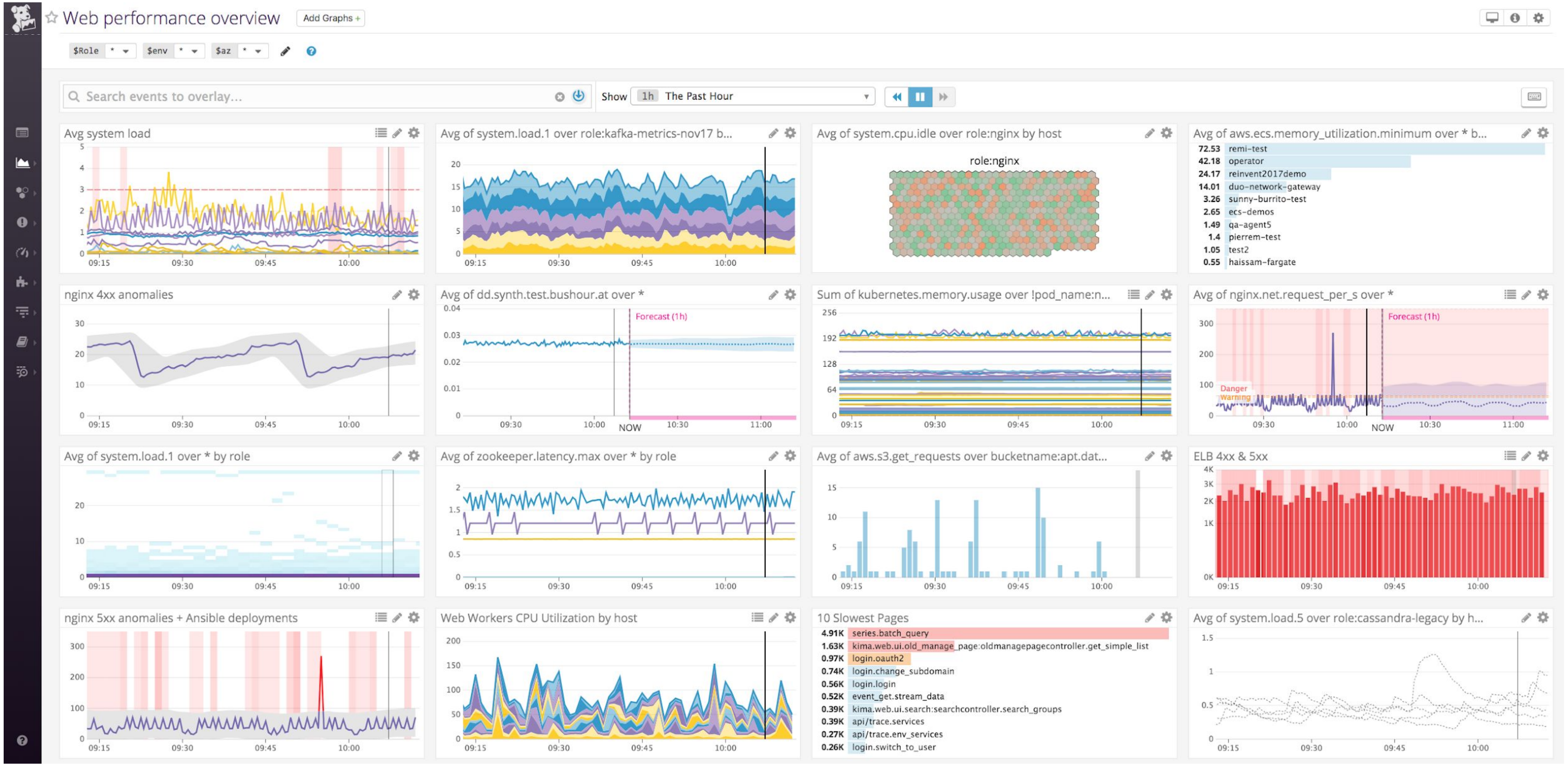
Building highly reliable data pipelines @ Datadog

Quentin FRANCOIS

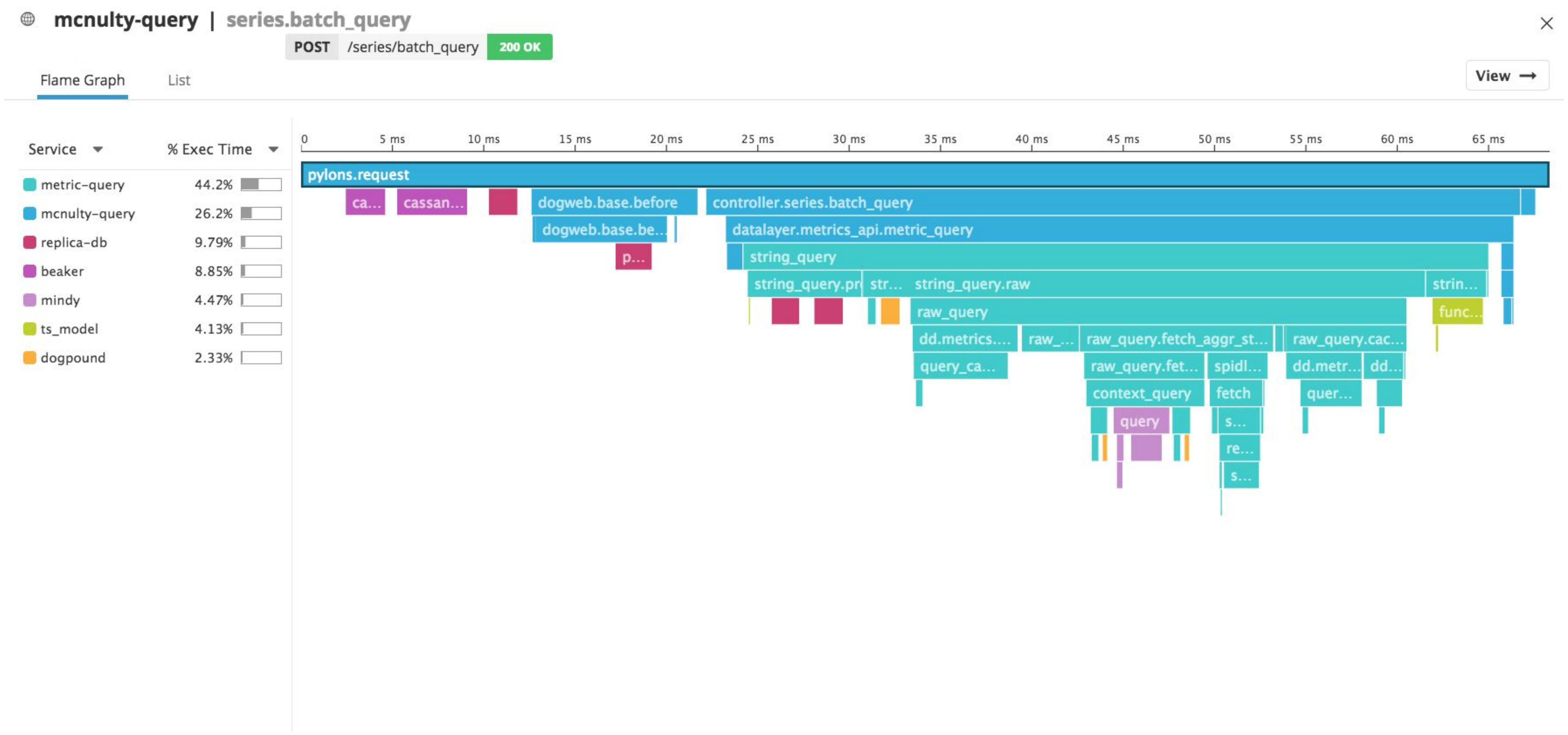
Team Lead, Data Engineering

DataEng Barcelona ‘18


Web performance overview
Add Graphs +
$Role *
$env *
$az *
Search events to overlay...
Show 1h The Past Hour
Avg system load
Avg of system.load.1 over role:kafka-metrics-nov17 b...
Avg of system.cpu.idle over role:nginx by host
role:nginx
Avg of aws.ecs.memory_utilization.minimum over * b...
nginx 4xx anomalies
Avg of dd.synth.test.bushour.at over *
Forecast (1h)
NOW
Sum of kubernetes.memory.usage over !pod_name:n...
Avg of nginx.net.request_per_s over *
Danger
Warning
Avg of system.load.1 over * by role
Avg of zookeeper.latency.max over * by role
Avg of aws.s3.get_requests over bucketname:apt.dat...
ELB 4xx & 5xx
nginx 5xx anomalies + Ansible deployments
Web Workers CPU Utilization by host
10 Slowest Pages
Avg of system.load.5 over role:cassandra-legacy by h...

72.53	remi-test
42.18	operator
24.17	reinvent2017demo
14.01	duo-network-gateway
3.26	sunny-burrito-test
2.65	ecs-demos
1.49	qa-agent5
1.4	pierrem-test
1.05	test2
0.55	haissam-fargate

4.91K	series.batch_query
1.63K	kima.web.ui.old_manage_page:oldmanagepagecontroller.get_simple_list
0.97K	login.oauth2
0.74K	login.change_subdomain
0.56K	login.login
0.52K	event_get.stream_data
0.39K	kima.web.ui.search:searchcontroller.search_groups
0.39K	api/trace.services
0.27K	api/trace.env_services
0.26K	login.switch_to_user

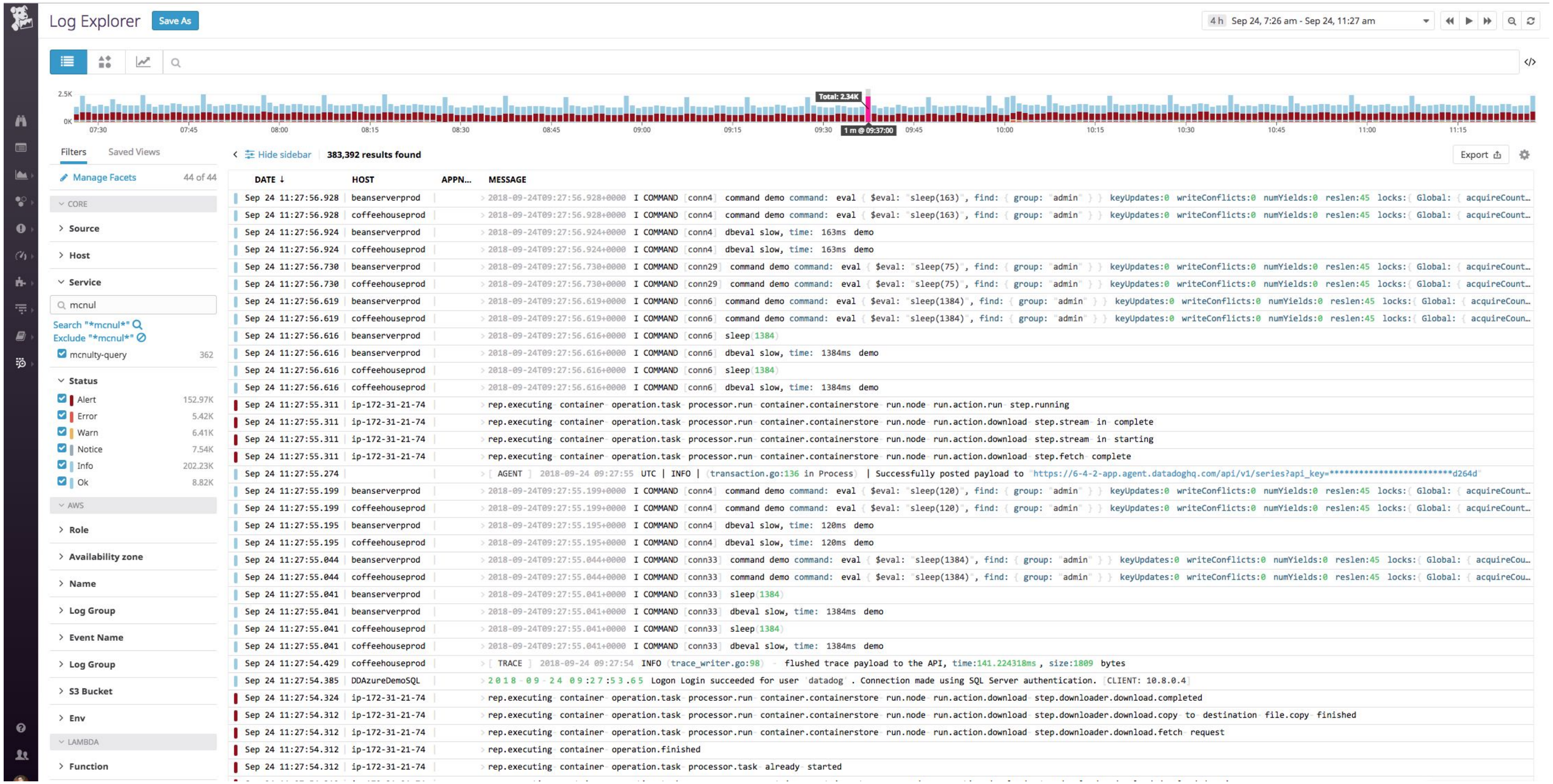


mcnulty-query | series.batch_query

POST /series/batch_query 200 OK

Flame Graph List View ⟶

Service	% Exec Time
metric-query	44.2%
mcnulty-query	26.2%
replica-db	9.79%
beaker	8.85%
mindy	4.47%
ts_model	4.13%
dogpound	2.33%



