USDF Monitoring

Jhonatan Amado (FNAL).

U.S. DEPARTMENT OF
ENERGY

VERA C. RUBIN OBSERVATORY

Outline

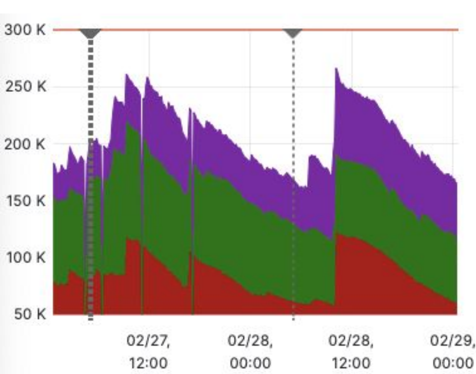
1. Why monitoring?
    a. Quick brainstorm. We need to answer some question first!
2. Database
    a. influxDB
    b. Prometheus
    c. Loki
3. Grafana
    a. Data source (**influxDB**, **Prometheus**, Elasticsearch, MySQL PostgreSQL, **Loki**.. others)
    b. Visualization (time series, tables, logs, heatmaps.gauges,histograms. Other plugins provides by grafana devs)
    c. Dynamic dashboards
    d. Alerting (email,slack)

VERA C. RUBIN
OBSERVATORY

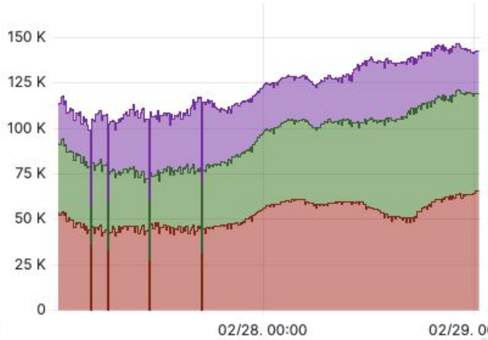
WHY monitoring

Why monitoring?

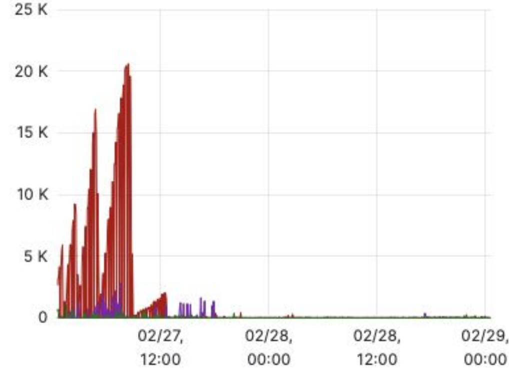
1. Answer question of what the WM is doing.



Jobs Created



Jobs Pending



Jobs Failed

2. What other information?

Data Bases

VERA C. RUBIN OBSERVATORY

DataBases (influxDB)

1. Influxdb is a time-series database. Designed for time series [aggregation in time**]
    a. Monitoring metrics and events in “real-time”
    b. “Schema-less” design that simplifies data ingestion (telegraf)
    c. influxQL (~SQL)
    d. Data is organized in table,tags,fields
        i. Tags: indexed metadata useful to group / filter metrics
            1. Query tags is fast **BUT!!** Each new tag = new index. Known issue called **cardinality**
            2. Categorical information. Not abuse of tags to allow good behaviour in the system
        ii. Fields: Actual measure. Not indexed
            1. Fields is where we store metrics that change over time

VERA C. RUBIN
OBSERVATORY

DataBases (influxDB)

2. Collector and reporting the metrics used by USDF is telegraf.
3. Syntax:
   a. table_name,tag1=x,tag2=y field1=z field2=w timestamp[ns]-<(optional+good_practice)
4. Code monitoring USDF
5. Take the necessary amount of time [x2] to design & think how many tags,fields can/will be used
6. Who will create/give support to monitoring.

```
squeue,user=lsstsvc1,partition=roma,account=rubin:production,qos=normal,state=RUNNING
jobs=2i,cpu=9i,gpu=0i,mem=32769i,tasks=9i,billing=9i

squeue_json,account=rubin:production,job_state=RUNNING,partition=milano,qos=normal,user_name=lsstsvc1,state_reason=None,facility
=production,repo=rubin,long_pending=False,long_running=True,multi_partition=False,multi_host=False,node=sdfmilan231
accrue_time=1708890451u,job_id=39921762u,cpus=1u,priority=804u,start_time=1708890453u,submit_time=1708890451u,eligible_time=1
708890451u,end_time=1709149653u,suspend_time=0u,preempt_time=0u,pre_sus_time=0u,comp_startime=2u,memory=32000u,cores=1u
1708890457814467072
```

