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:

- JupyterHub (shared between users)
- Dask scheduler
  - Jupyter kernel
  - Dask workers
- Data delivery services - ServiceX
- Skyhook
- XCache

Data flow:
- X sends requests to Y

Grid / cluster site resources
- Kubernetes resources
  - Per-user resources
  - Shared resources between users

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.
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:

- 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

Could be enabled for any type of handler (including custom)
Prometheus metrics for Coffea Casa:

```python
# Prometheus objects collected during gc
python_gc_objects_collected_total {generation=0} 1888800440
python_gc_objects_collected_total {generation=1} 0
python_gc_objects_collected_total {generation=2} 1888800440

# Prometheus objects uncollected total
python_gc_objects_uncollected_total 0

# Prometheus objects uncollectable total
python_gc_objects_uncollectable_total 0

# Prometheus objects uncollectable, total = Prometheus object found during GC
python_gc_objects_uncollectablefound_total 0

# Prometheus objects & endpoints that collect data for all plots.

# Endpoints: https://example.com/api/v1/metrics

# Available per user:

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

Helps to monitor progress with processing Dask tasks on scheduler

Available per user: dashboard is shown for Alexander Held
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

Helps to monitor the Dask tasks that were errored: significantly improve debugging
Graph for scraped metrics in Dask:

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.

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

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!
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:

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

Available per user: dashboard is shown for Oksana Shadura
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:
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.)
Future plans

- Deploy and test all achieved results on Opendata and CMS coffea-casa analysis facilities
- Investigate more complex analysis facilities metrics