383,392 results found

44 of 44

CORE
- Source
- Host
- Service
  - mcnul
  - Search "*mcnul*"
  - Exclude "*mcnul*"
  - ✓ mcnulty-query 362
- Status
  - ✓ Alert 152.97K
  - ✓ Error 5.42K
  - ✓ Warn 6.41K
  - ✓ Notice 7.54K
  - ✓ Info 202.23K
  - ✓ Ok 8.82K

AWS
- Role
- Availability zone
- Name
- Log Group
- Event Name
- Log Group
- S3 Bucket
- Env

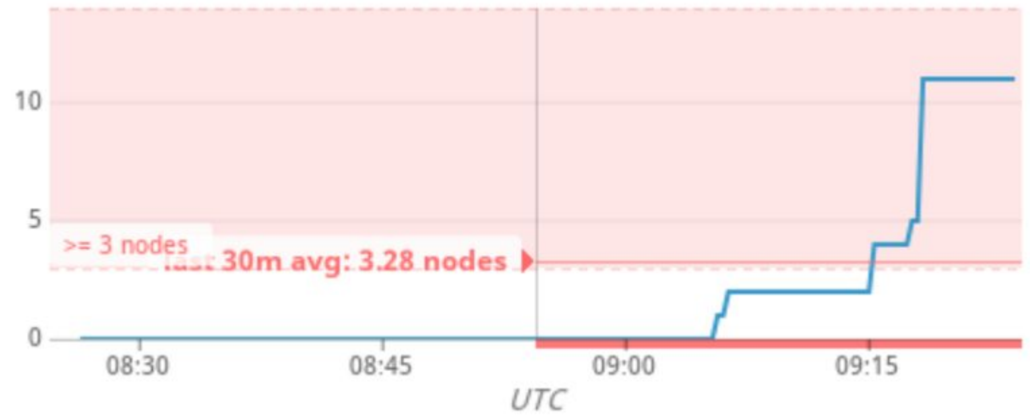
LAMBDA
- Function

```
DATE ↓                  HOST               APPN...  MESSAGE
Sep 24 11:27:56.928 | beanserverprod    |  > 2018-09-24T09:27:56.928+0000 I COMMAND [conn4] command demo command: eval { $eval: "sleep(163)", find: { group: "admin" } } keyUpdates:0 writeConflicts:0 numYields:0 reslen:45 locks:{ Global: { acquireCount…
Sep 24 11:27:56.928 | coffeehouseprod   |  > 2018-09-24T09:27:56.928+0000 I COMMAND [conn4] command demo command: eval { $eval: "sleep(163)", find: { group: "admin" } } keyUpdates:0 writeConflicts:0 numYields:0 reslen:45 locks:{ Global: { acquireCount…
Sep 24 11:27:56.924 | beanserverprod    |  > 2018-09-24T09:27:56.924+0000 I COMMAND [conn4] dbeval slow, time: 163ms demo
Sep 24 11:27:56.924 | coffeehouseprod   |  > 2018-09-24T09:27:56.924+0000 I COMMAND [conn4] dbeval slow, time: 163ms demo
Sep 24 11:27:56.730 | beanserverprod    |  > 2018-09-24T09:27:56.730+0000 I COMMAND [conn29] command demo command: eval { $eval: "sleep(75)", find: { group: "admin" } } keyUpdates:0 writeConflicts:0 numYields:0 reslen:45 locks:{ Global: { acquireCount…
Sep 24 11:27:56.730 | coffeehouseprod   |  > 2018-09-24T09:27:56.730+0000 I COMMAND [conn29] command demo command: eval { $eval: "sleep(75)", find: { group: "admin" } } keyUpdates:0 writeConflicts:0 numYields:0 reslen:45 locks:{ Global: { acquireCount…
Sep 24 11:27:56.619 | beanserverprod    |  > 2018-09-24T09:27:56.619+0000 I COMMAND [conn6] command demo command: eval { $eval: "sleep(1384)", find: { group: "admin" } } keyUpdates:0 writeConflicts:0 numYields:0 reslen:45 locks:{ Global: { acquireCoun…
Sep 24 11:27:56.619 | coffeehouseprod   |  > 2018-09-24T09:27:56.619+0000 I COMMAND [conn6] command demo command: eval { $eval: "sleep(1384)", find: { group: "admin" } } keyUpdates:0 writeConflicts:0 numYields:0 reslen:45 locks:{ Global: { acquireCoun…
Sep 24 11:27:56.616 | beanserverprod    |  > 2018-09-24T09:27:56.616+0000 I COMMAND [conn6] sleep(1384)
Sep 24 11:27:56.616 | beanserverprod    |  > 2018-09-24T09:27:56.616+0000 I COMMAND [conn6] dbeval slow, time: 1384ms demo
Sep 24 11:27:56.616 | coffeehouseprod   |  > 2018-09-24T09:27:56.616+0000 I COMMAND [conn6] sleep(1384)
Sep 24 11:27:56.616 | coffeehouseprod   |  > 2018-09-24T09:27:56.616+0000 I COMMAND [conn6] dbeval slow, time: 1384ms demo
Sep 24 11:27:55.311 | ip-172-31-21-74   |  > rep.executing container operation.task processor.run container.containerstore run.node run.action.run step.running
Sep 24 11:27:55.311 | ip-172-31-21-74   |  > rep.executing container operation.task processor.run container.containerstore run.node run.action.download step.stream in complete
Sep 24 11:27:55.311 | ip-172-31-21-74   |  > rep.executing container operation.task processor.run container.containerstore run.node run.action.download step.stream in starting
Sep 24 11:27:55.311 | ip-172-31-21-74   |  > rep.executing container operation.task processor.run container.containerstore run.node run.action.download step.fetch complete
Sep 24 11:27:55.274 |                   |  > [ AGENT ] 2018-09-24 09:27:55 UTC | INFO | (transaction.go:136 in Process) | Successfully posted payload to "https://6-4-2-app.agent.datadoghq.com/api/v1/series?api_key=************************d264d"
Sep 24 11:27:55.199 | beanserverprod    |  > 2018-09-24T09:27:55.199+0000 I COMMAND [conn4] command demo command: eval { $eval: "sleep(120)", find: { group: "admin" } } keyUpdates:0 writeConflicts:0 numYields:0 reslen:45 locks:{ Global: { acquireCount…
Sep 24 11:27:55.199 | coffeehouseprod   |  > 2018-09-24T09:27:55.199+0000 I COMMAND [conn4] command demo command: eval { $eval: "sleep(120)", find: { group: "admin" } } keyUpdates:0 writeConflicts:0 numYields:0 reslen:45 locks:{ Global: { acquireCount…
Sep 24 11:27:55.195 | beanserverprod    |  > 2018-09-24T09:27:55.195+0000 I COMMAND [conn4] dbeval slow, time: 120ms demo
Sep 24 11:27:55.195 | coffeehouseprod   |  > 2018-09-24T09:27:55.195+0000 I COMMAND [conn4] dbeval slow, time: 120ms demo
Sep 24 11:27:55.044 | beanserverprod    |  > 2018-09-24T09:27:55.044+0000 I COMMAND [conn33] command demo command: eval { $eval: "sleep(1384)", find: { group: "admin" } } keyUpdates:0 writeConflicts:0 numYields:0 reslen:45 locks:{ Global: { acquireCou…
Sep 24 11:27:55.044 | coffeehouseprod   |  > 2018-09-24T09:27:55.044+0000 I COMMAND [conn33] command demo command: eval { $eval: "sleep(1384)", find: { group: "admin" } } keyUpdates:0 writeConflicts:0 numYields:0 reslen:45 locks:{ Global: { acquireCou…
Sep 24 11:27:55.041 | beanserverprod    |  > 2018-09-24T09:27:55.041+0000 I COMMAND [conn33] sleep(1384)
Sep 24 11:27:55.041 | beanserverprod    |  > 2018-09-24T09:27:55.041+0000 I COMMAND [conn33] dbeval slow, time: 1384ms demo
Sep 24 11:27:55.041 | coffeehouseprod   |  > 2018-09-24T09:27:55.041+0000 I COMMAND [conn33] sleep(1384)
Sep 24 11:27:55.041 | coffeehouseprod   |  > 2018-09-24T09:27:55.041+0000 I COMMAND [conn33] dbeval slow, time: 1384ms demo
Sep 24 11:27:54.429 | coffeehouseprod   |  > [ TRACE ] 2018-09-24 09:27:54 INFO (trace_writer.go:98) - flushed trace payload to the API, time:141.224318ms , size:1809 bytes
Sep 24 11:27:54.385 | DDAzureDemoSQL    |  > 2018-09-24 09:27:53.65 Logon Login succeeded for user 'datadog' . Connection made using SQL Server authentication. [CLIENT: 10.8.0.4]
Sep 24 11:27:54.324 | ip-172-31-21-74   |  > rep.executing container operation.task processor.run container.containerstore run.node run.action.download step.downloader.download.completed
Sep 24 11:27:54.312 | ip-172-31-21-74   |  > rep.executing container operation.task processor.run container.containerstore run.node run.action.download step.downloader.download.copy to destination file.copy finished
Sep 24 11:27:54.312 | ip-172-31-21-74   |  > rep.executing container operation.task processor.run container.containerstore run.node run.action.download step.downloader.download.fetch request
Sep 24 11:27:54.312 | ip-172-31-21-74   |  > rep.executing container operation.finished
Sep 24 11:27:54.312 | ip-172-31-21-74   |  > rep.executing container operation.task processor.task already started
```

[Triggered] [cortado] Cluster lost 3.275 NodeManagers

#account:prod #aws:elasticmapreduce:job-flow-id: ...

Cluster lost at least 3.0 NodeManagers. This may lead to `ExternalShuffleService` issue



```
avg(last_30m):max:yarn.metrics.unhealthy_nodes{*} by {mortar_cluster_id,host,mortar_user} + max:yarn.metrics.lost_nodes{*} by
{mortar_cluster_id,host,mortar_user} + max:yarn.metrics.decommissioned_nodes{*} by {mortar_cluster_id,host,mortar_user} >= 3
```

Building highly reliable data pipelines @ Datadog

Quentin FRANCOIS

Team Lead, Data Engineering

DataEng Barcelona ‘18

Reliability is the probability that a system will produce correct outputs up to some given time **t**.

Source: E.J. McClusky & S. Mitra (2004). "Fault Tolerance" in Computer Science Handbook 2ed. ed. A.B. Tucker. CRC Press.

