EAF and Monitoring

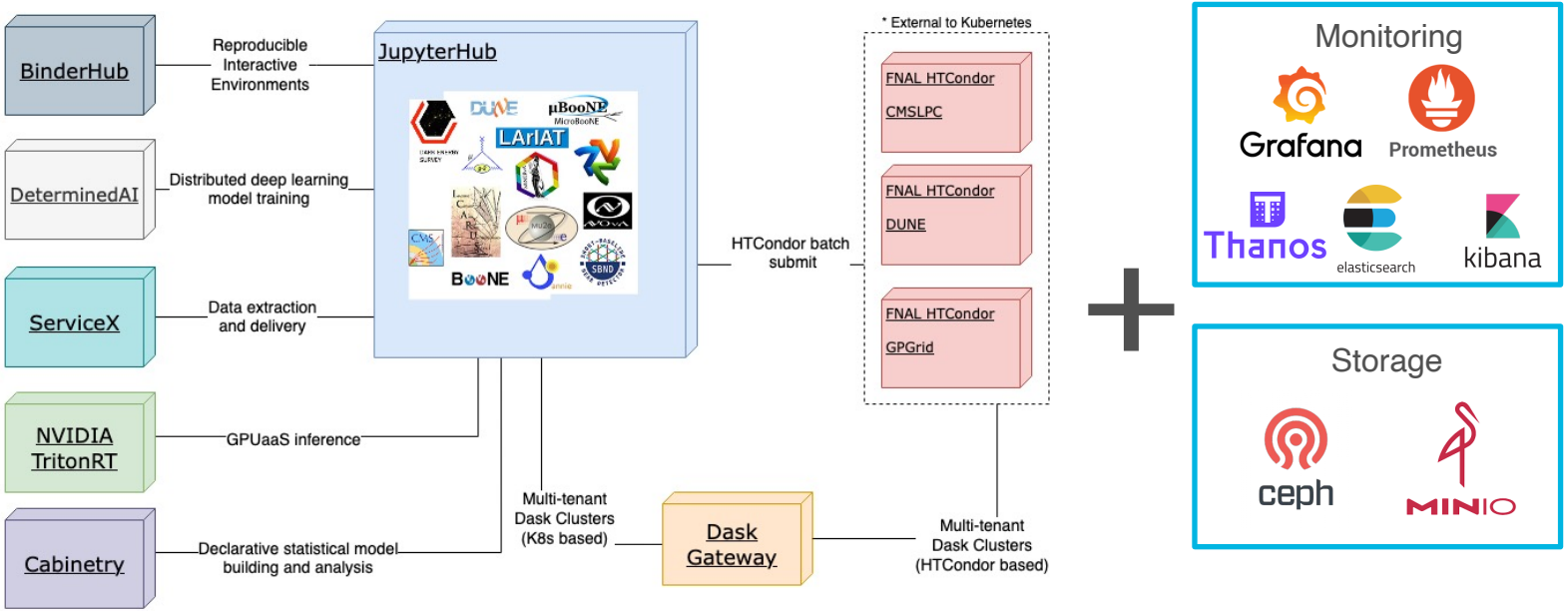
Burt Holzman, on behalf of the EAF team

FNAL Monitoring Workshop

July 24th, 2024

EAF applications ecosystem

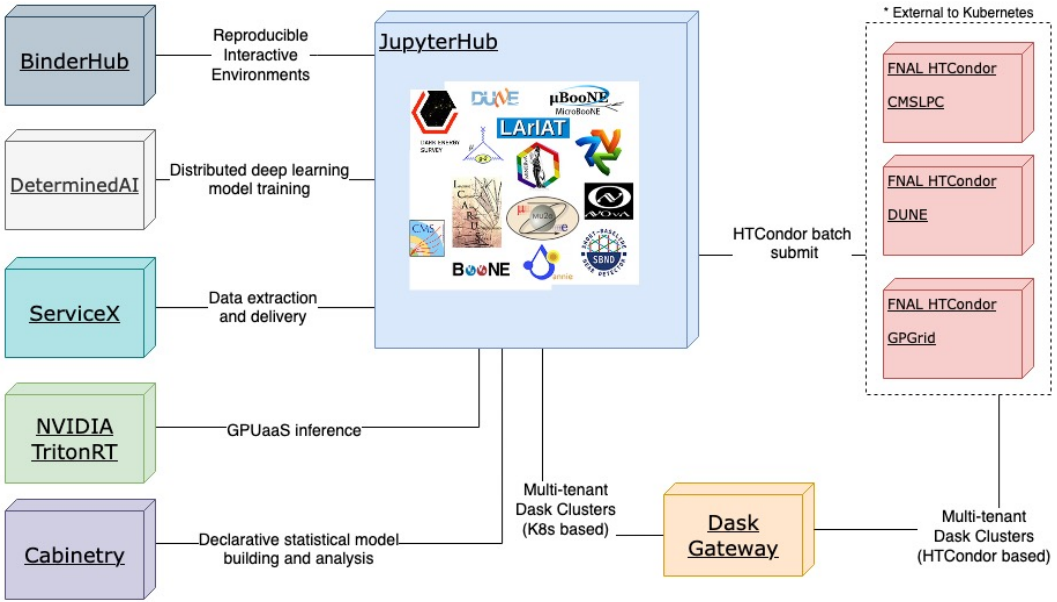
Secure | Integrated & functional | Multi-VO | DevOps (operational sustainability) | Active collaboration

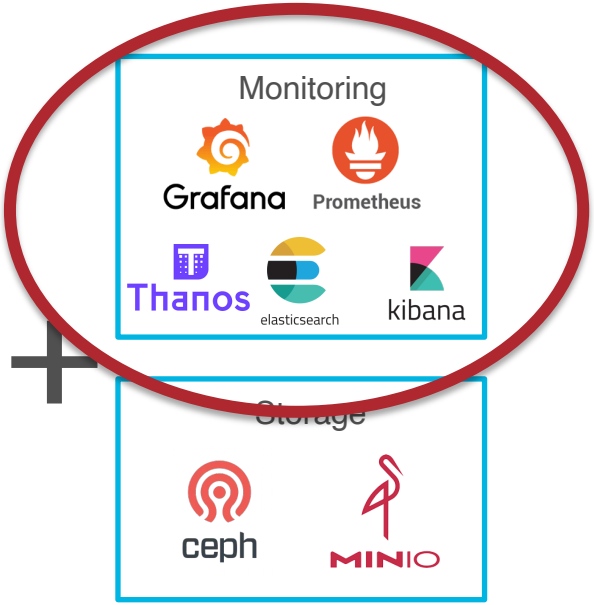


Fermilab

EAF applications ecosystem

Secure | Integrated & functional | Multi-VO | DevOps (operational sustainability) | Active collaboration



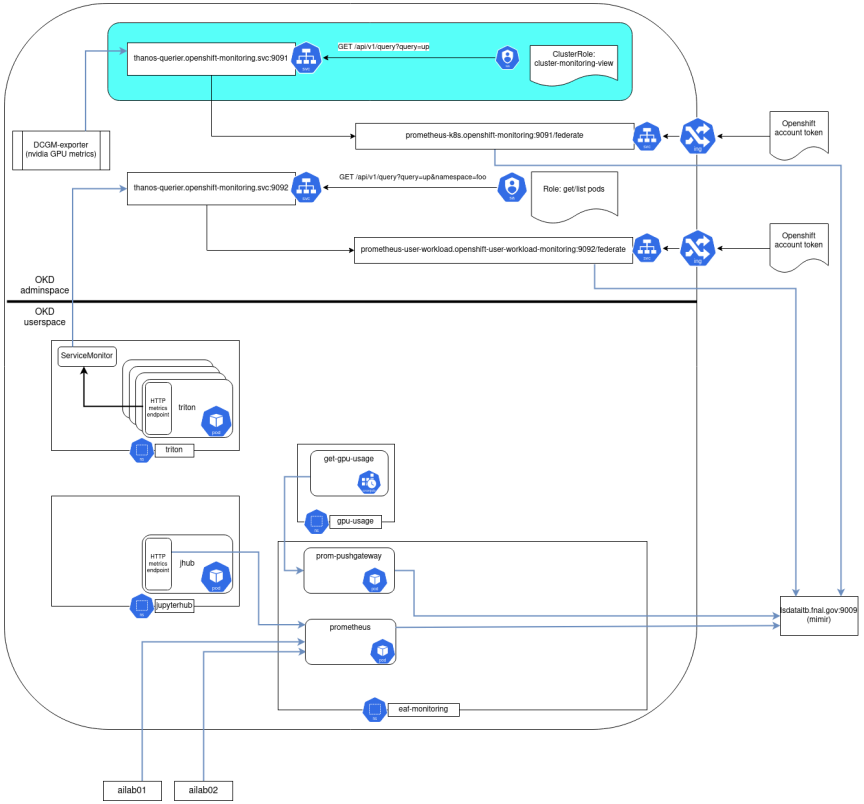


Fermilab

Overall Monitoring Diagram (don't worry, we'll zoom in)