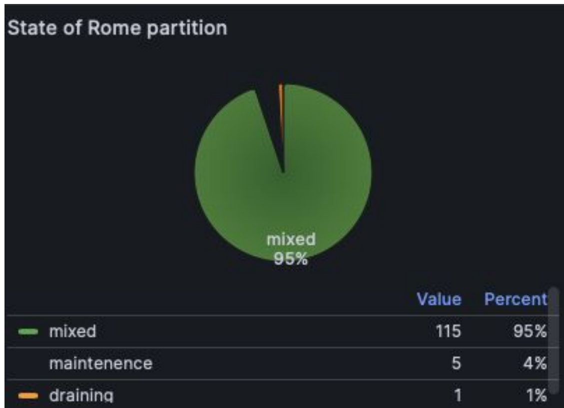
VERA C. RUBIN OBSERVATORY

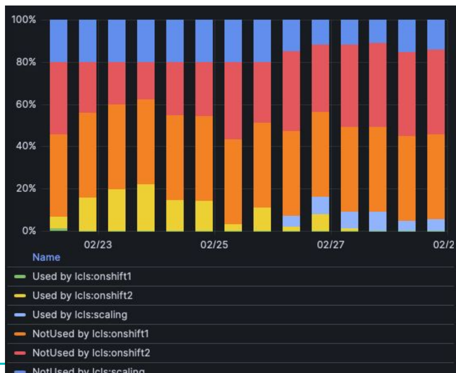
DataBases (influxDB) Cons&Pros

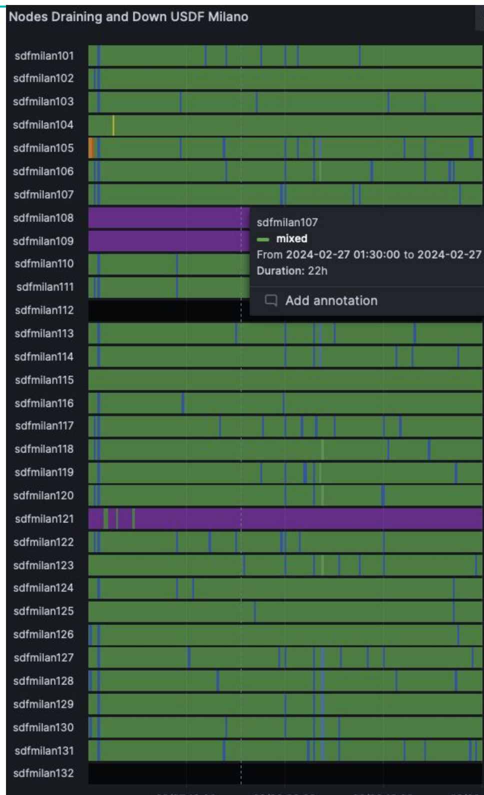
5. Pros:
   a. Optimized for time-series and efficient data compression
   b. High-velocity data ingestion
6. Cons:
   a. High cardinality challenges
   b. Know influxql (~sql)
   c. Complex queries can demand a fair amount of computing resources
7. Example query:

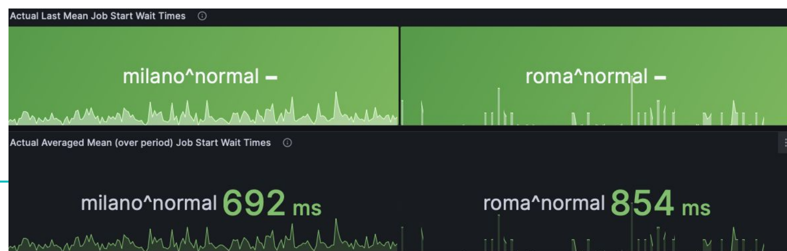
SELECT operator("fieldx") FROM "table_name" WHERE (“tag1”=”tag1value”) AND $timeFilter GROUP BY time(),“tag1”, “tag2”