Highly reliable data pipelines

1. Architecture

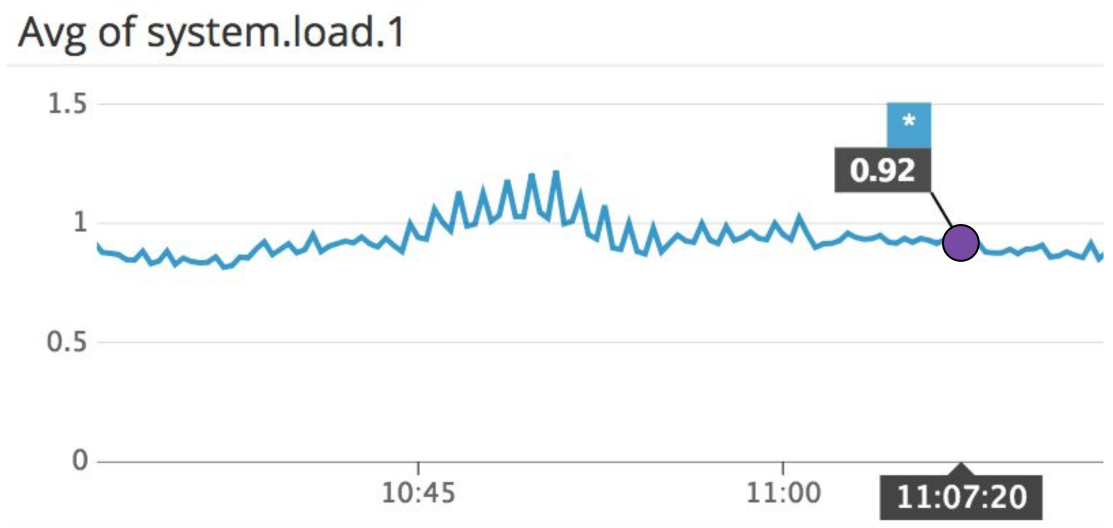
Highly reliable data pipelines

1. Architecture
2. Monitoring

Highly reliable data pipelines

1. Architecture
2. Monitoring
3. Failures handling

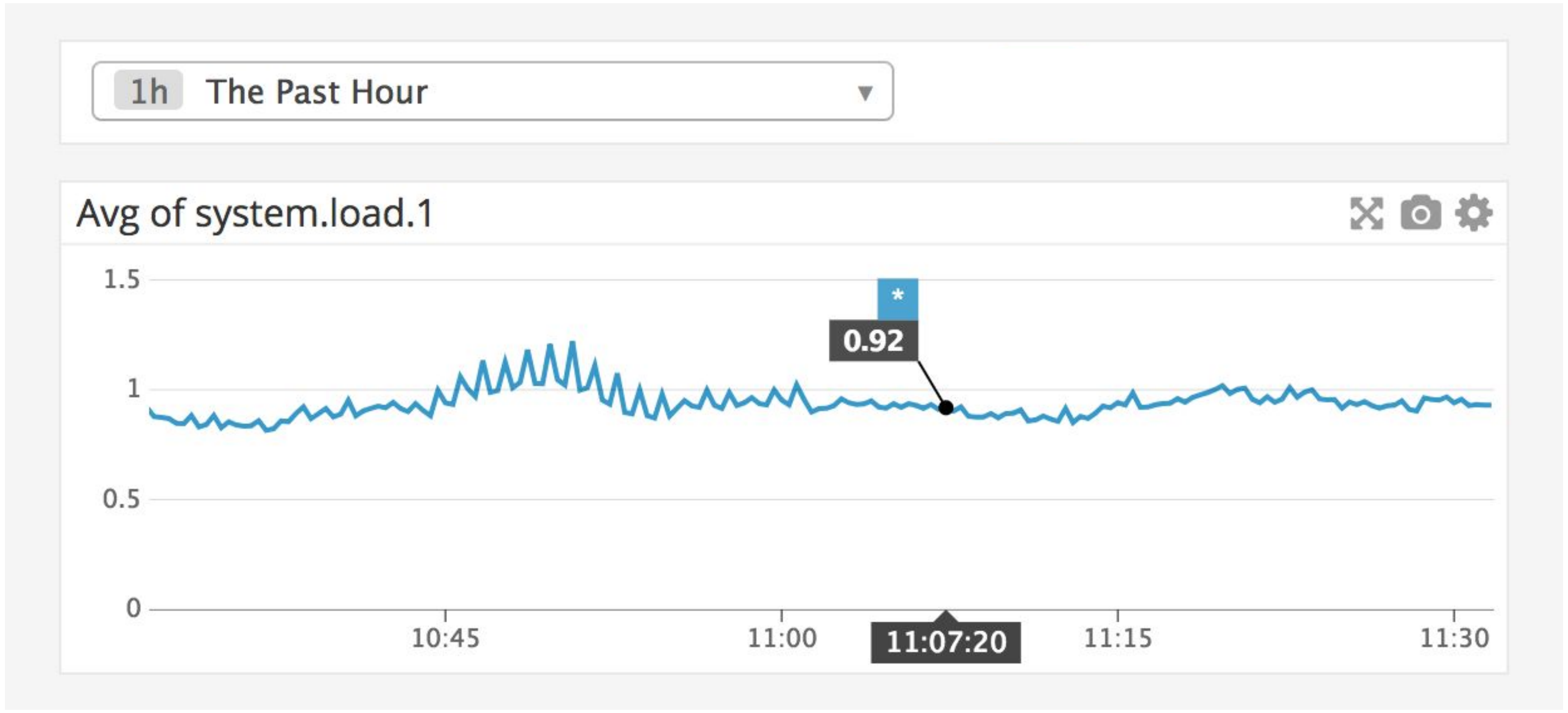
Historical metric queries



Time series data

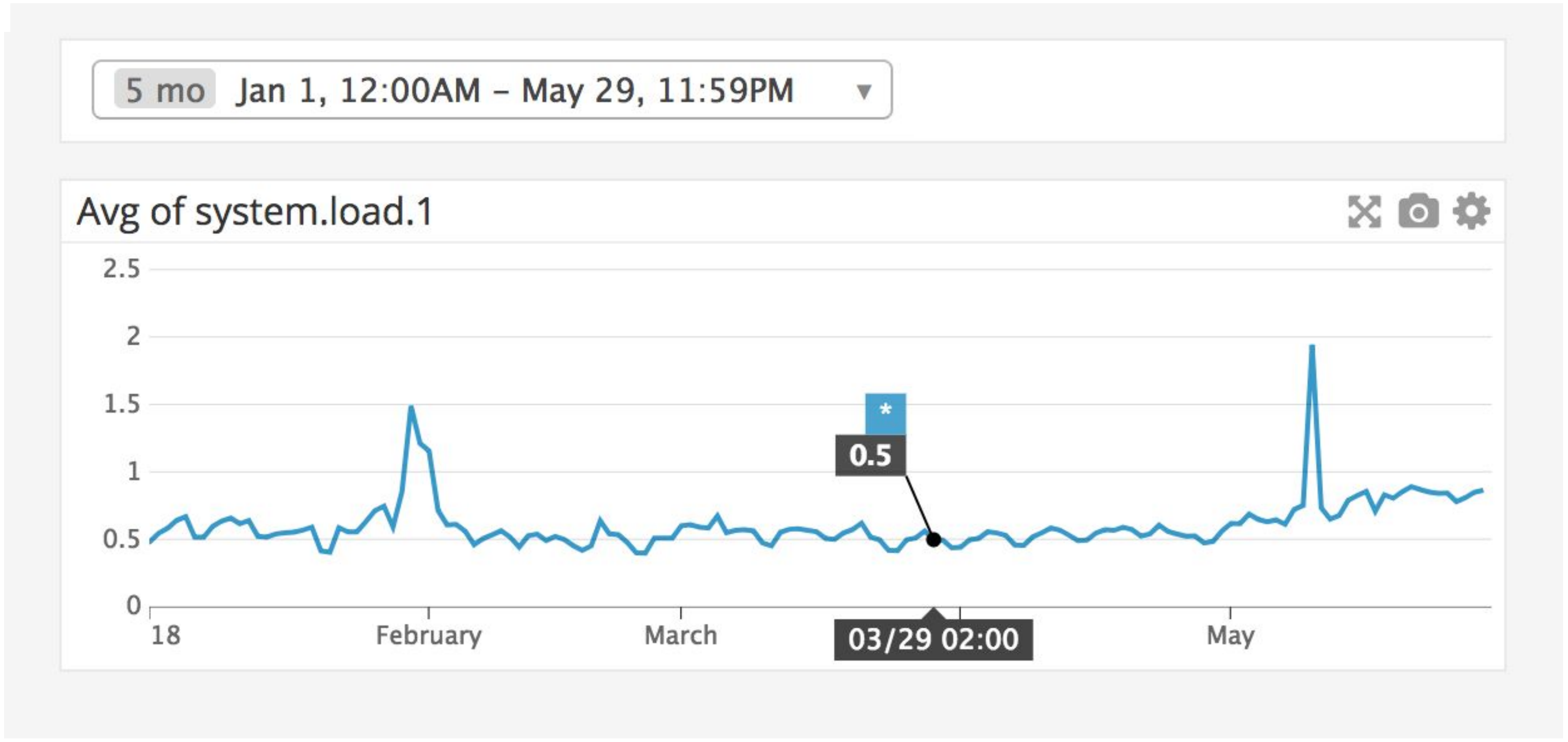
metric	system.load.1
timestamp	1526382440
value	0.92
tags	host:i-xyz,env:dev,...