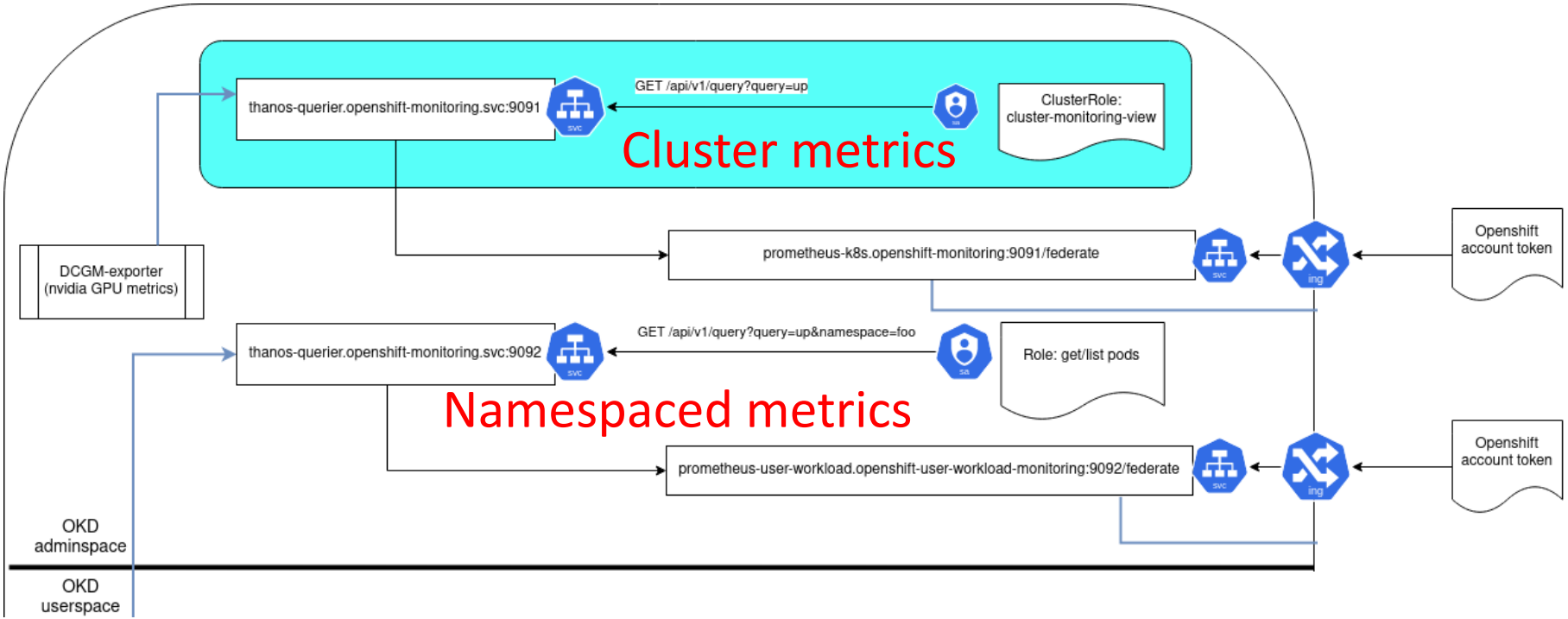
Infrastructure (OKD)

