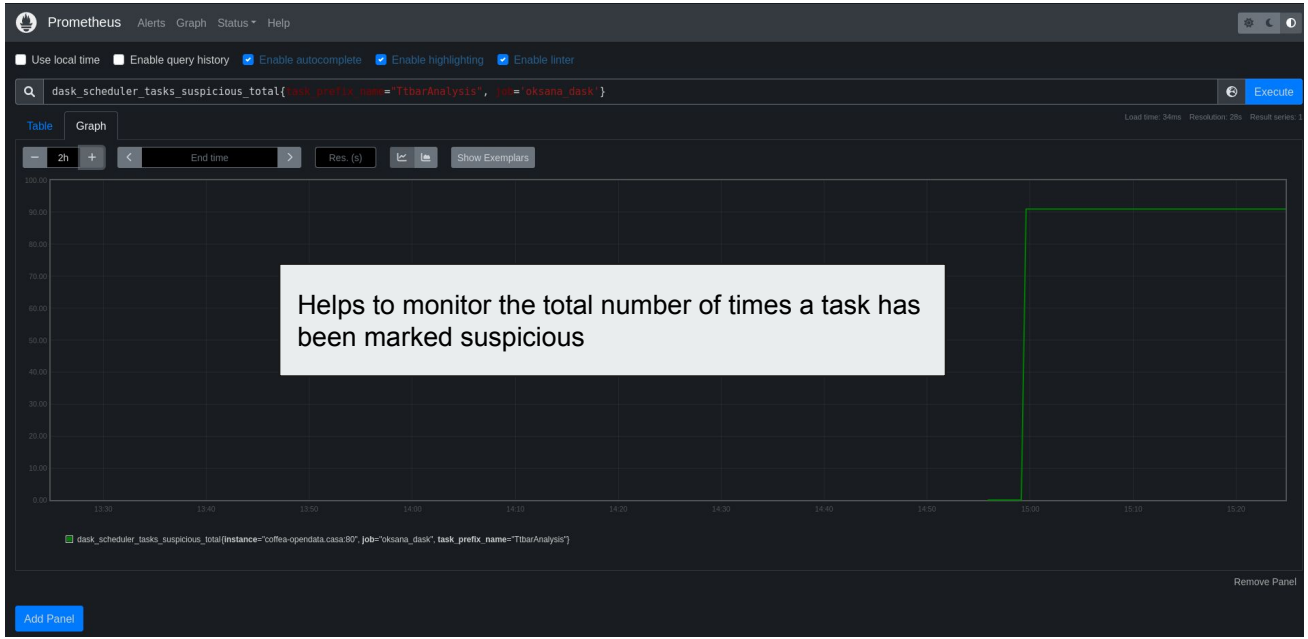
Graph for scraped metrics in Dask:

Available per user:
dashboard is shown for Oksana Shadura

Helps to monitor the Dask tasks that were in waiting state: significantly improve debugging (detect deadlocks)

Graph for scraped metrics in Dask:

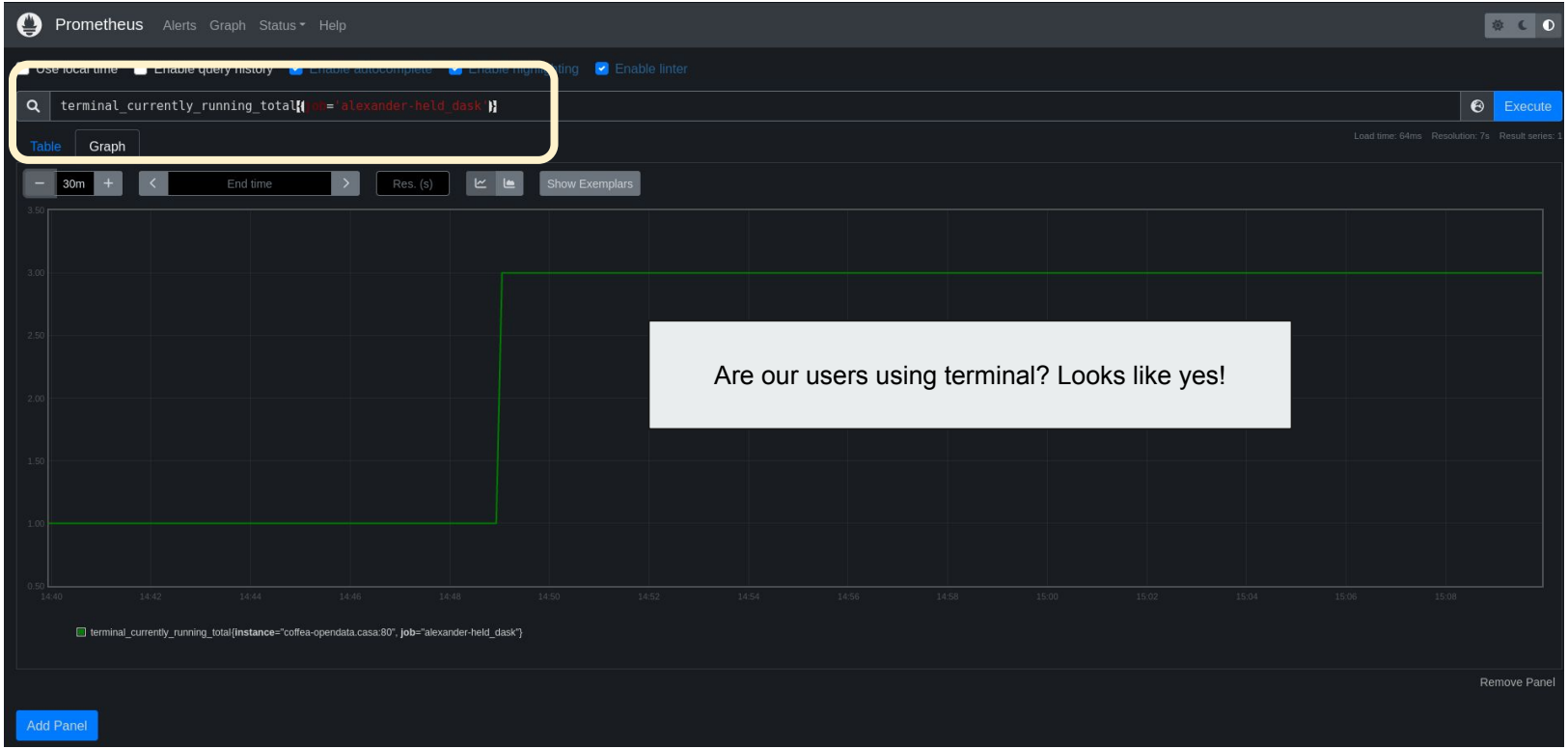
Available per user:
dashboard is shown for Oksana Shadura



The number of times task has been involved in a worker death. Some tasks may cause workers to die (such as calling `os._exit(0)`). When a worker dies, all of the tasks on that worker are re-assigned to others. This combination of behaviors can cause a bad task to catastrophically destroy all workers on the cluster, one after another. Whenever a worker dies, we mark each task currently processing on that worker as suspicious.

Graph for scraped metrics in Dask:

Available per user:
dashboard is shown for
Alex Held



Are our users using terminal? Looks like yes!