Historical metric queries



1 point/second

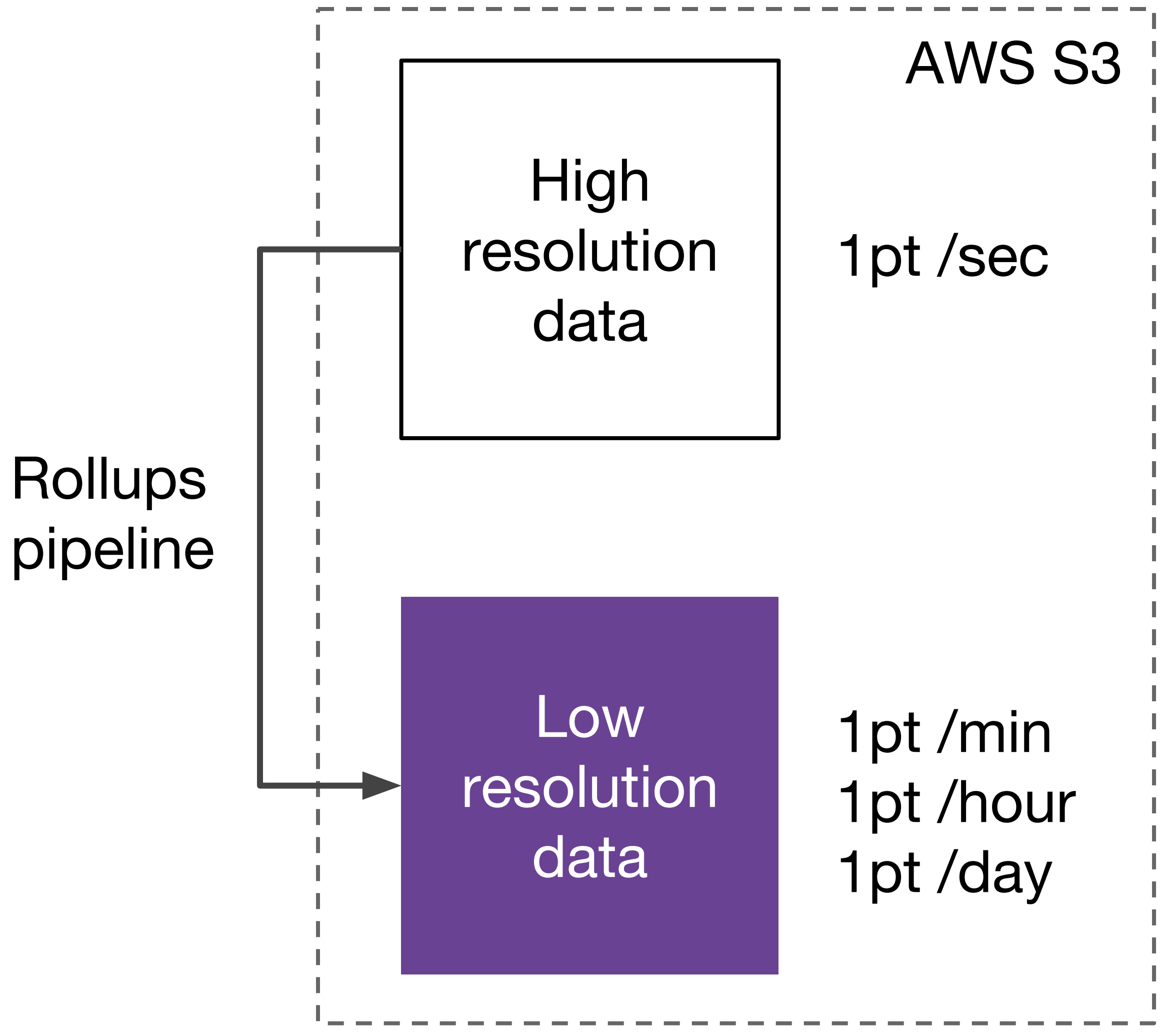
Historical metric queries


5 mo Jan 1, 12:00AM – May 29, 11:59PM

Avg of system.load.1

1 point/day

Historical metric queries

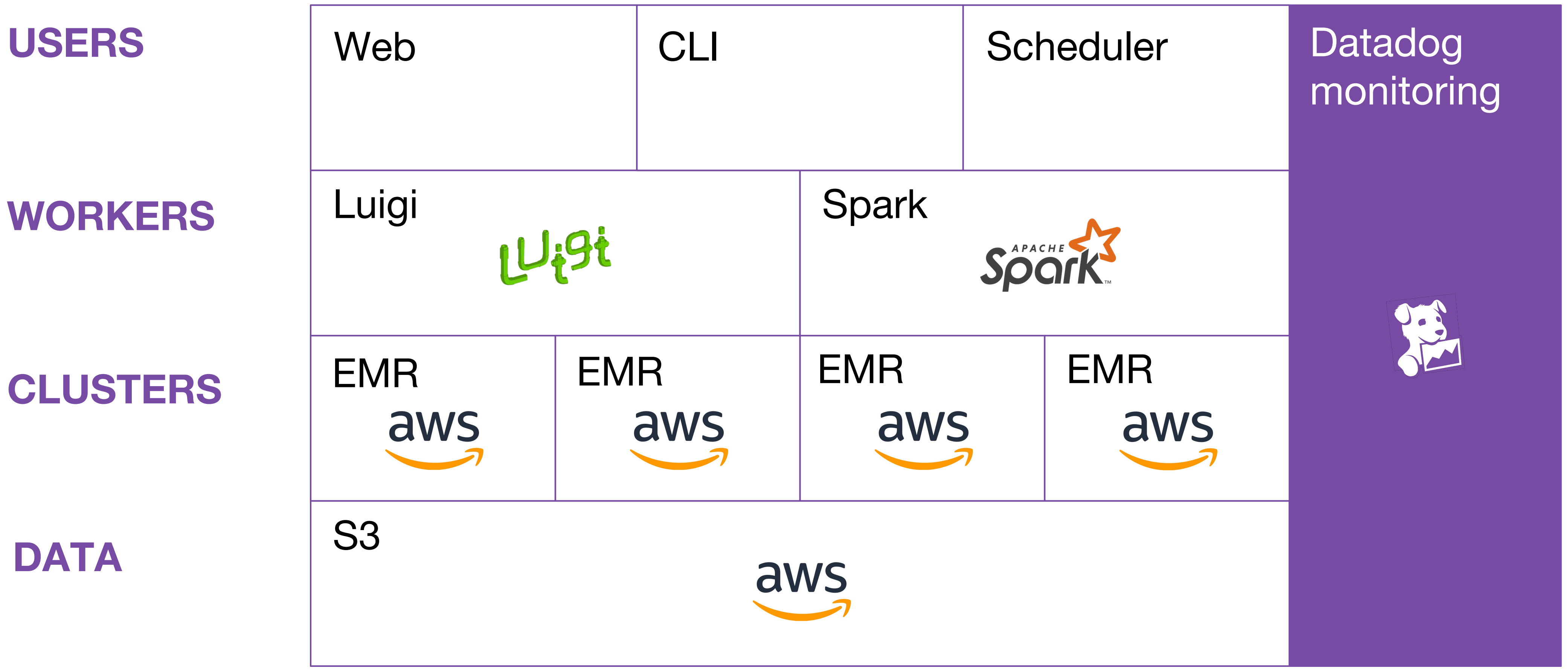


- Runs once a day.
- Dozens of TBs of input data.
- Trillions of points processed.