Applications



Fermilab

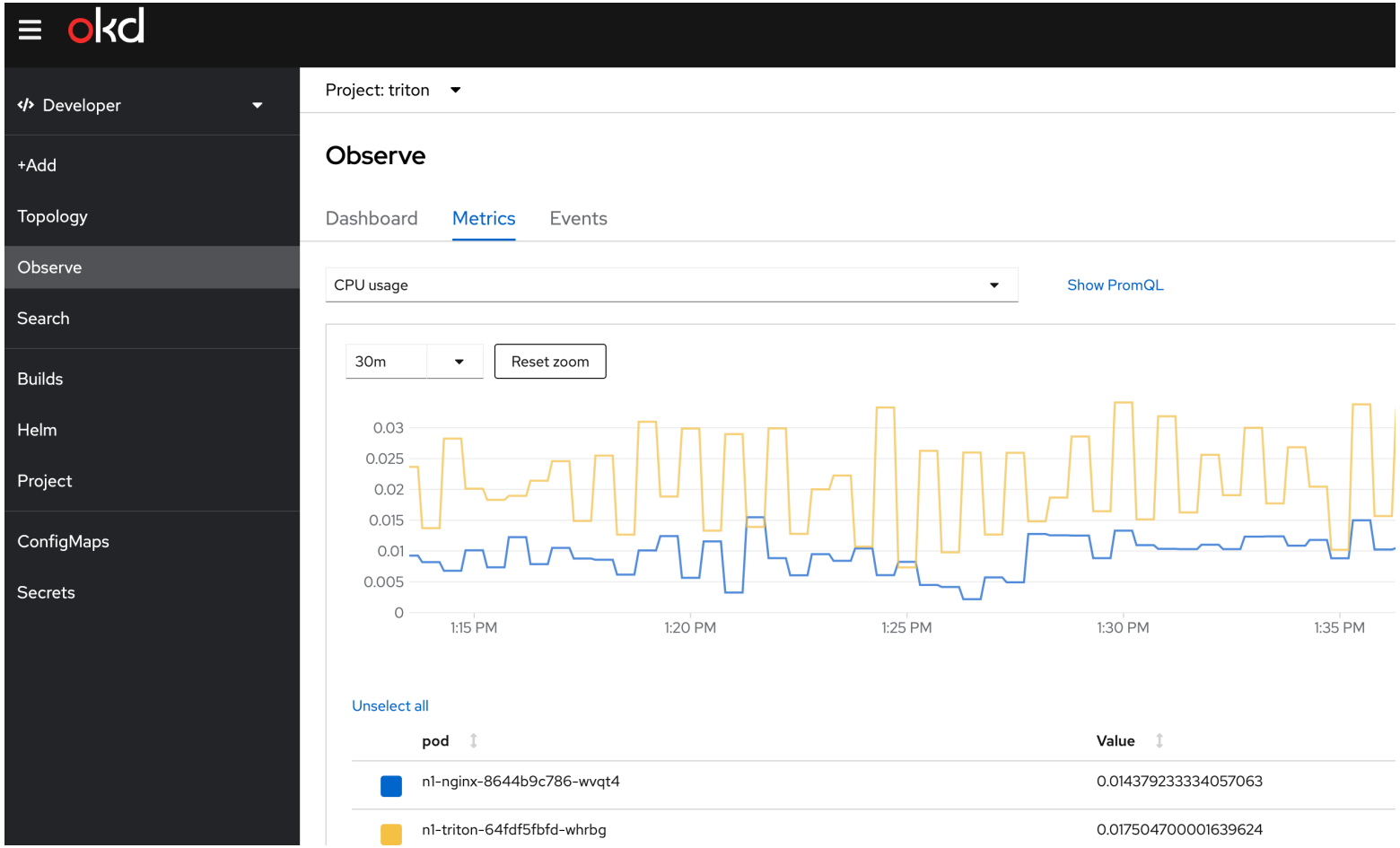
Monitoring in OKD



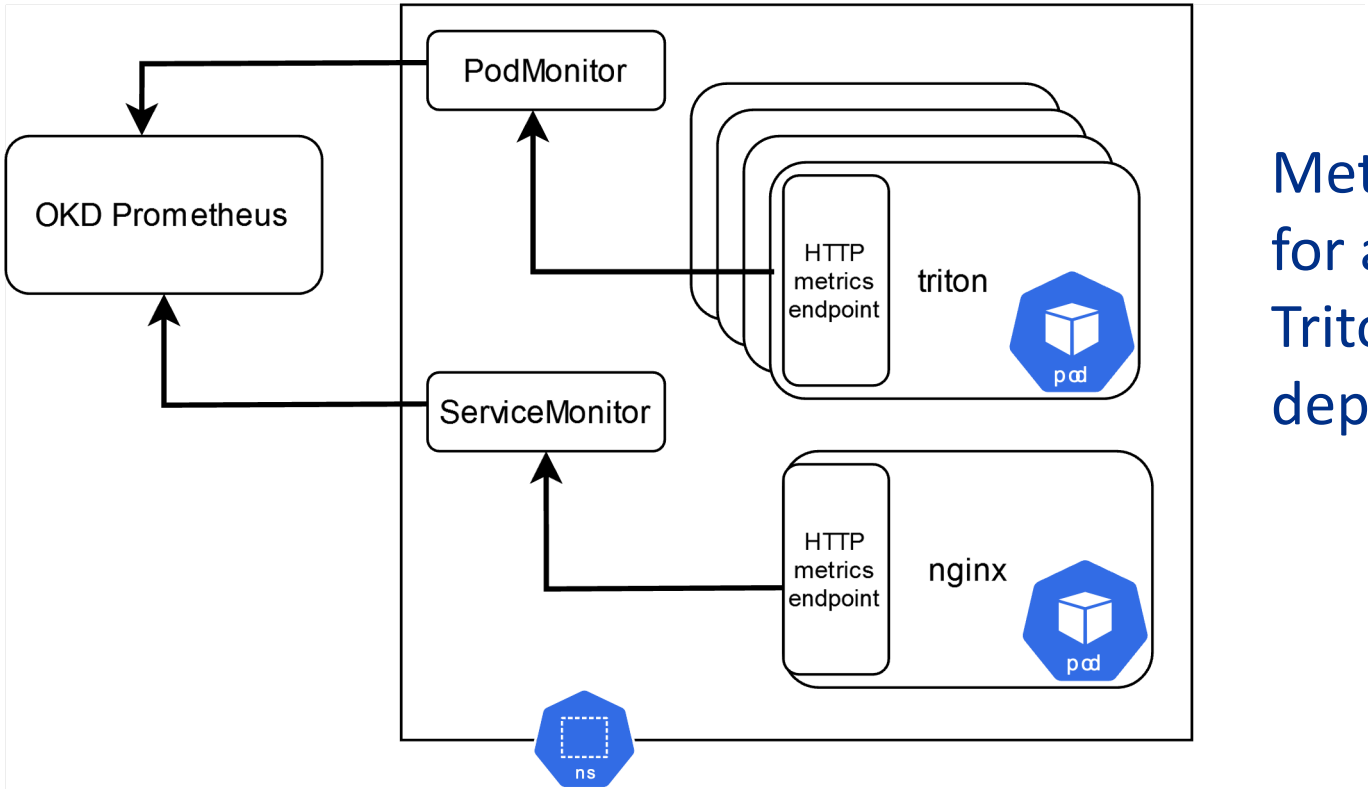
Fermilab

Monitoring in OKD (using OKD services)

- Built-in Prometheus server (with Thanos for HA)
  - Cluster-wide metrics
  - Project-level metrics via `podmonitor` and `servicemonitor` objects
  - Alerting rules, etc.
  - Short (~2 week) retention
  - Exporting to landscape (mimir)
- Pod logging
  - Exports to landscape (unless `metadata.labels.exportLogs` is set to `false`)
  - But not in strict time order
- Built-in dashboards (see next slide)

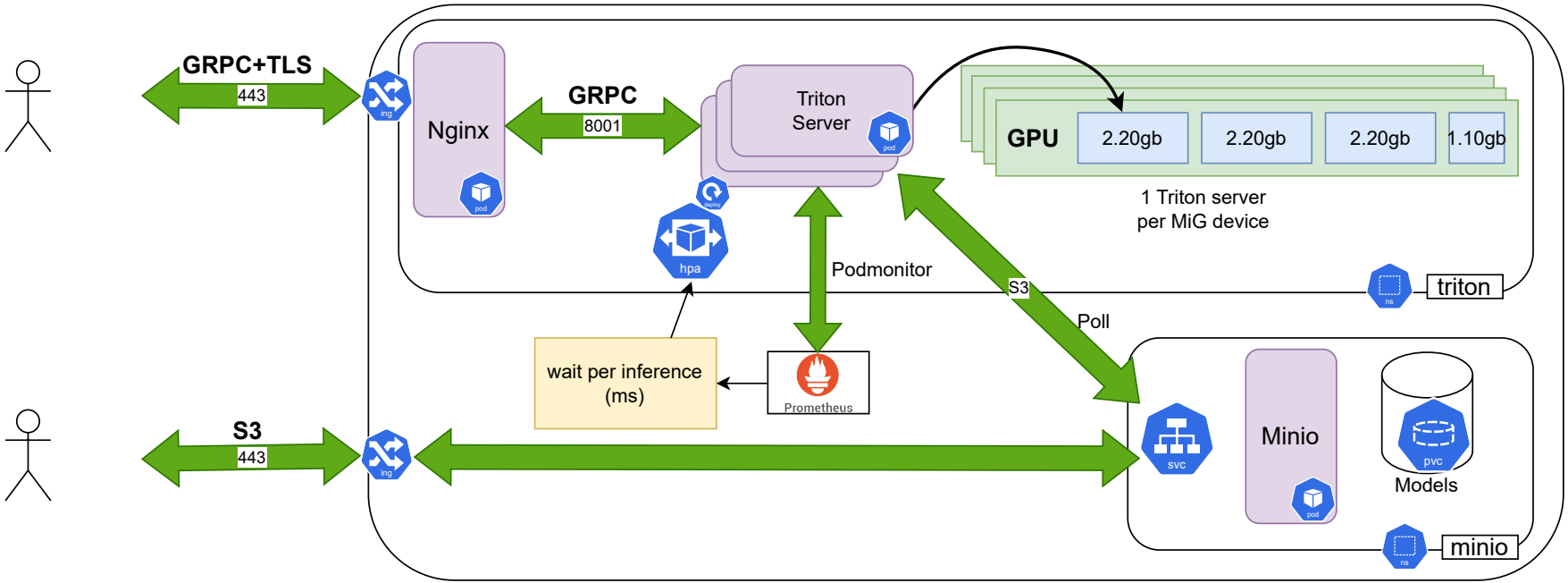

okd
Developer
+Add
Topology
Observe
Search
Builds
Helm
Project
ConfigMaps
Secrets
Project: triton
Observe
Dashboard
Metrics
Events
CPU usage
Show PromQL
30m
Reset zoom
0.03
0.025
0.02
0.015
0.01
0.005
0
1:15 PM
1:20 PM
1:25 PM
1:30 PM
1:35 PM
Unselect all
pod
Value
n1-nginx-8644b9c786-wvqt4
0.014379233334057063
n1-triton-64fdf5fbfd-whrbg
0.017504700001639624