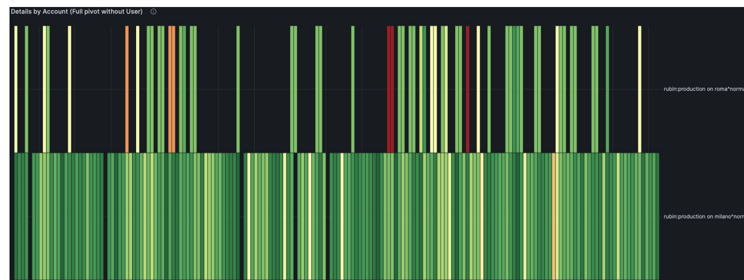
Visualization







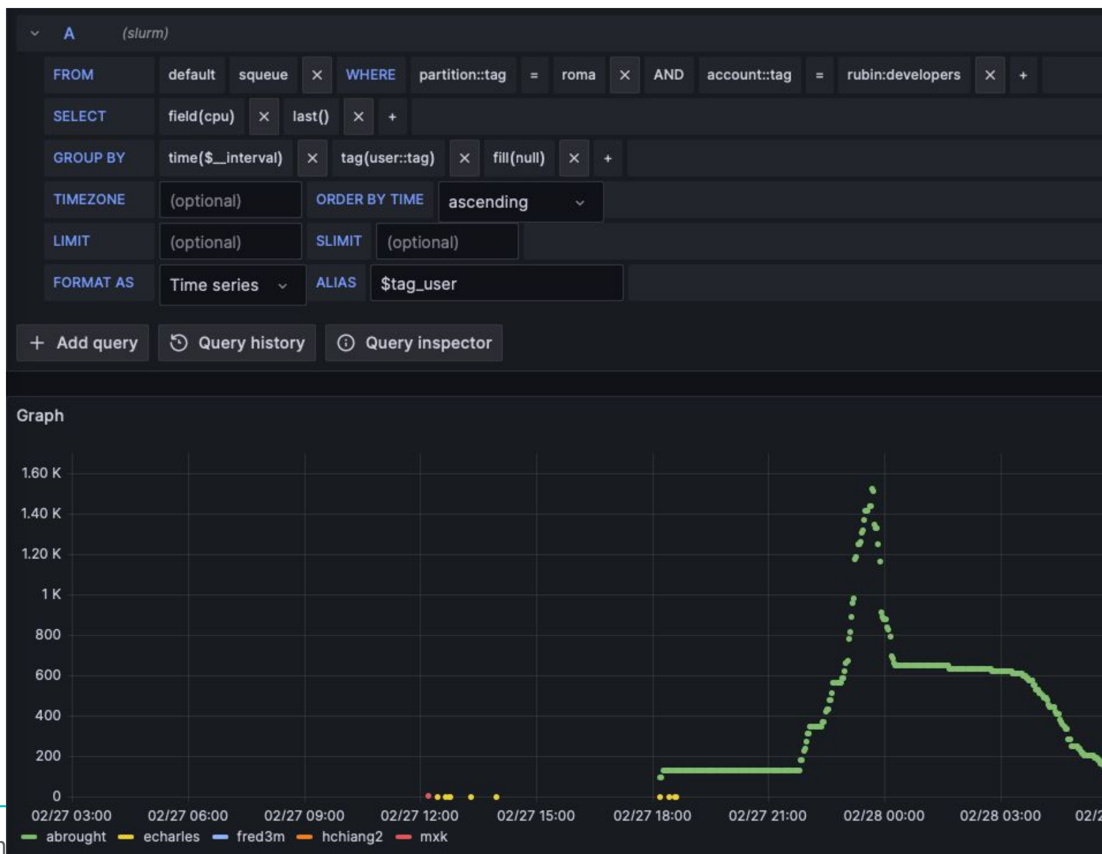




Explore

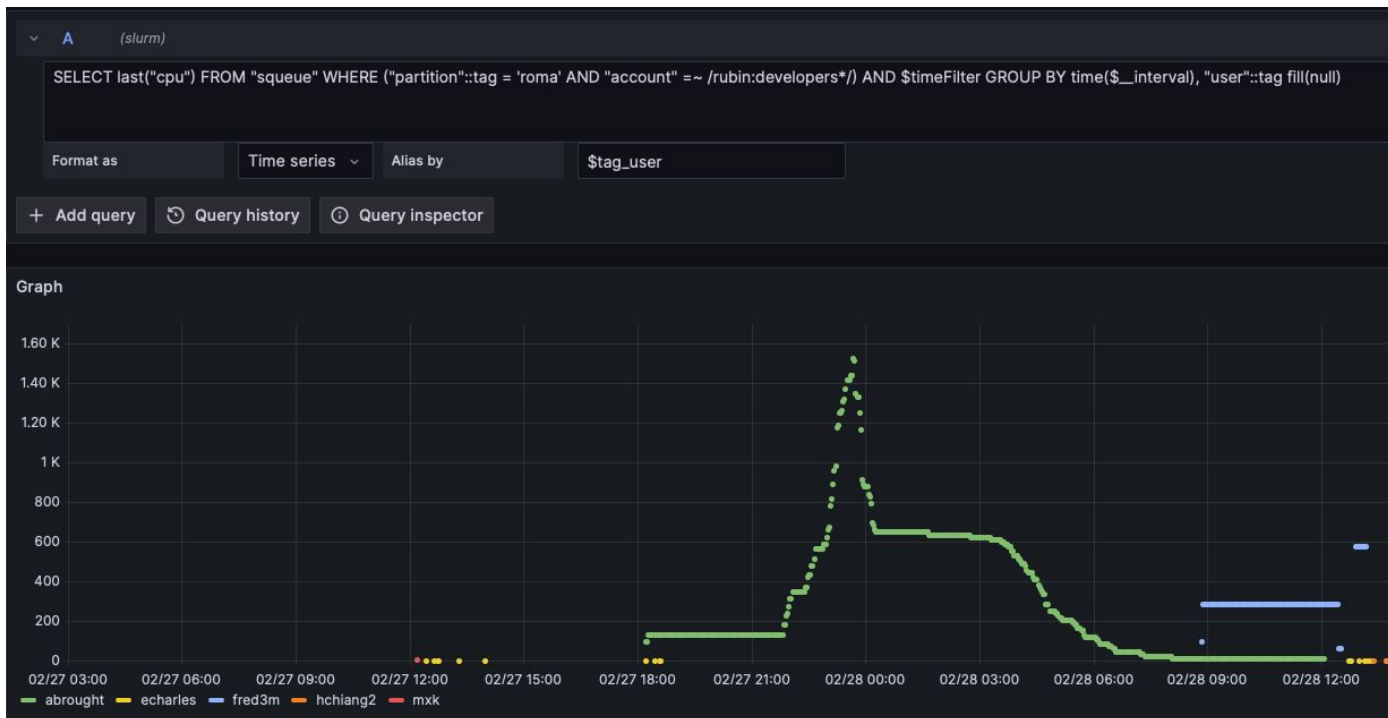
Home

- Home
- Starred
- Dashboards
- Explore
- Alerting
- Connections
- Administration