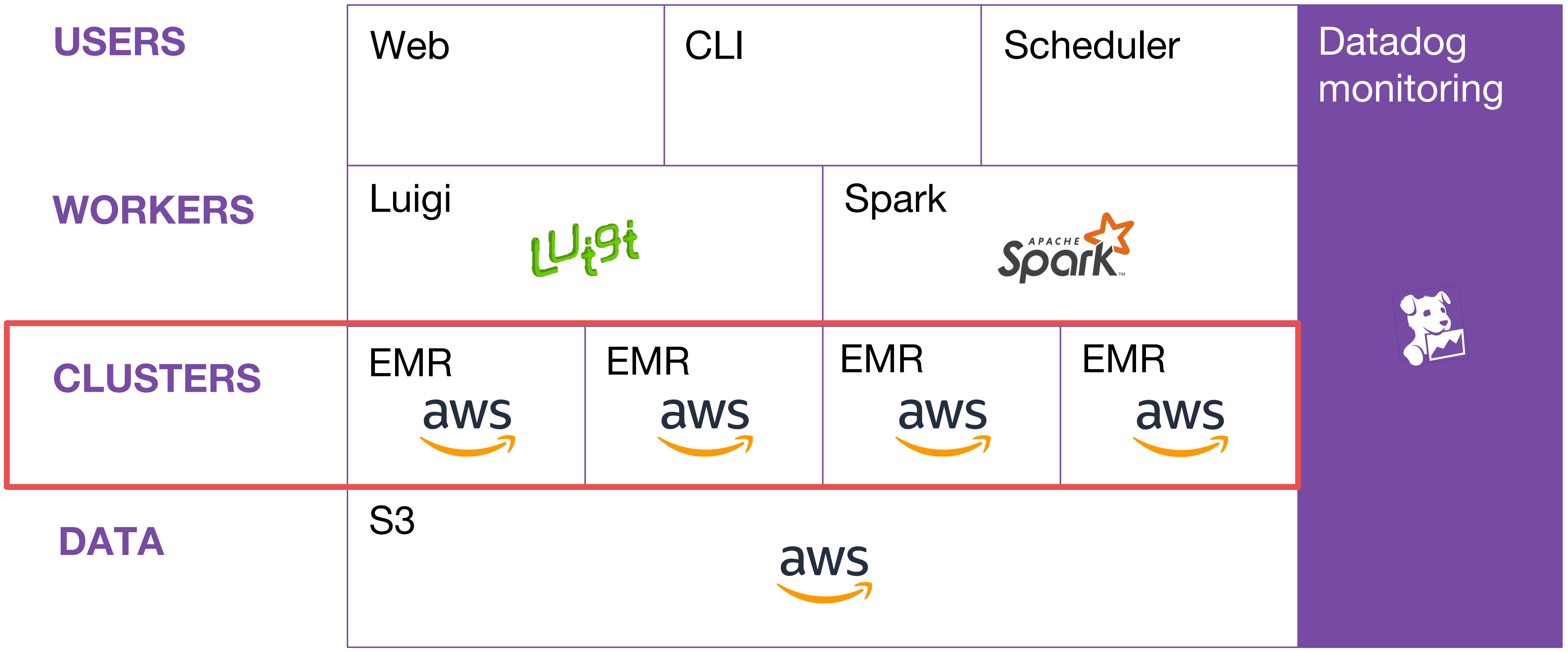
Highly reliable data pipelines

1. Architecture
2. Monitoring
3. Failures handling

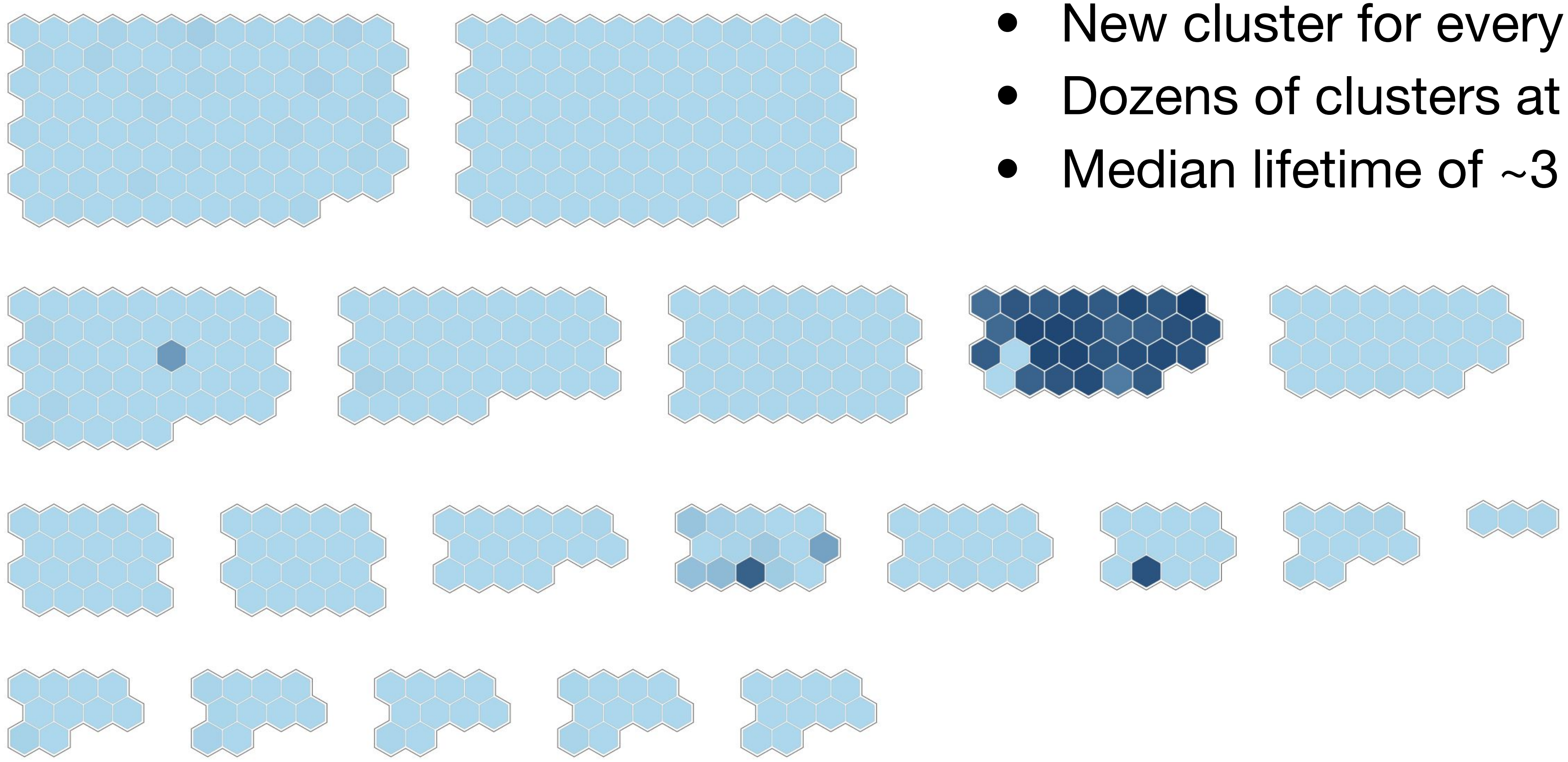
Our big data platform architecture



Our big data platform architecture

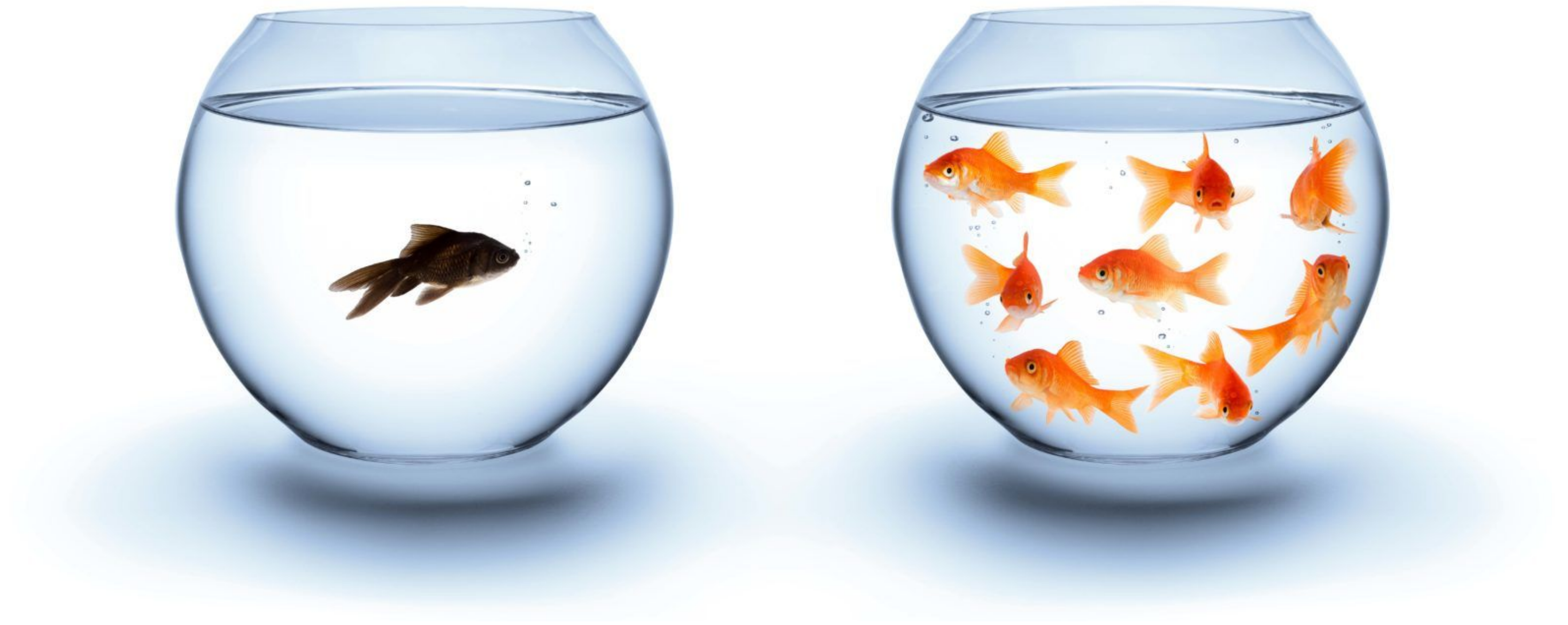


Many ephemeral clusters



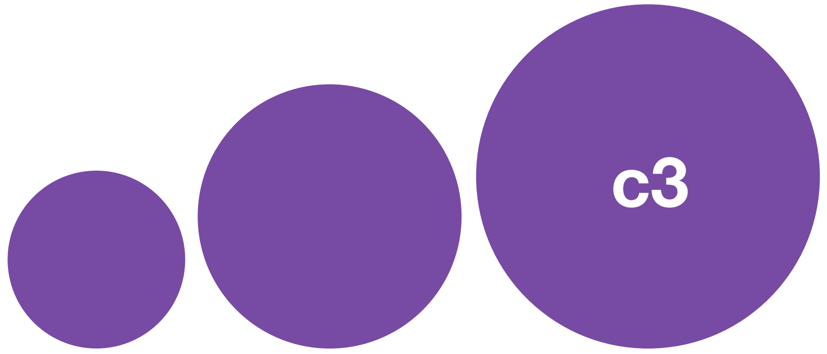
- New cluster for every pipeline.
- Dozens of clusters at a time.
- Median lifetime of ~3 hours.

Total isolation



We know what is happening and why.

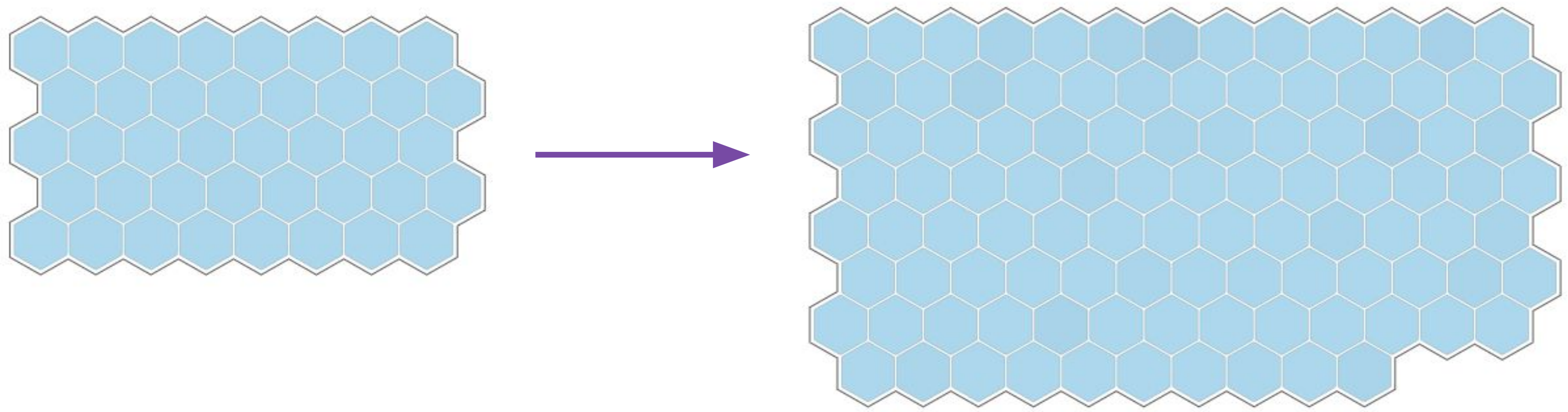
Pick the best hardware for each job



For CPU-bound jobs

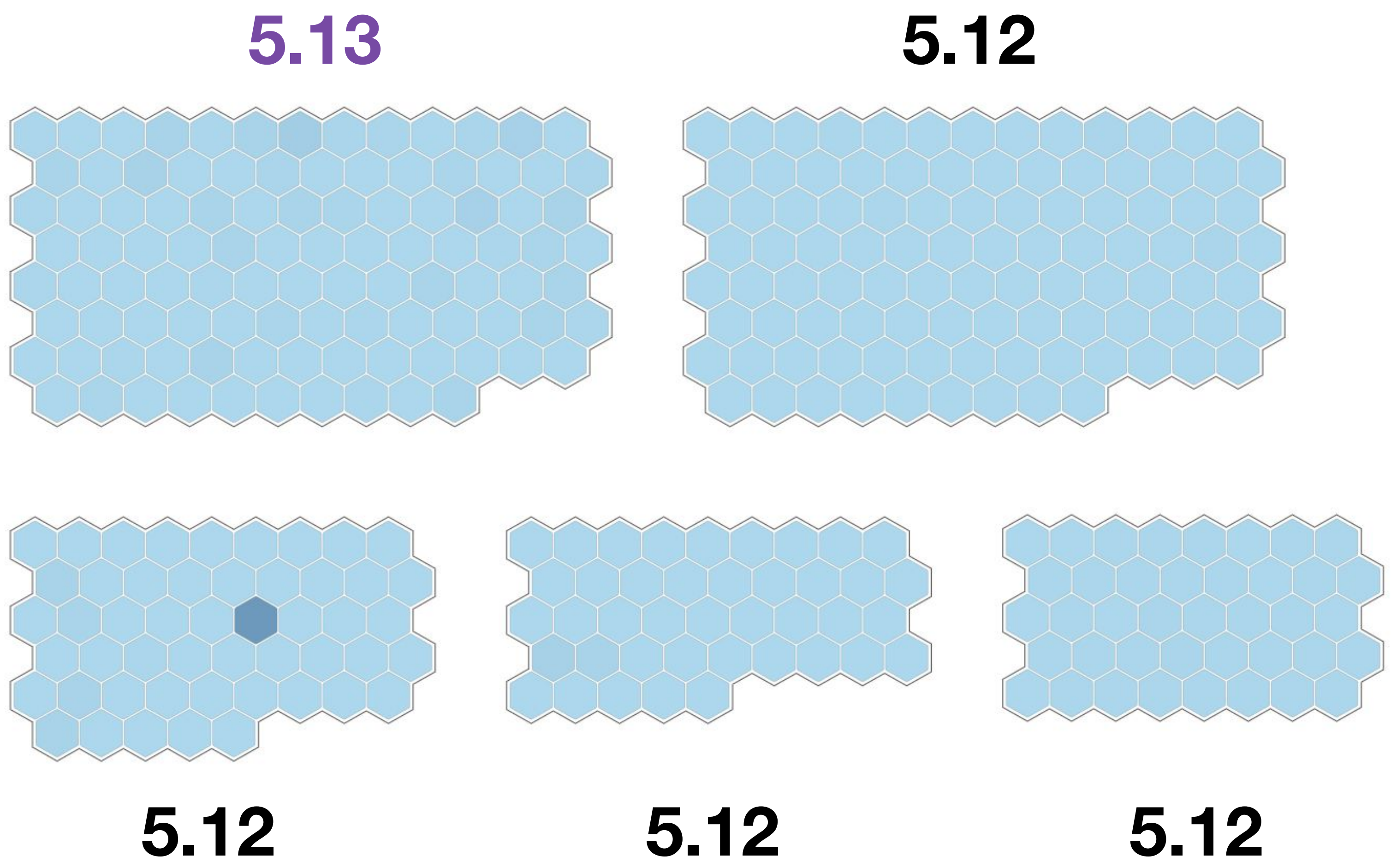
For memory-bound jobs

Scale up/down clusters



- If we are behind.
- Scale as we grow.
- No more waiting on loaded clusters.

Safer upgrades of EMR/Hadoop/Spark



Spot-instance clusters

+ Ridiculous savings
(up to 80% off the on-demand price)

- Nodes can die at any time

Spot-instance clusters

+ Ridiculous savings
(up to 80% off the on-demand price)

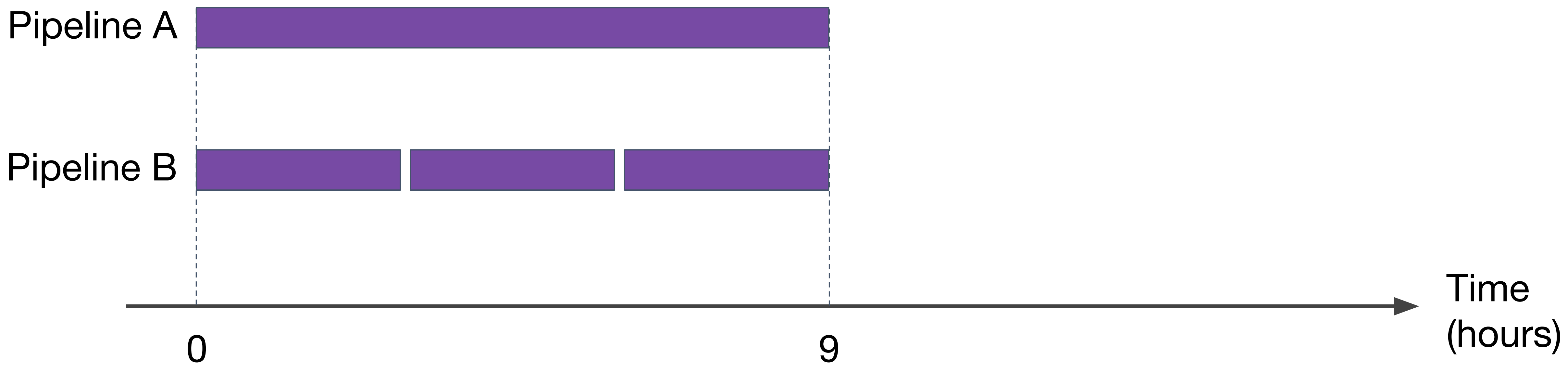
- Nodes can die at any time

How can we build **highly reliable** data pipelines with **instances killed randomly** all the time?