Example: Using OKD monitoring with Triton application

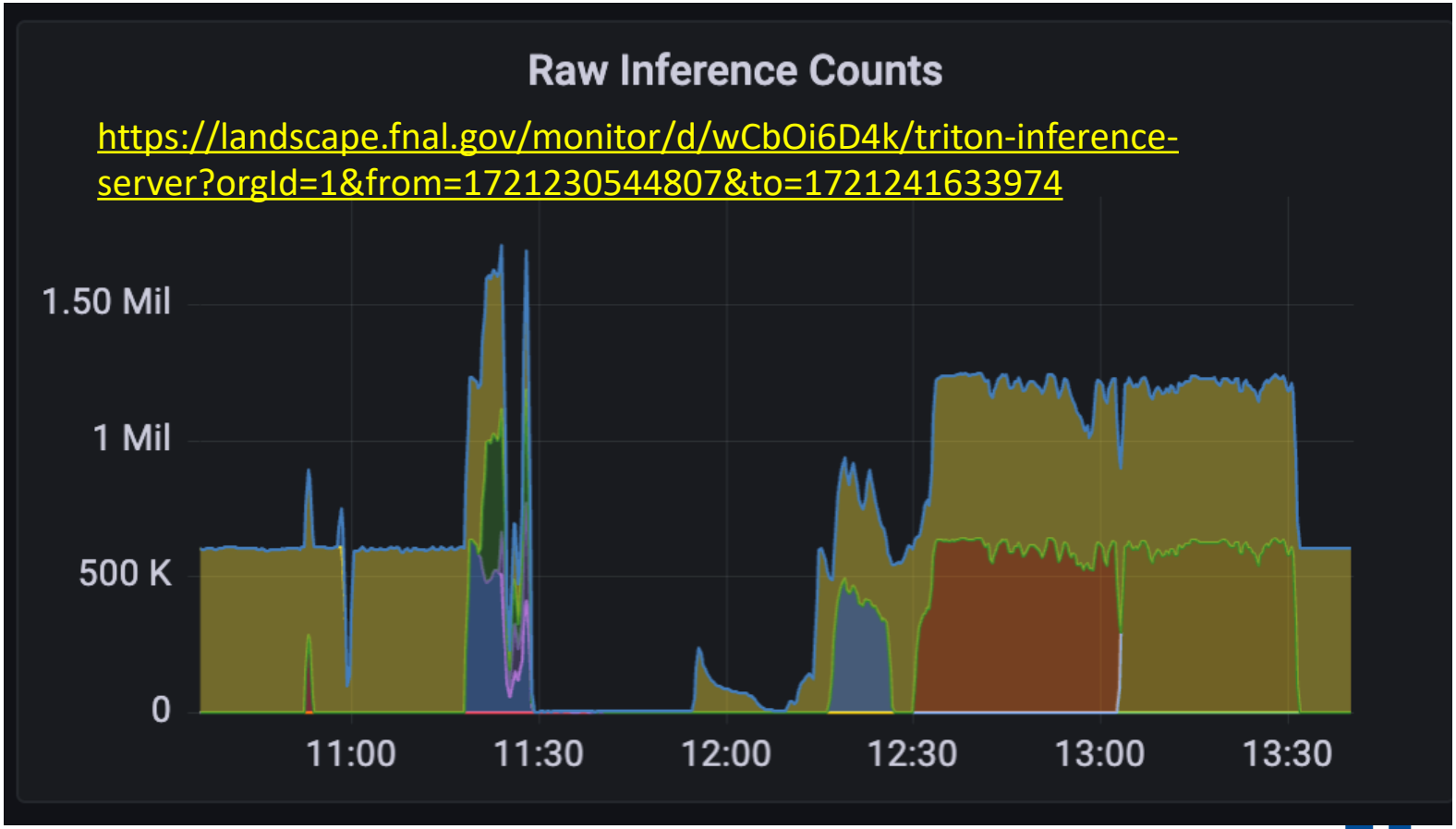


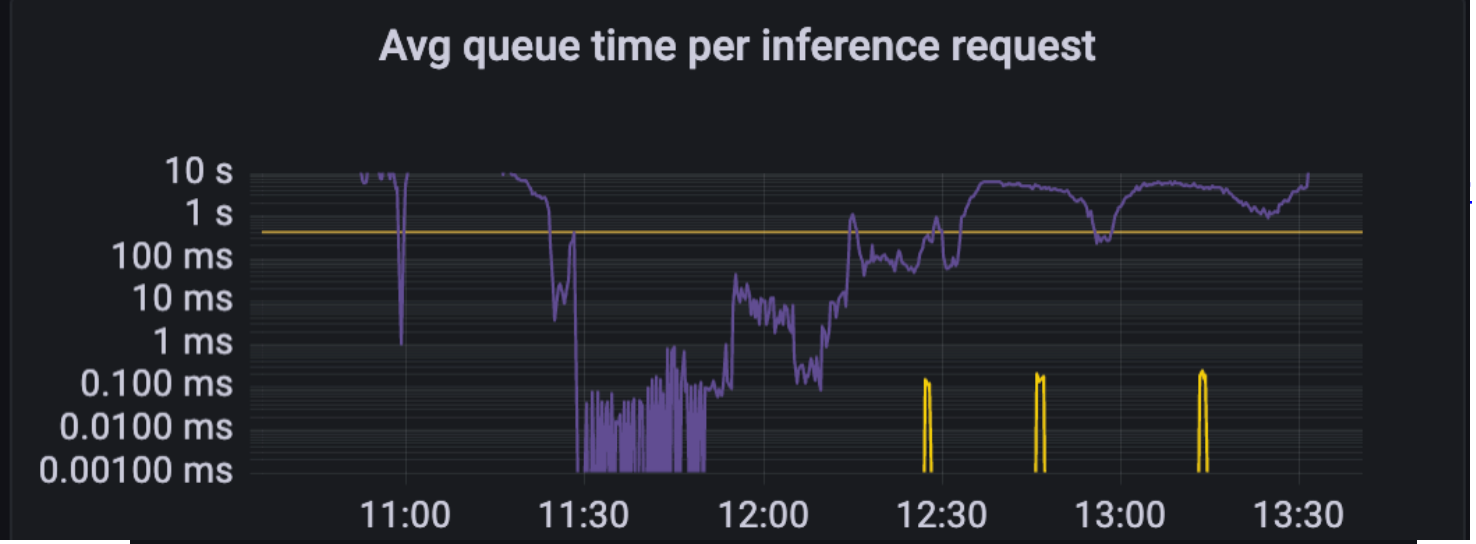
Metrics are used for auto-scaling Triton inference server deployment