Explore

Home
Home
Starred
Dashboards
Explore
Alerting
Connections
Administration



VERA C. RUBIN
OBSERVATORY

Exercise!!

Exercise!

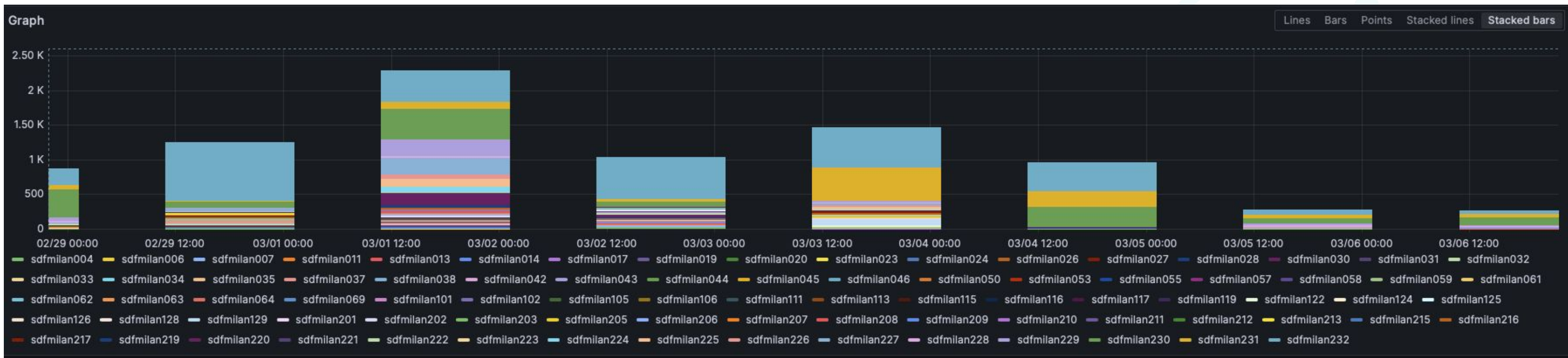
1. From the following metrics, lets try to solve the following question.