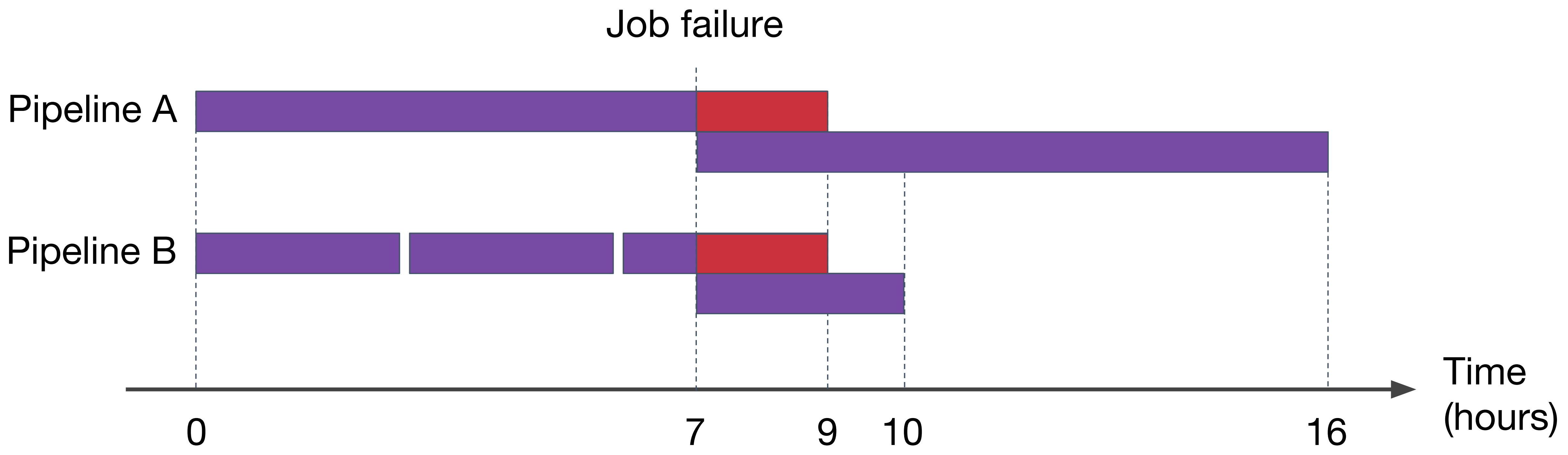
No long running jobs

- The longer the job, the more work you lose on average.
- The longer the job, the longer it takes to recover.

No long running jobs



No long running jobs

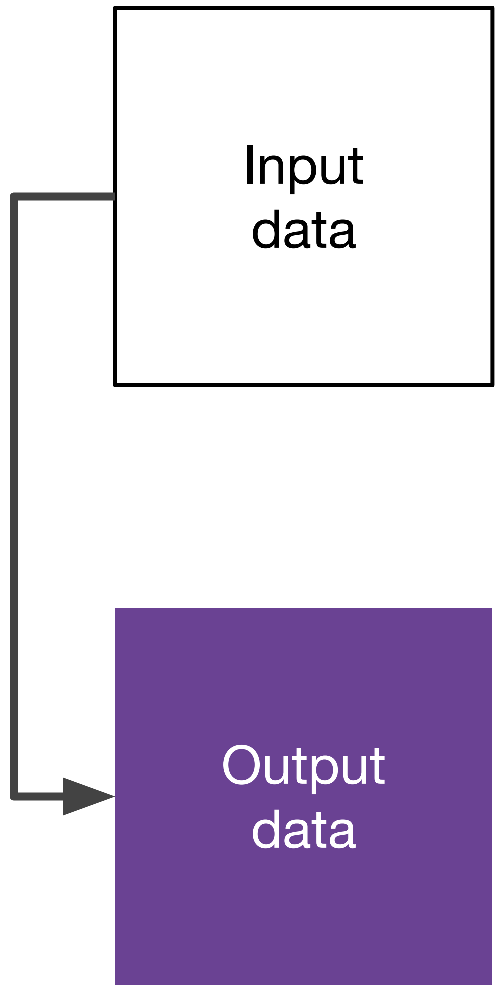


Break down jobs into smaller pieces

Vertically - persist intermediate data between transformations.

Horizontally - partition the input data.

Example
Rollups pipeline



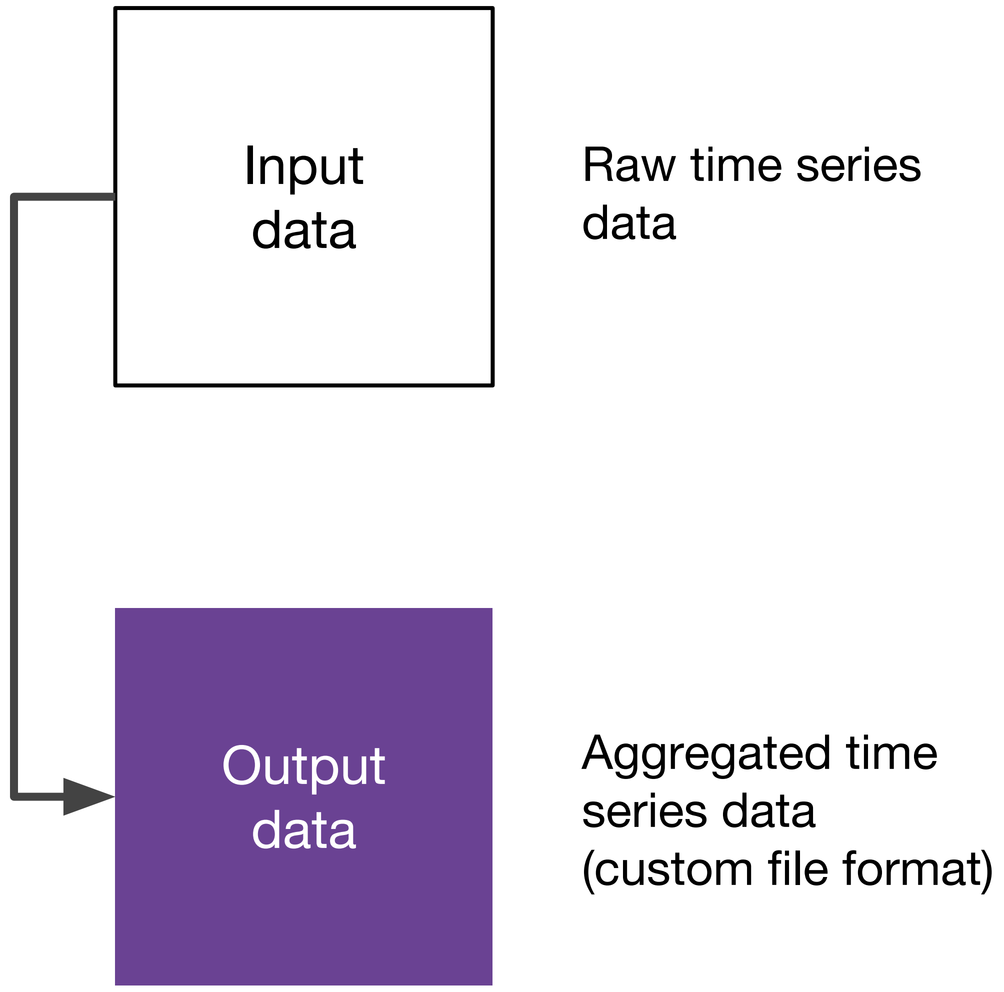
Raw time series data

Aggregated time series data (custom file format)

Example

Rollups pipeline

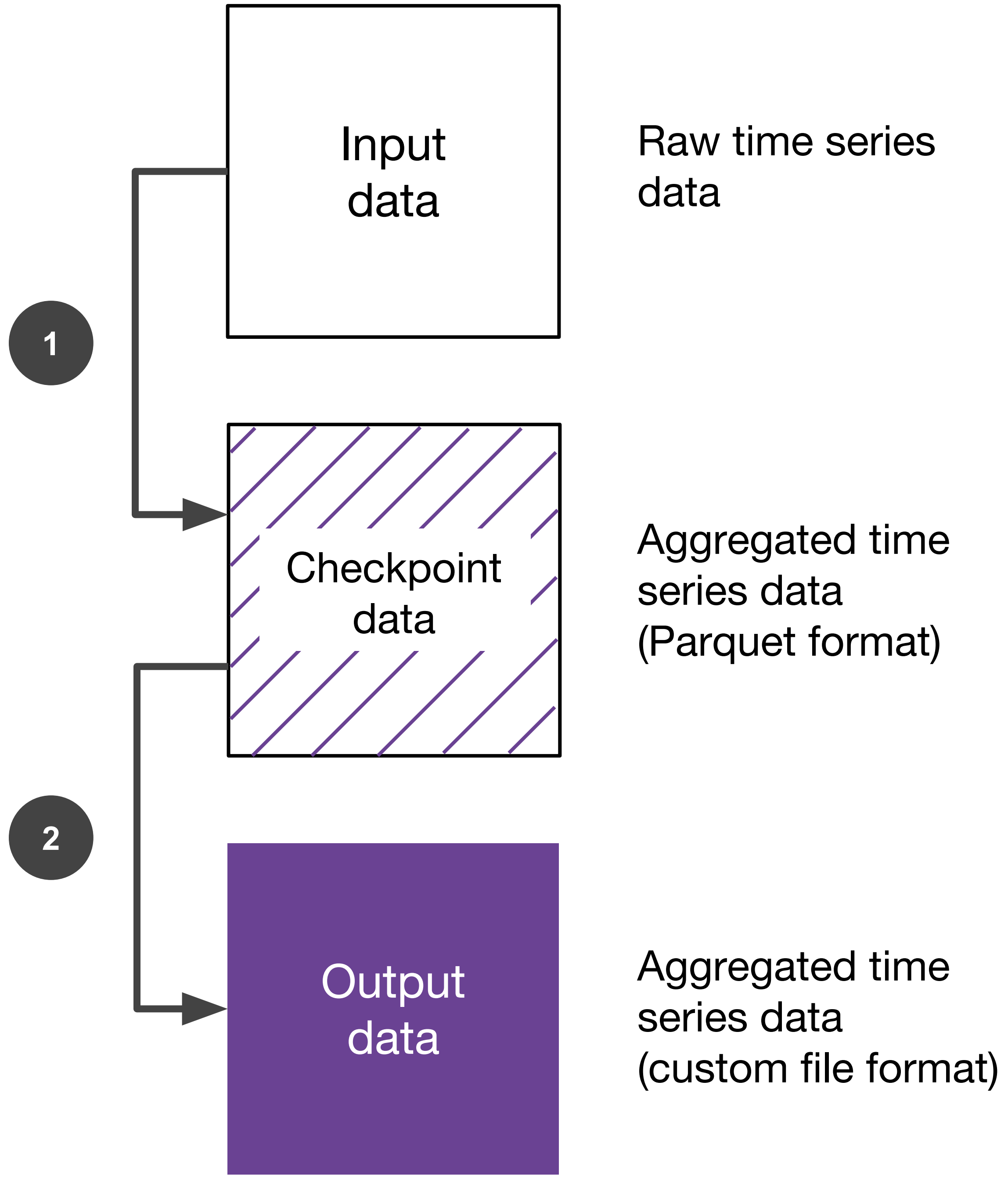
1. **Aggregate** high resolution data.
2. **Store** the aggregated data in our custom file format.



Example
Vertical split

1. **Aggregate** high resolution data.

2. **Store** the aggregated data in our custom file format.