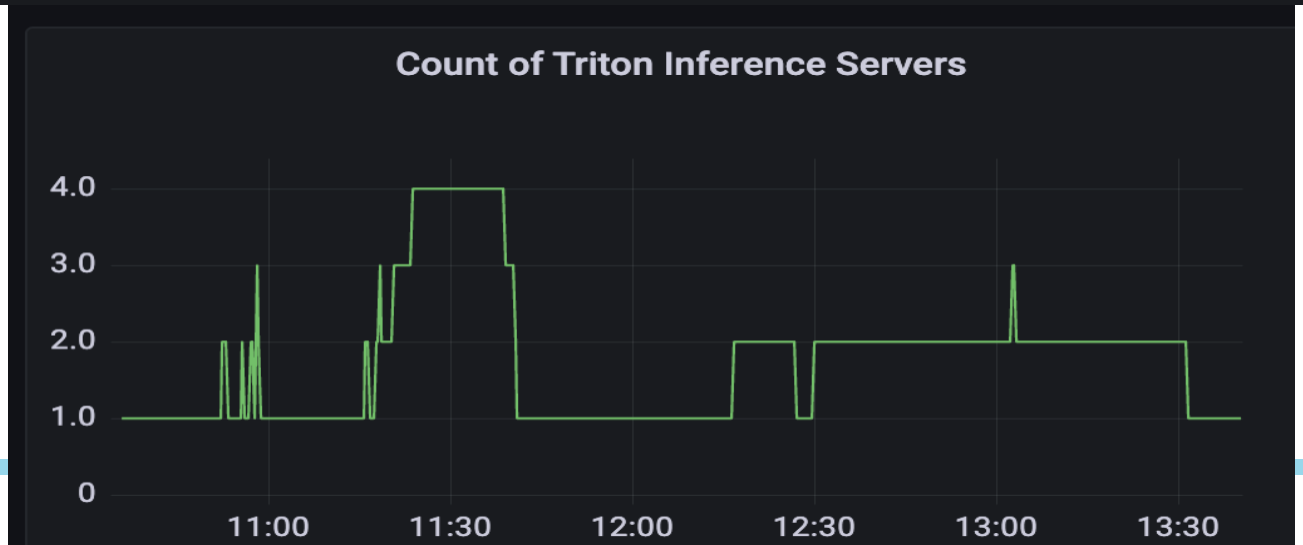
Triton Autoscaling



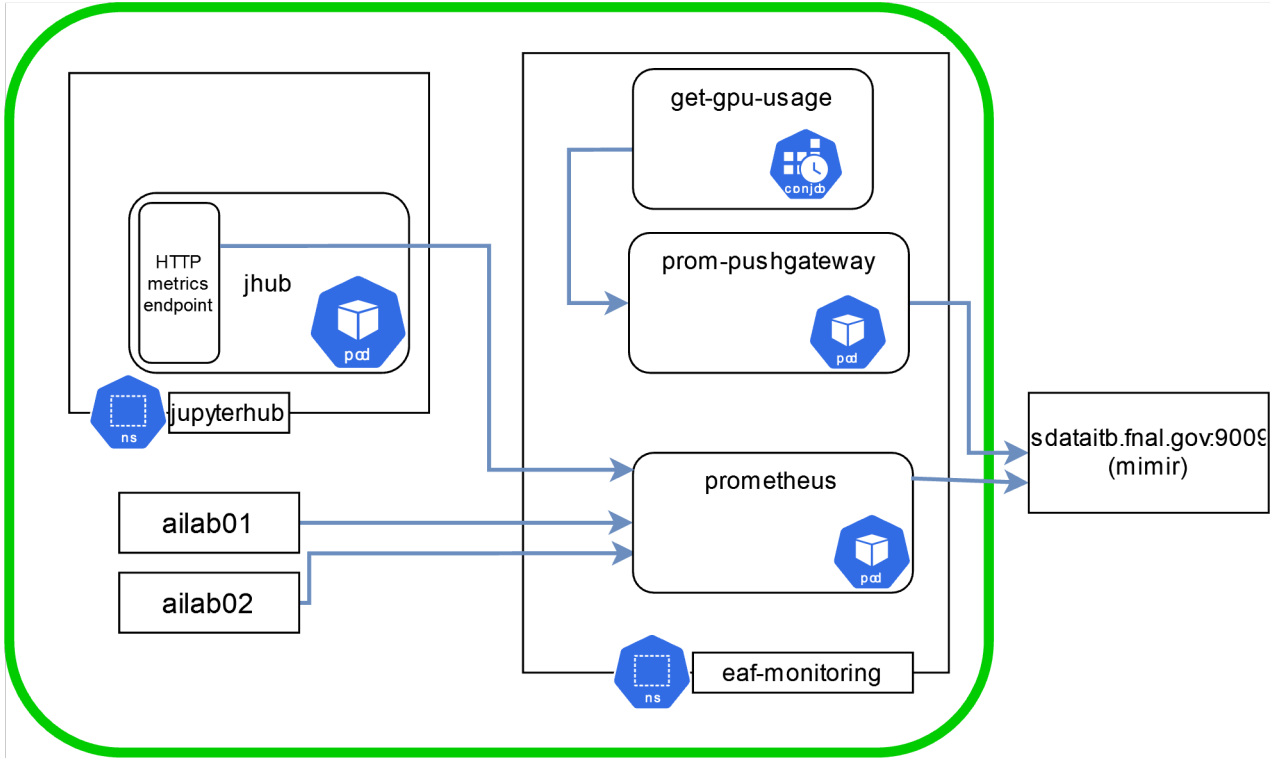
Fermilab


Raw Inference Counts
https://landscape.fnal.gov/monitor/d/wCbOi6D4k/triton-inference-server?orgId=1&from=1721230544807&to=1721241633974
1.50 Mil
1 Mil
500 K
0
11:00
11:30
12:00
12:30
13:00
13:30

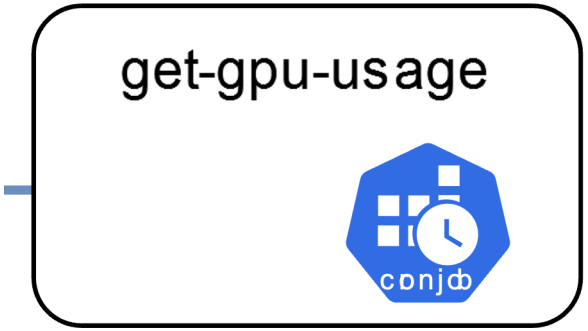

Avg queue time per inference request
10 s
1 s
100 ms
10 ms
1 ms
0.100 ms
0.0100 ms
0.00100 ms
11:00
11:30
12:00
12:30
13:00
13:30


Count of Triton Inference Servers
4.0
3.0
2.0
1.0
0
11:00
11:30
12:00
12:30
13:00
13:30

Application monitoring with our own Prometheus



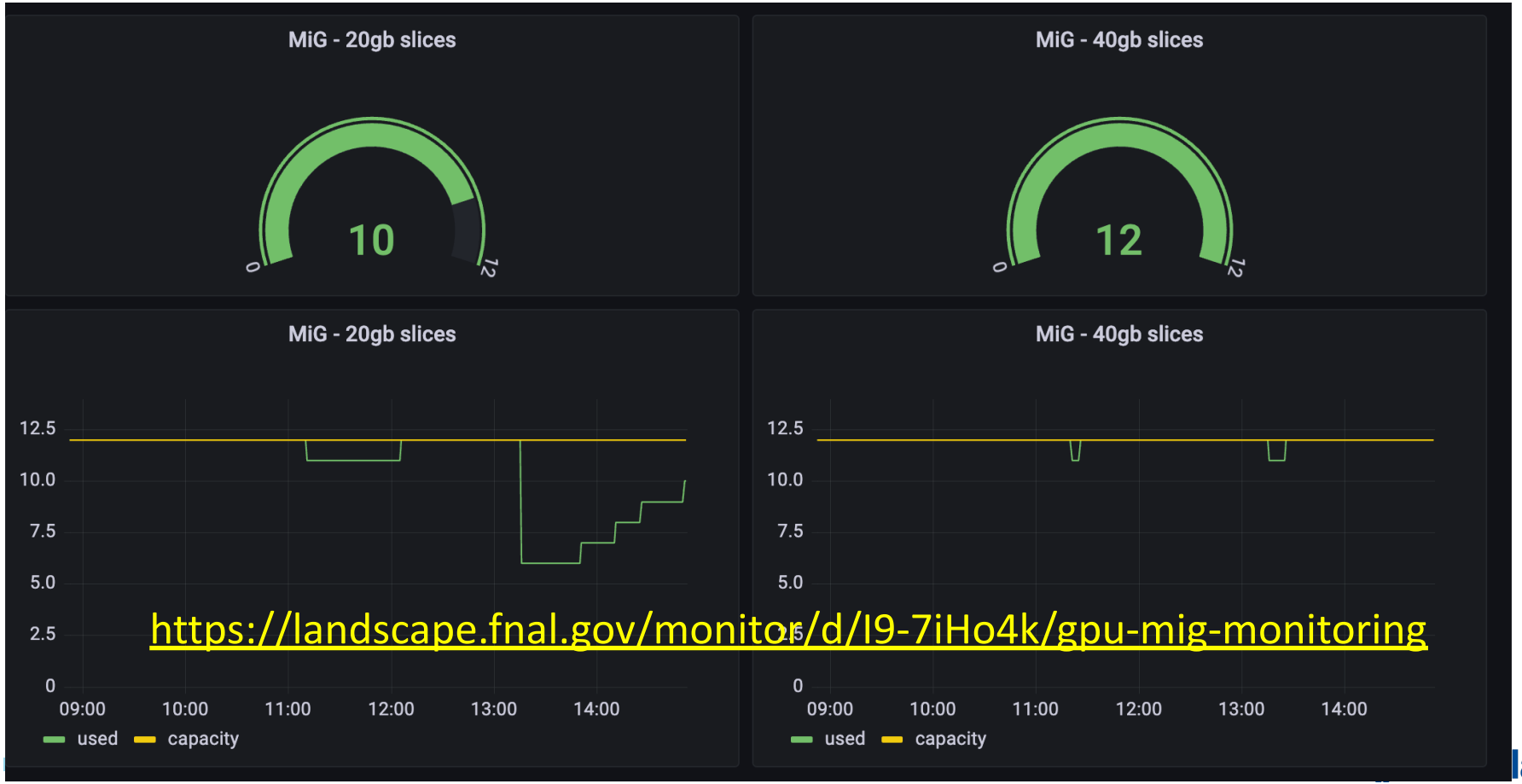
Application monitoring with our own Prometheus