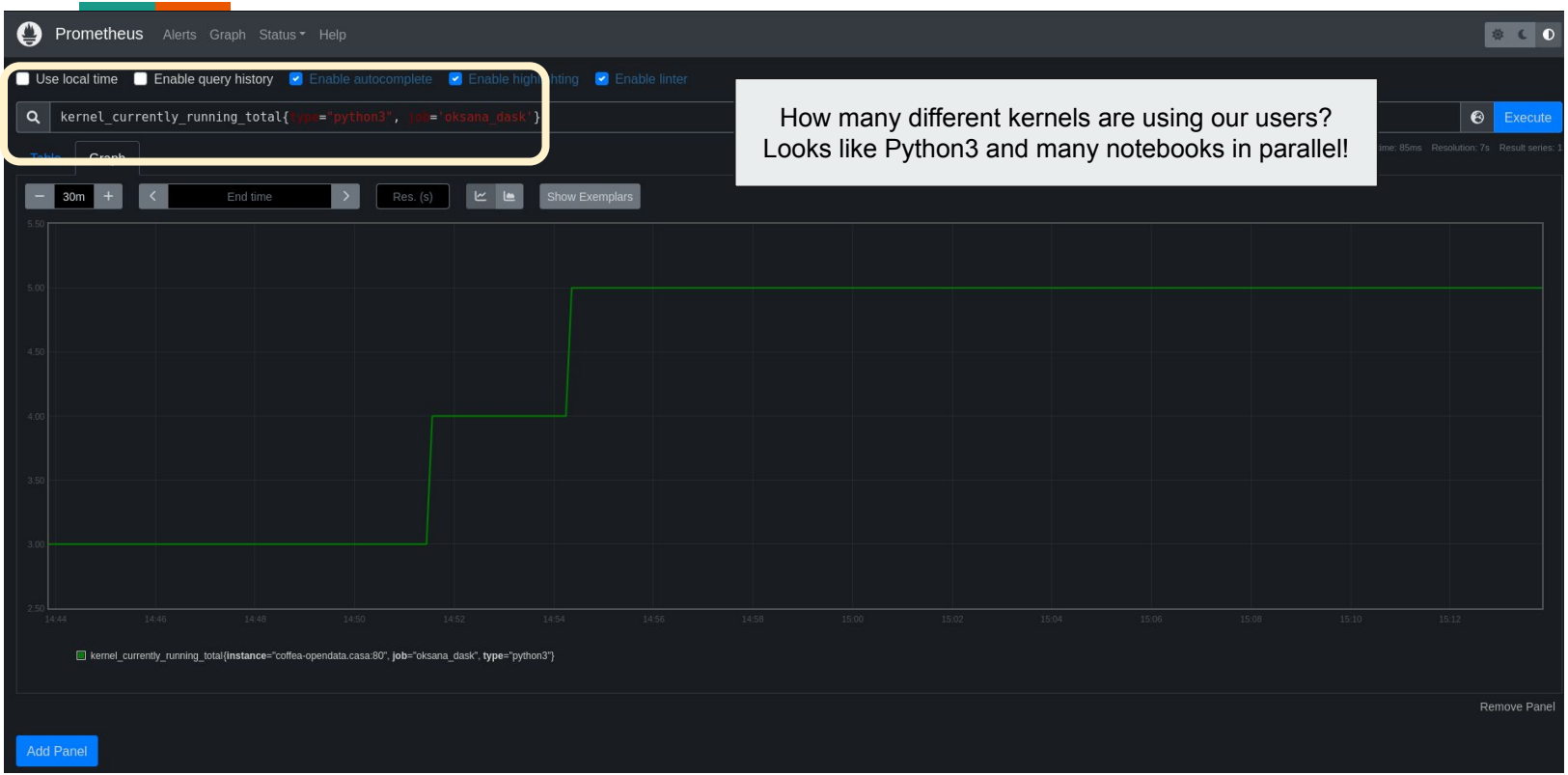
Graph for scraped metrics in Dask:

Available per user:
dashboard is shown for Oksana Shadura

Task is no longer needed by any client or dependent task, so it disappears from the scheduler as well. As soon as a task reaches this state, it is immediately dereferenced from the scheduler.

Graph for scraped metrics in Dask:

Available per user:
dashboard is shown for Oksana Shadura



How many different kernels are using our users?
Looks like Python3 and many notebooks in parallel!

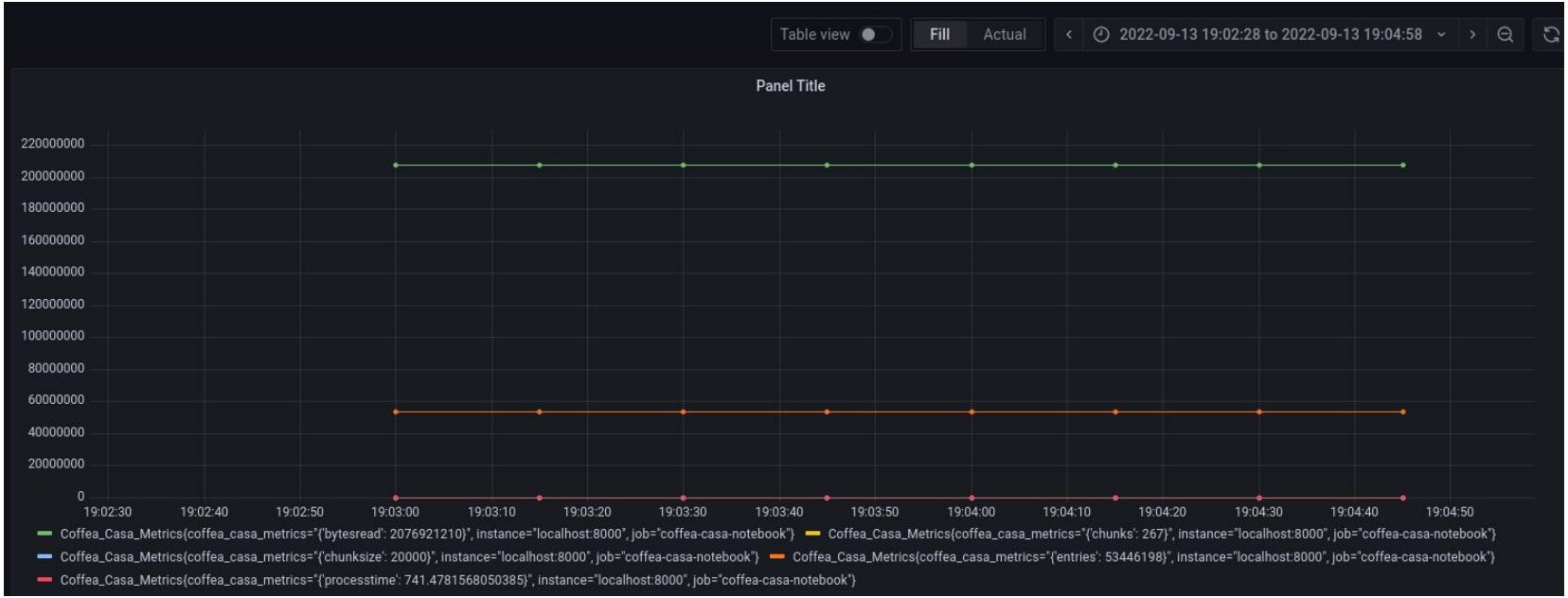
Investigating if we can use/develop own handlers to collect custom information: we are interested to collect custom metrics for AF such as efficiency of analysis