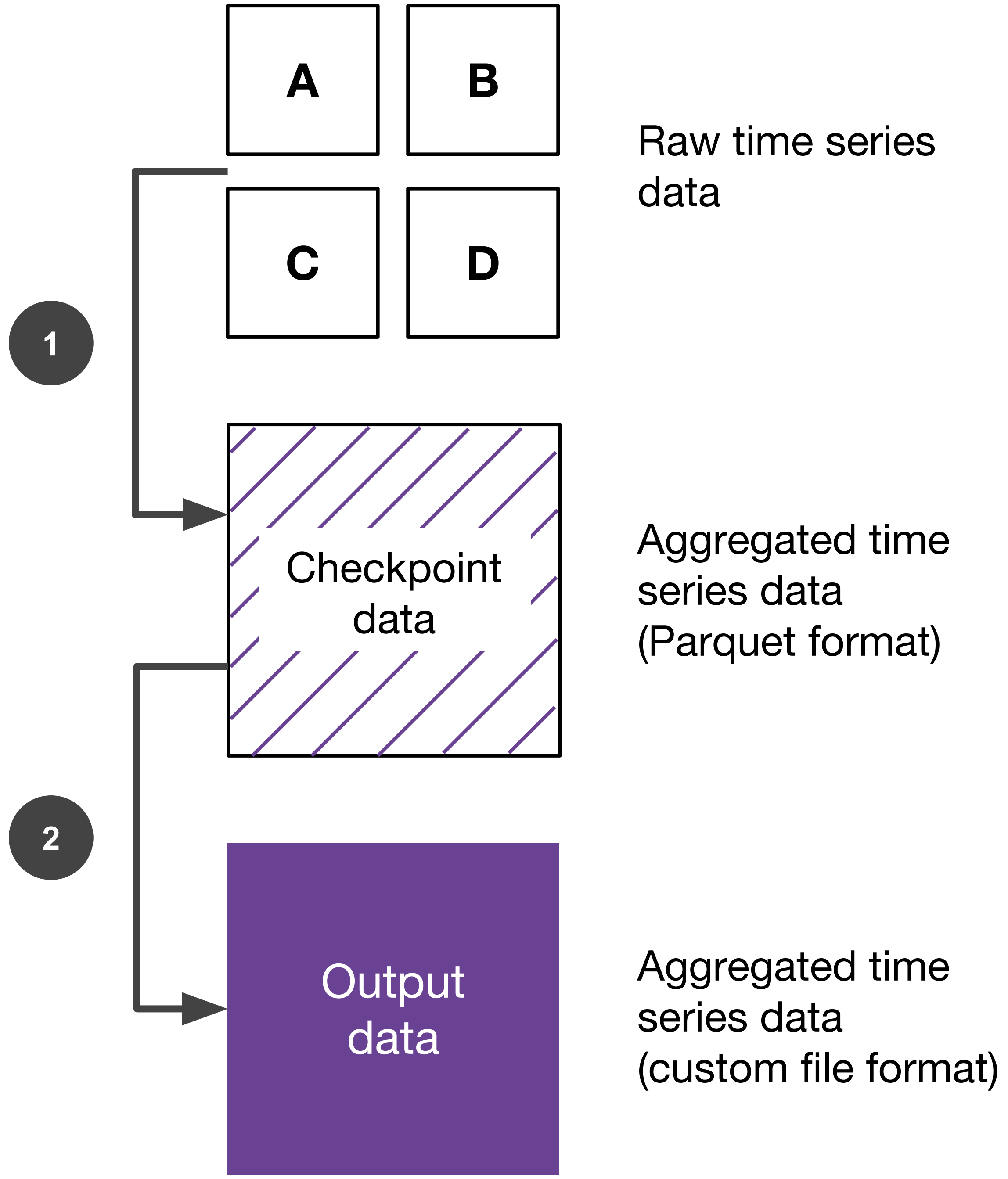


1
2
Input data
Raw time series data
Checkpoint data
Aggregated time series data (Parquet format)
Output data
Aggregated time series data (custom file format)

Example
Horizontal split

1. **Aggregate** high resolution data.

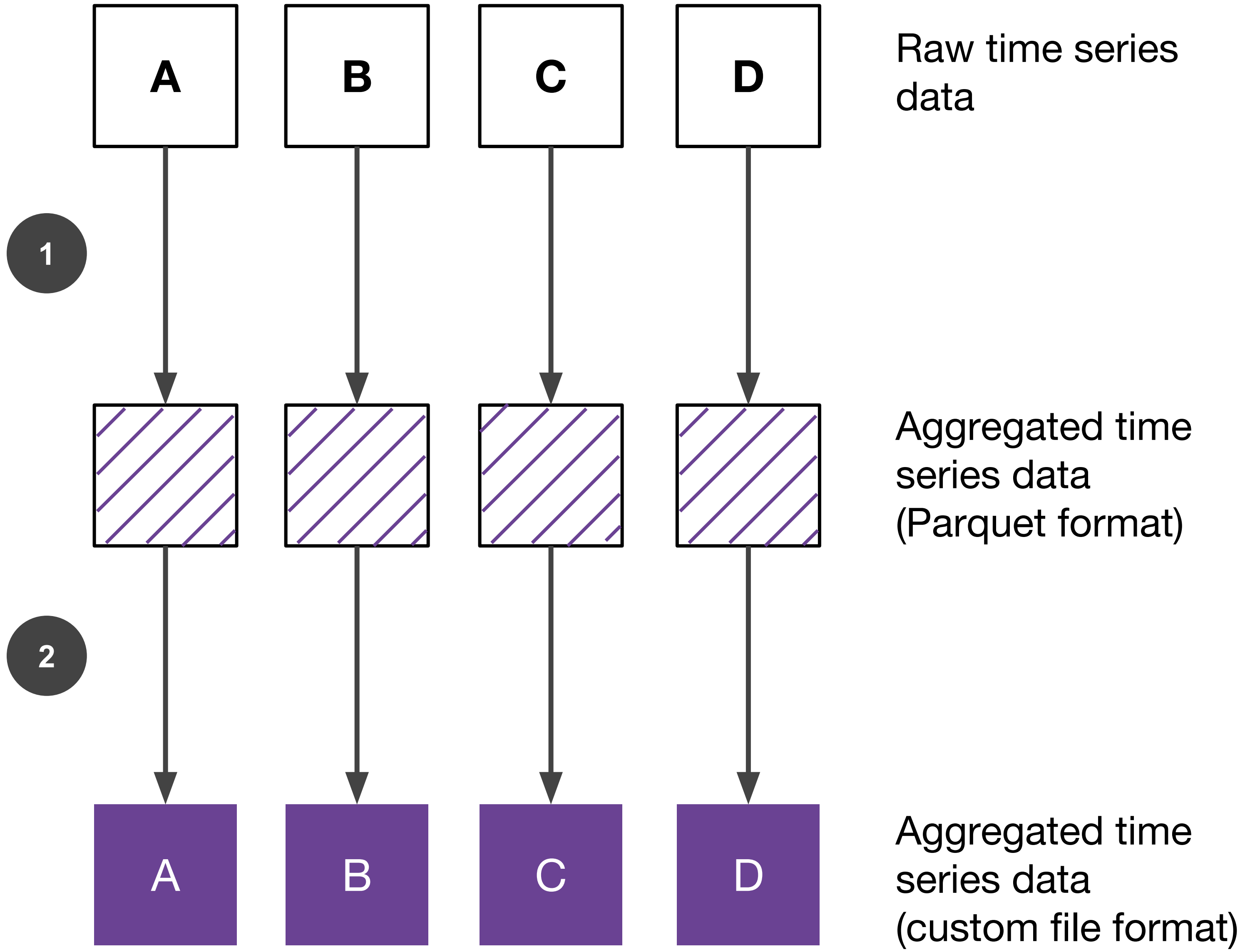
2. **Store** the aggregated data in our custom file format.


A
B
C
D
Raw time series data
1
Checkpoint data
Aggregated time series data (Parquet format)
2
Output data
Aggregated time series data (custom file format)

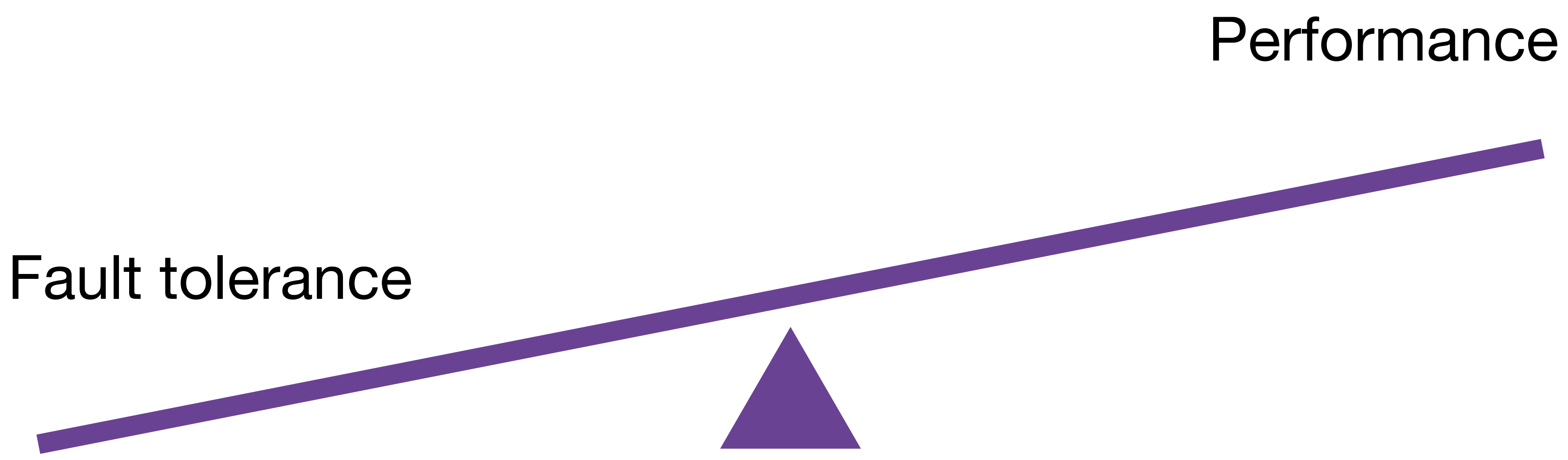
Example
Horizontal split

1. **Aggregate** high resolution data.

2. **Store** the aggregated data in our custom file format.


A
B
C
D
Raw time series data
1
Aggregated time series data (Parquet format)
2
A
B
C
D
Aggregated time series data (custom file format)

Break down jobs into smaller pieces



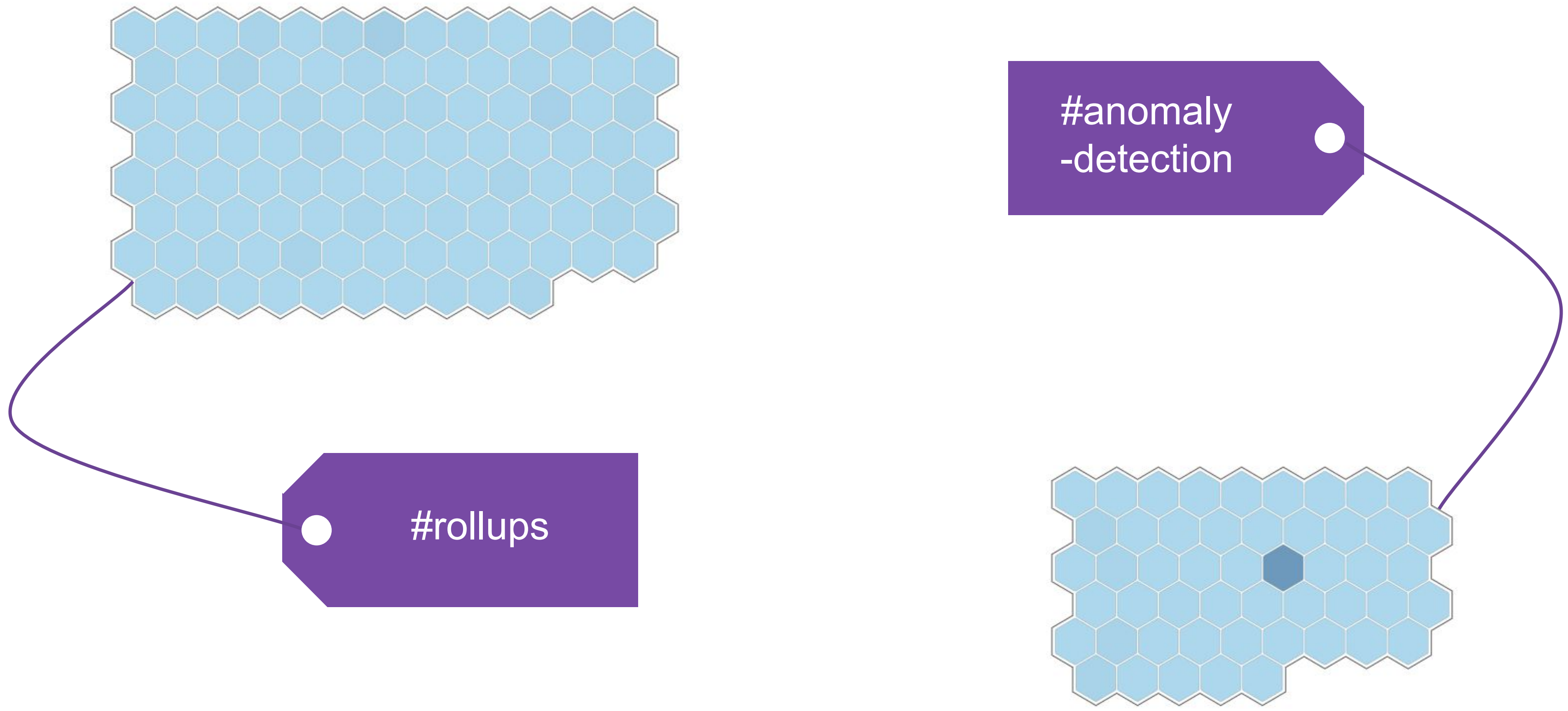
Lessons

- Many clusters for better isolation.
- Break down jobs into smaller pieces.
- Trade-off between performance and fault tolerance.