squeue_json,account=rubin:production,job_state=RUNNING,partition=milano,qos=normal,user_name=lsstsvc1,**state_reason**=None,facility=production,repo=rubin,long_pending=False,long_running=True,multi_partition=False,multi_host=False,node=sdfmilan231 accrue_time=1708890451u,job_id=39921762u,cpus=1u,priority=804u,start_time=1708890453u,submit_time=1708890451u,eligible_time=1708890451u,end_time=1709149653u,suspend_time=0u,preempt_time=0u,pre_sus_time=0u,comp_startime=2u,memory=32000u,cores=1u 1708890457814467072

Is it possible to find the node(s) from the account starting with rubin.* where jobs failed with out of memory?

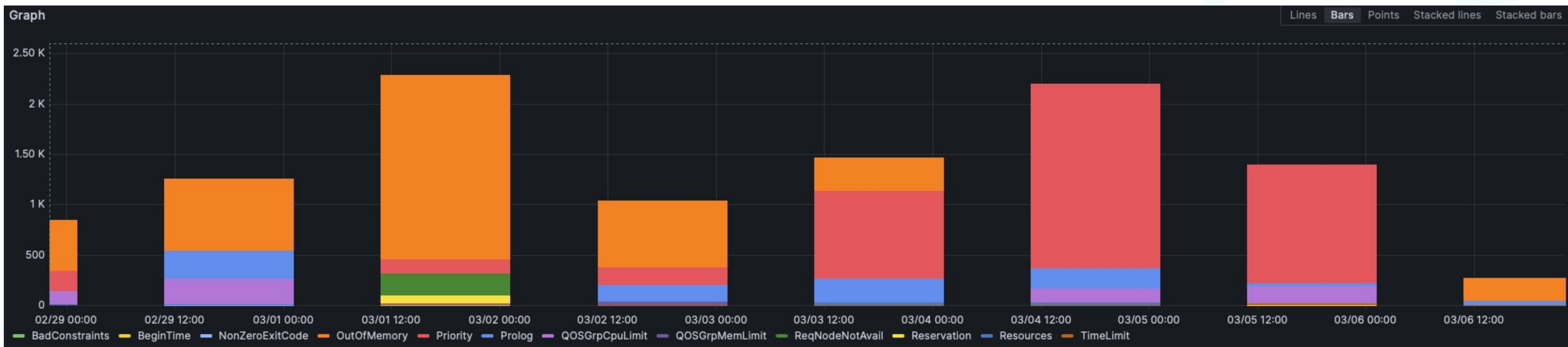
Exercise!

SELECT time, count("job_identifier") FROM (
SELECT distinct("job_id") as "job_identifier" FROM "squeue_json" WHERE ("partition"::tag = 'milano' AND "account" =~ /rubin.*/ AND "state_reason" = 'OutOfMemory') AND $timeFilter GROUP BY time($__interval), "node" fill(null))
GROUP BY time(1d), "node" fill(null)



What about per each error?

SELECT time, count("job_identifier") FROM (
SELECT distinct("job_id") as "job_identifier" FROM "squeue_json" WHERE ("partition"::tag = 'milano' AND "account" =~ /rubin.*/ AND "state_reason" != 'None') AND $timeFilter GROUP BY time($__interval), "node" fill(null))
GROUP BY time(1d), "state_reason" fill(null)