Prometheus metrics for notebook results in Coffea Casa Server:

```
# HELP python_gc_objects_collected_total Objects collected during gc
# TYPE python_gc_objects_collected_total counter
python_gc_objects_collected_total{generation="0"} 69.0
python_gc_objects_collected_total{generation="1"} 289.0
python_gc_objects_collected_total{generation="2"} 0.0
# HELP python_gc_objects_uncollectable_total Uncollectable object found during GC
# TYPE python_gc_objects_uncollectable_total counter
python_gc_objects_uncollectable_total{generation="0"} 0.0
python_gc_objects_uncollectable_total{generation="1"} 0.0
python_gc_objects_uncollectable_total{generation="2"} 0.0
# HELP python_gc_collections_total Number of times this generation was collected
# TYPE python_gc_collections_total counter
python_gc_collections_total{generation="0"} 55.0
python_gc_collections_total{generation="1"} 5.0
python_gc_collections_total{generation="2"} 0.0
# HELP python_info Python platform information
# TYPE python_info gauge
python_info{implementation="CPython",major="3",minor="10",patchlevel="6",version="3.10.6"} 1.0
# HELP process_virtual_memory_bytes Virtual memory size in bytes.
# TYPE process_virtual_memory_bytes gauge
process_virtual_memory_bytes 4.07425024e+08
# HELP process_resident_memory_bytes Resident memory size in bytes.
# TYPE process_resident_memory_bytes gauge
process_resident_memory_bytes 2.6574848e+07
# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
process_start_time_seconds 1.66126333172e+09
# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.21
# HELP process_open_fds Number of open file descriptors.
# TYPE process_open_fds gauge
process_open_fds 31.0
# HELP process_max_fds Maximum number of open file descriptors.
# TYPE process_max_fds gauge
process_max_fds 8192.0
# HELP Coffea_Casa_Metrics Custom_facility_metrics
# TYPE Coffea_Casa_Metrics gauge
Coffea_Casa_Metrics{coffea_casa_metrics="{'bytesread': 2076921210}"} 0.0
# HELP Coffea_Casa_Metrics Custom_facility_metrics
# TYPE Coffea_Casa_Metrics gauge
Coffea_Casa_Metrics{coffea_casa_metrics="{'entries': 53446198}"} 0.0
# HELP Coffea_Casa_Metrics Custom_facility_metrics
# TYPE Coffea_Casa_Metrics gauge
Coffea_Casa_Metrics{coffea_casa_metrics="{'processtime': 741.4781568050385}"} 0.0
# HELP Coffea_Casa_Metrics Custom_facility_metrics
# TYPE Coffea_Casa_Metrics gauge
Coffea_Casa_Metrics{coffea_casa_metrics="{'chunks': 267}"} 0.0
```

Investigating if we can use/develop own handlers to collect custom information: we are interested to collect custom metrics for AF such as efficiency of analysis (events/second/thread, how quickly we process data and etc.)

Grafana Dashboard for Coffea Casa Notebook:



Future plans

- Deploy and test all achieved results on Opendata and CMS coffea-casa analysis facilities
- Investigate more complex analysis facilities metrics
- Add custom request handler to serve efficiently analysis-specific AF metrics (https://jupyter-notebook.readthedocs.io/en/stable/extending/handlers.html)