Cronjob that queries GPU availability, usage

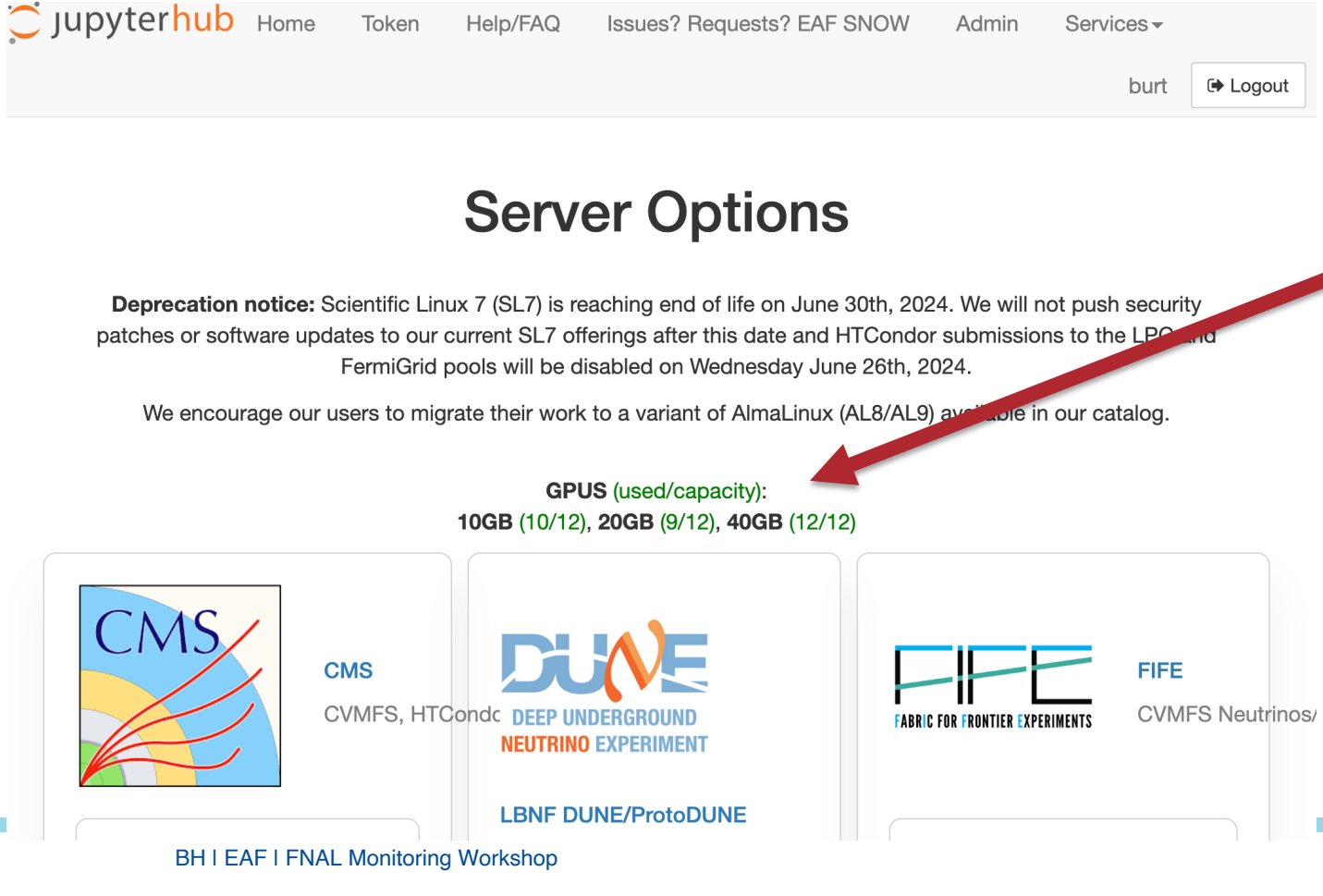
Pushes metrics to a prometheus push-gateway

Application monitoring with our own Prometheus



https://landscape.fnal.gov/monitor/d/l9-7iHo4k/gpu-mig-monitoring

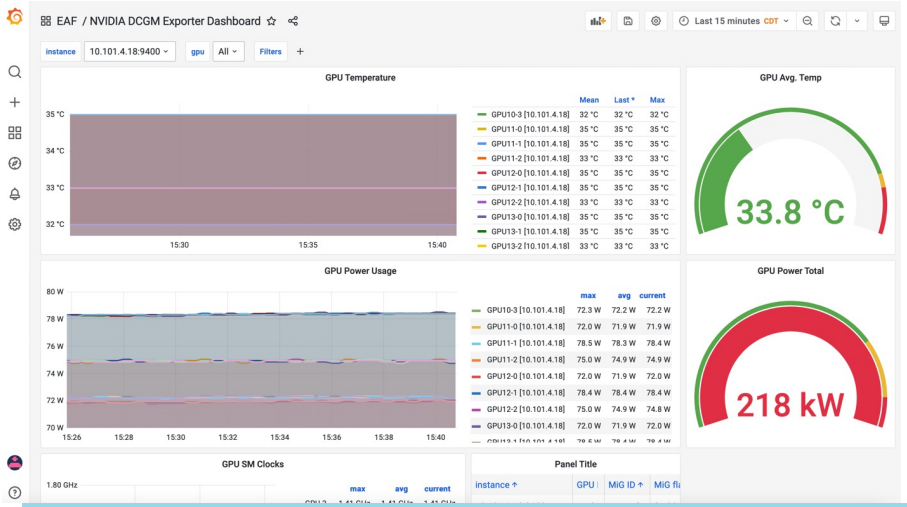
Also metrics are consumed (from landscape) by initcontainer

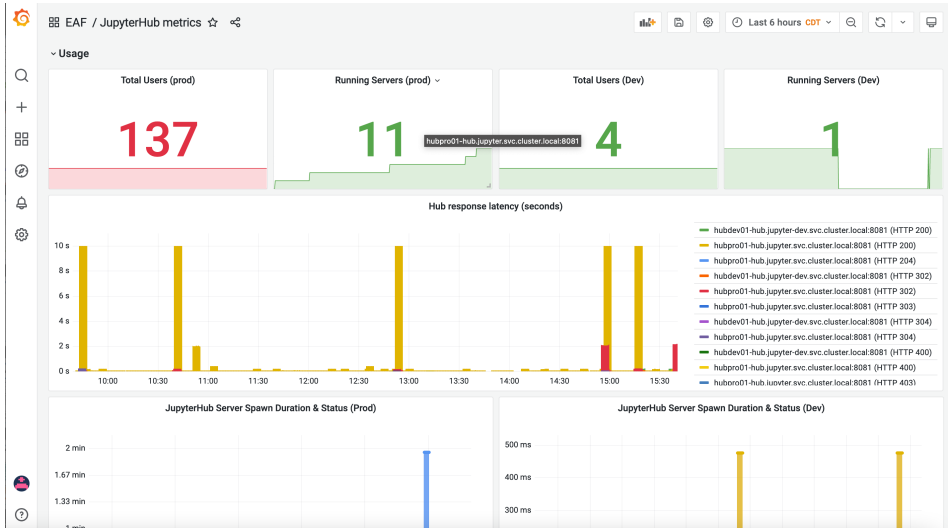


Monitoring and metrics

https://landscape.fnal.gov/monitor/dashboards/f/kngVRjPVz/eaf

- Grafana + Prometheus + InfluxDB monitoring hosted at FNAL Landscape.
- GPU statistics, CPU/Memory usage, network usage per notebook, JupyterHub metrics, TritonRT inference dashboards
- Having trouble on EAF? Check the status page, JupyterHub may be having trouble! (Hint: look for spikes in 400 or 500 HTTP errors)





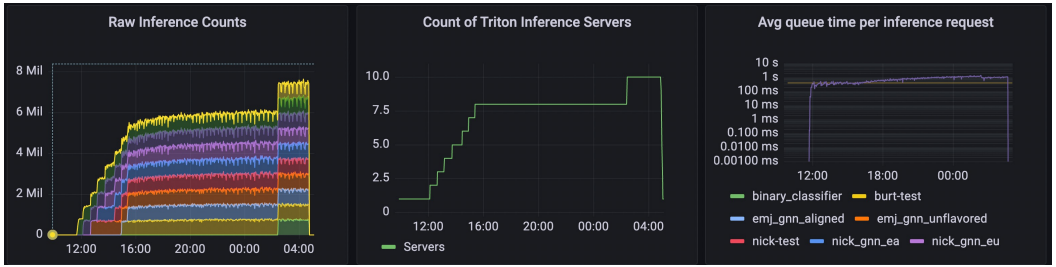
Fermilab

Monitoring and metrics

Insight on spawning process duration and outcomes for each step: poll, spawn, stop:

Spawning process duration and Hub (application) startup time:

Triton autoscaling:



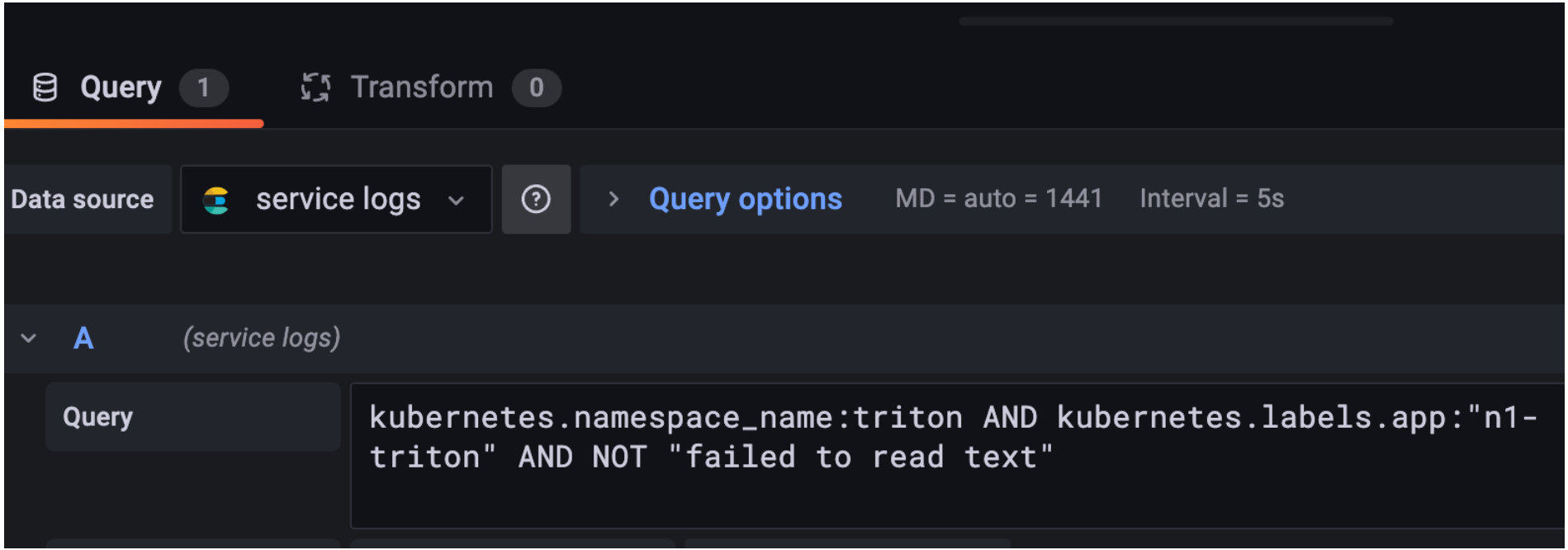
Pod logging

- Most services send log out put to stdout/stderr
- Application providers can retrieve from OKD with

  ```
  oc logs <podname>
  ```

- This is great, except if:
  - You are an end user (no access to kubernetes) rather than an application provider
  - Pod has been removed – no native log retention
- Logs available via Kibana and Grafana (through elasticsearch connector)

Pod logging


Query 1
Transform 0
Data source
service logs
Query options
MD = auto = 1441
Interval = 5s
A
(service logs)
Query
kubernetes.namespace_name:triton AND kubernetes.labels.app:"n1-triton" AND NOT "failed to read text"

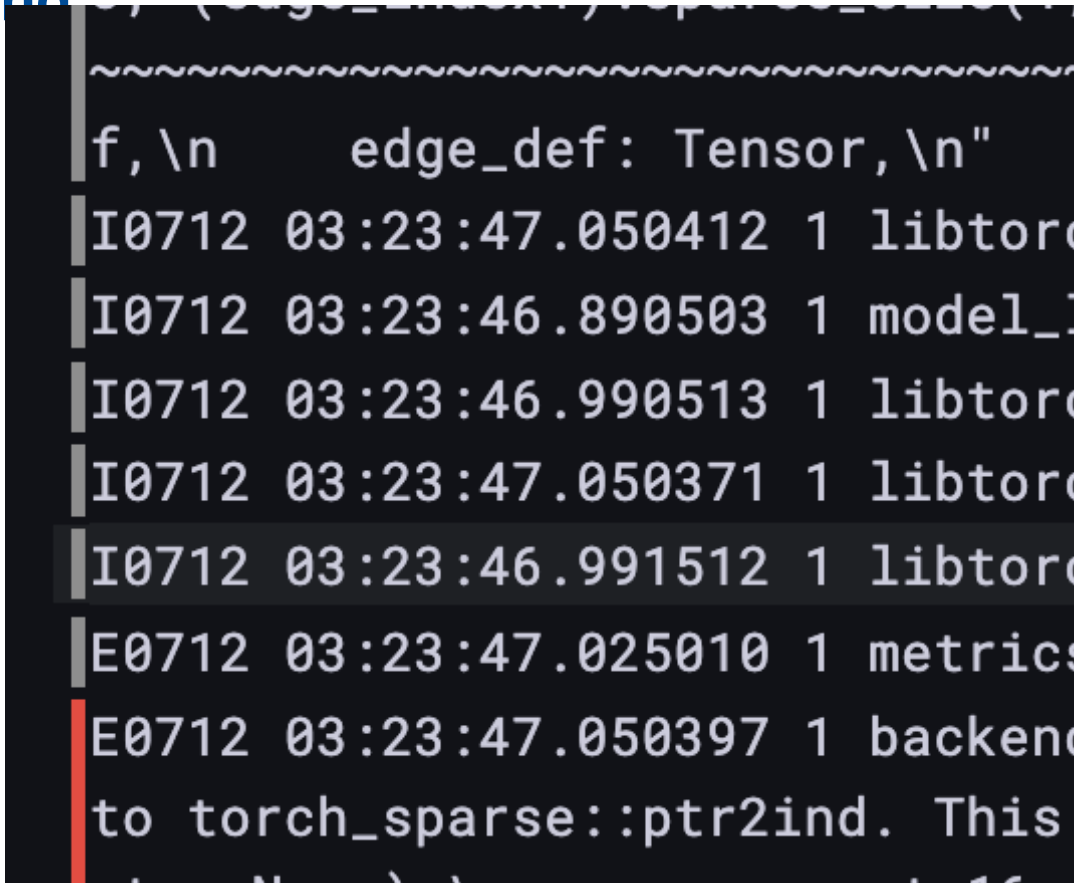
Pod logging

```
e-packages/torch_sparse/storage.py\", line 130\n        if not is_sorted:\n            idx = self._col.new_zeros(self._col.numel() + 1)\n            idx[1:] = self.row()\n
~~~~~~~~ <--- HERE\n            idx[1:] *= self._sparse_sizes[1]\n            idx[1:] += self._col\nSerialized   File \"code/__torch__/torch_sparse/storage.py\", line 277\n      _30 = torch.a
dd(torch.numel(self._col), 1)\n      idx = torch.new_zeros(_29, [_30], dtype=None, layout=None, device=None, pin_memory=None)\n      _31 = (self).row()\n      ~~~~~~~~~~~~~~~~~ <--- HERE\n
_32 = torch.slice(idx, 0, 1, 9223372036854775807, 1)\n      _33 = torch.copy_(_32, _31, False)\n'SparseStorage.__init__' is being compiled since it was called from 'GATConvJittable_3095bf.__c
heck_input____1'\nSerialized   File \"code/__torch__/GATConvJittable_3095bf.py\", line 49\n      pass\n    return the_size\n  def __check_input____1(self: __torch__.GATConvJittable_3095bf.GAT
ConvJittable_3095bf,\n      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    edge_index: __torch__.torch_sparse.tensor.SparseTensor,\n    ~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    size: Optional[Tuple[int, int]]) -> List[Optional[int]]:\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    the_size = annotate(L
ist[Optional[int]], [None, None])\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    edge_index1 = unchecked_cast(__torch__.torch_sparse.tensor.SparseTensor, edge_index)\n    ~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    _4 = torch.eq(self.flow, \"target_to_source\")\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    i
f _4:\n    ~~~~~~\n      ops.prim.RaiseException(\"Exception\")\n      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    else:\n    ~~~~~\n      pass\n      ~~~~\n    _5 = torch._set_item(the_size,
0, (edge_index1).sparse_size(1, ))\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    _6 = torch._set_item(the_size, 1, (edge_index1).sparse_size(0, ))\n    ~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    return the_size\n    ~~~~~~~~~~~~~~~ <--- HERE\n  def __collect____0(self: __torch__.GATConvJittable_3095bf.GATConvJittable_3095b
f,\n    edge_def: Tensor,\n"
I0712 03:23:47.050412 1 libtorch.cc:2593] "TRITONBACKEND_ModelFinalize: delete model state"
I0712 03:23:46.890503 1 model_lifecycle.cc:472] "loading: gat_test:1"
I0712 03:23:46.990513 1 libtorch.cc:2570] "TRITONBACKEND_ModelInitialize: gat_test (version 1)"
I0712 03:23:47.050371 1 libtorch.cc:2648] "TRITONBACKEND_ModelInstanceFinalize: delete instance state"
I0712 03:23:46.991512 1 libtorch.cc:389] "Inference Mode is enabled for model instance 'gat_test'"
E0712 03:23:47.025010 1 metrics.cc:1015] "Unable to get device UUID: Bad parameter passed to function"
E0712 03:23:47.050397 1 backend_model.cc:692] "ERROR: Failed to create instance: failed to load model 'gat_test': \nUnknown builtin op: torch_sparse::ptr2ind.\nCould not find any similar ops
to torch_sparse::ptr2ind. This op may not exist or may not be currently supported in TorchScript.\n:\n  File \"code/__torch__/torch_sparse/storage.py\", line 822\n      if torch.__isnot__(row
ptr, None):\n        rowptr16 = unchecked_cast(Tensor, rowptr)\n        row18 = ops.torch_sparse.ptr2ind(rowptr16, torch.numel(self._col))\n                ~~~~~~~~~~~~~~~~~~~~~~~ <--- HERE
\n        self._row = row18\n        _134 = row18\n'SparseStorage.row' is being compiled since it was called from 'SparseStorage.__init__'\n  File \"/opt/conda/lib/python3.6/site-packages/tor
ch_sparse/storage.py\", line 130\n        if not is_sorted:\n            idx = self._col.new_zeros(self._col.numel() + 1)\n            idx[1:] = self.row()\n            ~~~~~~~~ <--
- HERE\n            idx[1:] *= self._sparse_sizes[1]\n            idx[1:] += self._col\nSerialized   File \"code/__torch__/torch_sparse/storage.py\", line 277\n      _30 = torch.add(torch.num
el(self._col), 1)\n      idx = torch.new_zeros(_29, [_30], dtype=None, layout=None, device=None, pin_memory=None)\n      _31 = (self).row()\n      ~~~~~~~~~~~~~~~~~ <--- HERE\n      _32 = tor
ch.slice(idx, 0, 1, 9223372036854775807, 1)\n      _33 = torch.copy_(_32, _31, False)\n'SparseStorage.__init__' is being compiled since it was called from 'GATConvJittable_3095bf.__check_inpu
t____1'\nSerialized   File \"code/__torch__/GATConvJittable_3095bf.py\", line 49\n      pass\n    return the_size\n  def __check_input____1(self: __torch__.GATConvJittable_3095bf.GATConvJitta
ble_3095bf,\n      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    edge_index: __torch__.torch_sparse.tensor.SparseTensor,\n    ~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    size: Optional[Tuple[int, int]]) -> List[Optional[int]]:\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    the_size = annotate(List[Optio
nal[int]], [None, None])\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    edge_index1 = unchecked_cast(__torch__.torch_sparse.tensor.SparseTensor, edge_index)\n    ~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    _4 = torch.eq(self.flow, \"target_to_source\")\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    if _4:\n
```

Pod logging