VERA C. RUBIN OBSERVATORY

Prometheus

1. Used in microservices, k8. Stores data as time series
   a. Identified by **metric_name** and **key/value**
   b. **Each** different metric_name and combination of **key/value** is a different time series.
   c. PromQL.
   d. Easy scalable. Efficient handling large volume of metrics.
   e. Data with a schema. metric_name{key1=value1,key_n=value_n} value
      i. Cardinality issue
         1. Label design
         2. Try to create encoded labels (not dynamically). **AVOID** dynamic labels (labels that change frequently like timestamp or UUID).
            a. Better approach (but not the best) create a new metric_name
      ii. Take the time x2 to design the schema.
   f. Different type of metrics (counter, gouge, histogram, summary)

VERA C. RUBIN OBSERVATORY

Prometheus

2. Complex Analysis:
    a. Association from different metrics_names.
        i. Complexity associated to PromQL but allows to deliver analysis on what's going on. (association between **key/values**)
3. Used in :
    a. Kubernetes Clustering Monitoring
    b. Microservices Monitoring
    c. Infrastructure health (hardware,network, others)
4. USDF scraps metrics from an endpoint.

VERA C. RUBIN
OBSERVATORY

Example.

Measure the ingestion of data from the different instruments at the summit.

Insights:

1. Redis DB
2. Ingested files contains 3 keys in the Redis dbs. **Recv, Ingested** and **OBS_DAY**

VERA C. RUBIN OBSERVATORY

Example.

Measure the ingestion of data from the different instruments at the summit.

Insights:

1. Redis DB
2. Ingested files contains 3 keys in the Redis dbs. **Recv, Ingested** and **OBS_DAY**

Solution:

Get from the database all the files and:

Compute with a microservice all the latencies for all the files transfered for all the instruments for all obs_day

Example.

```
# HELP redis_latencies_ingested_data Latency distribution
# TYPE redis_latencies_ingested_data histogram
redis_latencies_ingested_data_bucket{day_window="5_days_ago",instrument="LATISS",le="0.75",obs_day="20240301",type="latency"} 1288.0
redis_latencies_ingested_data_bucket{day_window="5_days_ago",instrument="LSSTComCam",le="0.75",obs_day="20240301",type="latency"} 36.0
# HELP redis_metric_ingested_data Latency and failure metrics for instruments
# TYPE redis_metric_ingested_data gauge
redis_metric_ingested_data{day_window="5_days_ago",instrument="LSSTComCam",obs_day="20240301",type="min_latency"} 0.05552792549133301
redis_metric_ingested_data{day_window="5_days_ago",instrument="LSSTComCam",obs_day="20240301",type="max_latency"} 0.36402440071105957
redis_metric_ingested_data{day_window="5_days_ago",instrument="LSSTComCam",obs_day="20240301",type="mean_latency"} 0.1658170223236084
redis_metric_ingested_data{day_window="today",instrument="LSSTComCam",obs_day="20240306",type="min_latency"} 0.09688687324523926
redis_metric_ingested_data{day_window="today",instrument="LSSTComCam",obs_day="20240306",type="max_latency"} 0.6725368499755859
redis_metric_ingested_data{day_window="today",instrument="LSSTComCam",obs_day="20240306",type="mean_latency"} 0.3714818292193942
redis_metric_ingested_data{day_window="today",instrument="LATISS",obs_day="20240306",type="min_latency"} 0.05445551872253418
redis_metric_ingested_data{day_window="today",instrument="LATISS",obs_day="20240306",type="max_latency"} 0.6649188995361328
redis_metric_ingested_data{day_window="today",instrument="LATISS",obs_day="20240306",type="mean_latency"} 0.11189095513862476
```

Example.

Measure the ingestion of data from the different instruments at the summit.

1. Measure in order of how many files per second, per instrument per obs_day
   a. Using PromQL with irate() between events (different time) for the same time series.
      i. redis_metric_ingested_data{day_window="5_days_ago",instrument="LSSTComCam",obs_day="20240301"} 2367
      ii. redis_metric_ingested_data{day_window="5_days_ago",instrument="LSSTComCam",obs_day="20240301"} 2460
2. Give statistics for files per instrument per obs_day how long it take to be ingested.

To run this microservice ->

1. https://github.com/slaclab/rubin-embargo-metrics/pull/1
2. https://github.com/slaclab/usdf-embargo-deploy/pull/15

JobSpec from panda ->

- https://github.com/PanDAWMS/panda-client/blob/36cdaf529ead1ec7fd1d43489e2e10228687b094/pandaclient/JobSpec.py#L14