```
e-packages/torch_sparse/storage.py\", line 130\n        if not is_sorted:\n            idx = self._col.new_zeros(self._col.numel() + 1)\n            idx[1:] = self.row()\n
~~~~~~~~ <--- HERE\n            idx[1:] *= self._sparse_sizes[1]\n            idx[1:] += self._col\nSerialized   File \"code/__torch__/torch_sparse/storage.py\", line 277\n      _30 = torch.a
dd(torch.numel(self._col), 1)\n      idx = torch.new_zeros(_29, [_30], dtype=None, layout=None, device=None, pin_memory=None)\n      _31 = (self).row()\n      ~~~~~~~~~~~~~~~~~ <--- HERE\n
_32 = torch.slice(idx, 0, 1, 9223372036854775807, 1)\n      _33 = torch.copy_(_32, _31, False)\n'SparseStorage.__init__' is being compiled since it was called from 'GATConvJittable_3095bf.__c
heck_input____1'\nSerialized   File \"code/__torch__/GATConvJittable_3095bf.py\", line 49\n      pass\n    return the_size\n  def __check_input____1(self: __torch__.GATConvJittable_3095bf.GAT
ConvJittable_3095bf,\n      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    edge_index: __torch__.torch_sparse.tensor.SparseTensor,\n    ~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    size: Optional[Tuple[int, int]]) -> List[Optional[int]]:\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    the_size = annotate(L
ist[Optional[ [illegible] None])\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    edge_index1 = unchecked_cast(__torch__.torch_sparse.tensor.SparseTensor, edge_index)\n    ~~
~~~~ [illegible] ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    _4 = torch.eq(self.flow, \"target_to_source\")\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    i
f ~~~~~~\n      op [illegible] RaiseException(\"Exception\")\n      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    else:\n    ~~~~~\n      pass\n      ~~~~\n    _5 = torch._set_item(the_size,
 [illegible] dge_index1).sparse_size(1, [illegible] ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    _6 = torch._set_item(the_size, 1, (edge_index1).sparse_size(0, ))\n    ~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ [illegible] ~~~~~~~~~~~~~~~~~~~\n    return the_size\n    ~~~~~~~~~~~~~~~ <--- HERE\n  def __collect____0(self: __torch__.GATConvJittable_3095bf.GATConvJittable_3095b
 [illegible] ,\n    edge_def: Tensor,\n"
I0712 03:23:47.050412 1 libtorch.cc: [illegible] ] "TRITONBACKEND_ModelFinalize: delete model state"
I0712 03:23:46.890503 1 model_lifecyc [illegible] :472] "loading: gat_test:1"
I0712 03:23:46.990513 1 libtorch.cc:25 [illegible] "TRITONBACKEND_ModelInitialize: gat_test (version 1)"
I0712 03:23:47.050371 1 libtorch.cc:264 [illegible] "TRITONBACKEND_ModelInstanceFinalize: delete instance state"
I0712 03:23:46.991512 1 libtorch.cc:389 [illegible] Inference Mode is enabled for model instance 'gat_test'"
E0712 03:23:47.025010 1 metrics.cc:101 [illegible] Unable to get device UUID: Bad parameter passed to function"
E0712 03:23:47.050397 1 backend_model. [illegible] 92] "ERROR: Failed to create instance: failed to load model 'gat_test': \nUnknown builtin op: torch_sparse::ptr2ind.\nCould not find any similar ops
to torch_sparse::ptr2ind. This op may [illegible] exist or may not be currently supported in TorchScript.\n:\n  File \"code/__torch__/torch_sparse/storage.py\", line 822\n      if torch.__isnot__(row
ptr, None):\n        rowptr16 = unch [illegible] d_cast(Tensor, rowptr)\n        row18 = ops.torch_sparse.ptr2ind(rowptr16, torch.numel(self._col))\n        ~~~~~~~~~~~~~~~~~~~~~~~ <--- HERE
 [illegible] n        self._row = row18\n [illegible] 34 = row18\n'SparseStorage.row' is being compiled since it was called from 'SparseStorage.__init__'\n  File \"/opt/conda/lib/python3.6/site-packages/tor
 [illegible] sparse/storage.py\", line 130 [illegible] if not is_sorted:\n            idx = self._col.new_zeros(self._col.numel() + 1)\n            idx[1:] = self.row()\n            ~~~~~~~~ <--
 [illegible] \n            idx[1:] * [illegible] ._sparse_sizes[1]\n            idx[1:] += self._col\nSerialized   File \"code/__torch__/torch_sparse/storage.py\", line 277\n      _30 = torch.add(torch.num
el( [illegible] 1), 1)\n [illegible] orch.new_zeros(_29, [_30], dtype=None, layout=None, device=None, pin_memory=None)\n      _31 = (self).row()\n      ~~~~~~~~~~~~~~~~~ <--- HERE\n      _32 = tor
ch.slice( [illegible] 6854775807, 1)\n      _33 = torch.copy_(_32, _31, False)\n'SparseStorage.__init__' is being compiled since it was called from 'GATConvJittable_3095bf.__check_inpu
t____1'\nSerial [illegible] File \"code/__torch__/GATConvJittable_3095bf.py\", line 49\n      pass\n    return the_size\n  def __check_input____1(self: __torch__.GATConvJittable_3095bf.GATConvJitta
ble_3095bf,\n      ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    edge_index: __torch__.torch_sparse.tensor.SparseTensor,\n    ~~~~~~~~~~~~~~~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    size: Optional[Tuple[int, int]]) -> List[Optional[int]]:\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    the_size = annotate(List[Optio
nal[int]], [None, None])\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    edge_index1 = unchecked_cast(__torch__.torch_sparse.tensor.SparseTensor, edge_index)\n    ~~~~~~~~~~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    _4 = torch.eq(self.flow, \"target_to_source\")\n    ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n    if _4:\n
```

Pod logging


f,\n edge_def: Tensor,\n"
I0712 03:23:47.050412 1 libtor
I0712 03:23:46.890503 1 model_
I0712 03:23:46.990513 1 libtor
I0712 03:23:47.050371 1 libtor
I0712 03:23:46.991512 1 libtor
E0712 03:23:47.025010 1 metric
E0712 03:23:47.050397 1 backen
to torch_sparse::ptr2ind. This