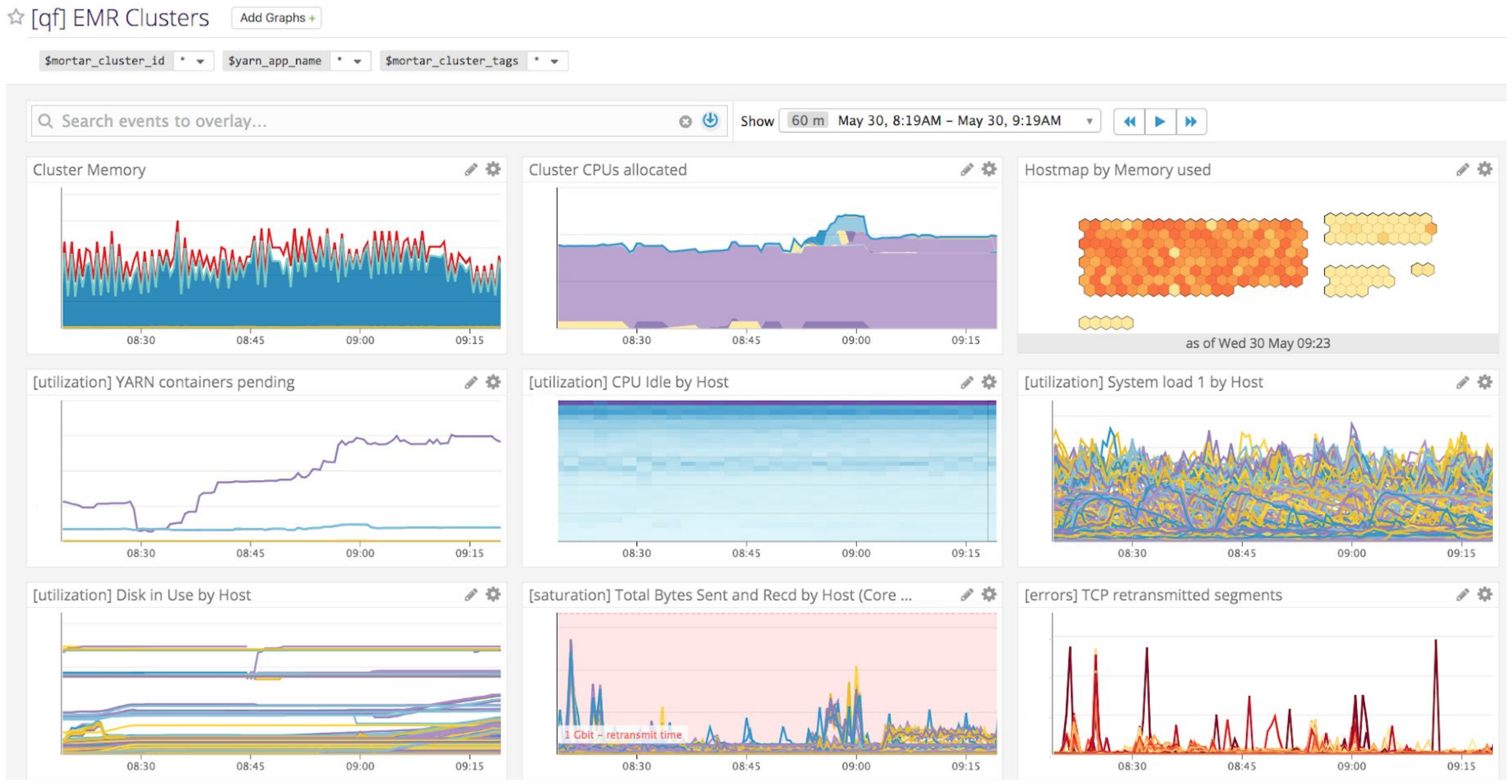
Highly reliable data pipelines

1. Architecture
2. Monitoring
3. Failures handling

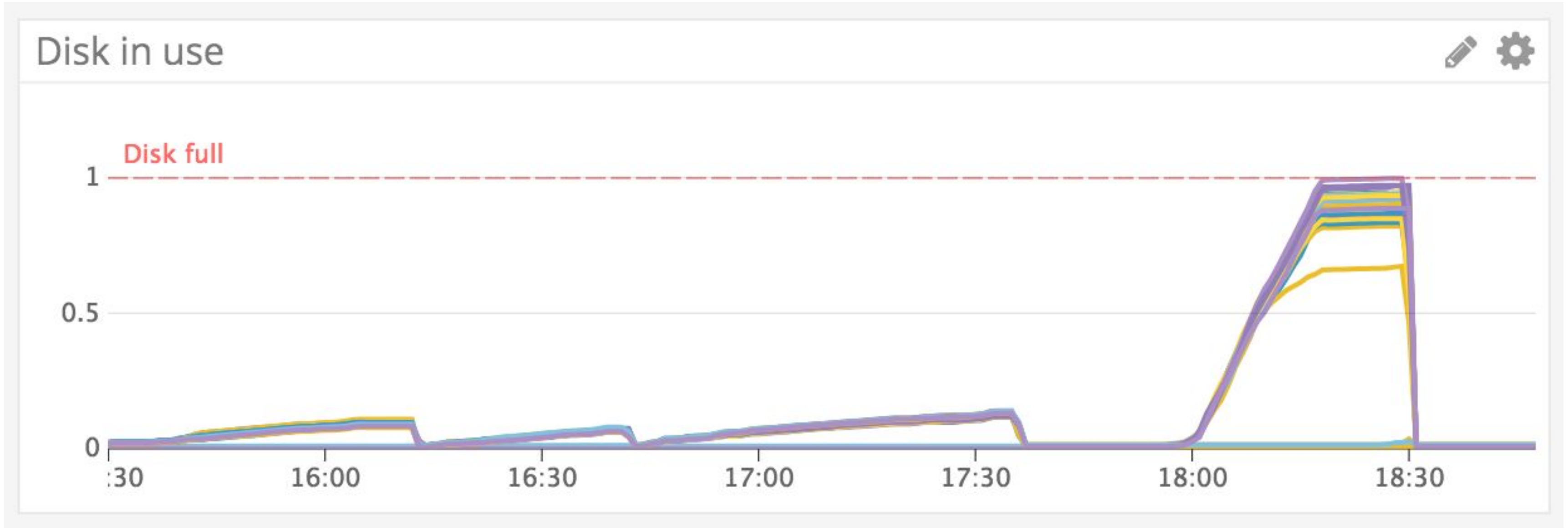
Cluster tagging



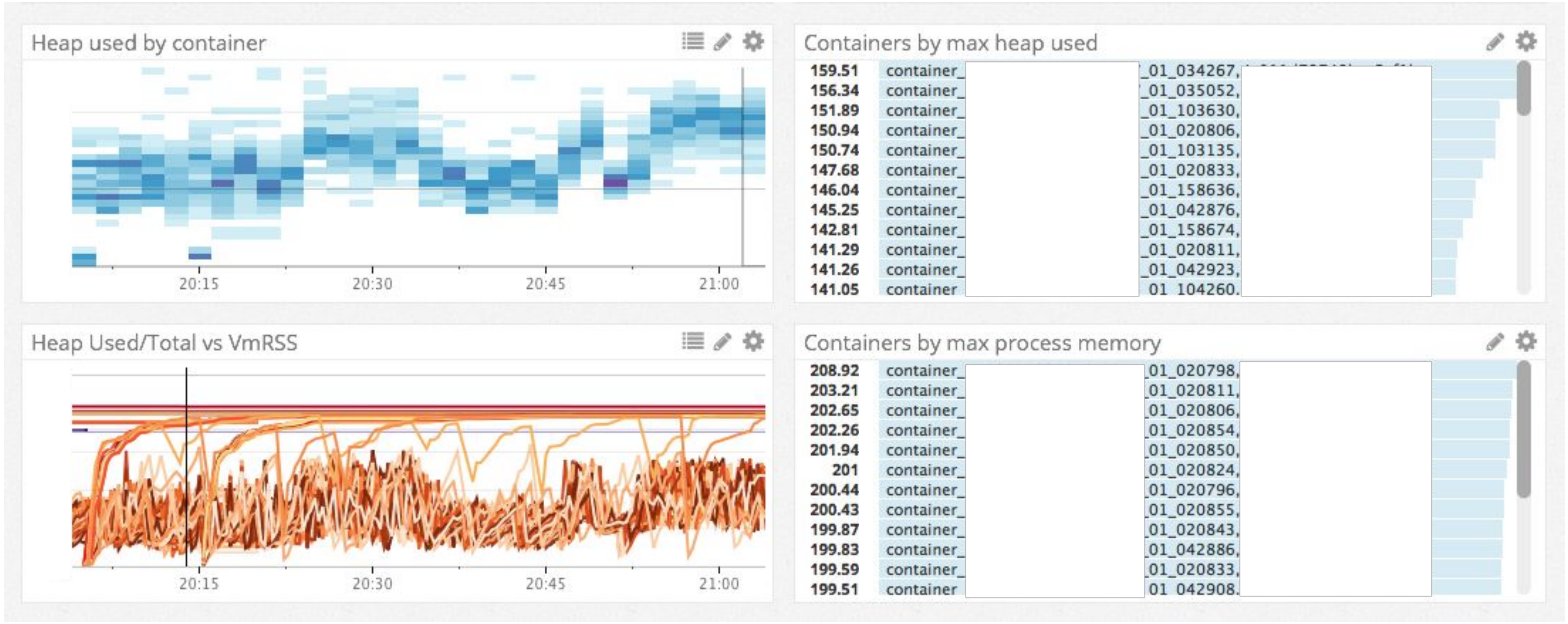
Monitor cluster metrics


[qf] EMR Clusters
Add Graphs +
$mortar_cluster_id *
$yarn_app_name *
$mortar_cluster_tags *
Search events to overlay...
Show
60 m
May 30, 8:19AM – May 30, 9:19AM
Cluster Memory
Cluster CPUs allocated
Hostmap by Memory used
as of Wed 30 May 09:23
[utilization] YARN containers pending
[utilization] CPU Idle by Host
[utilization] System load 1 by Host
[utilization] Disk in Use by Host
[saturation] Total Bytes Sent and Recd by Host (Core ...
1 Gbit – retransmit time
[errors] TCP retransmitted segments
08:30
08:45
09:00
09:15

Monitor cluster metrics



Monitor cluster metrics



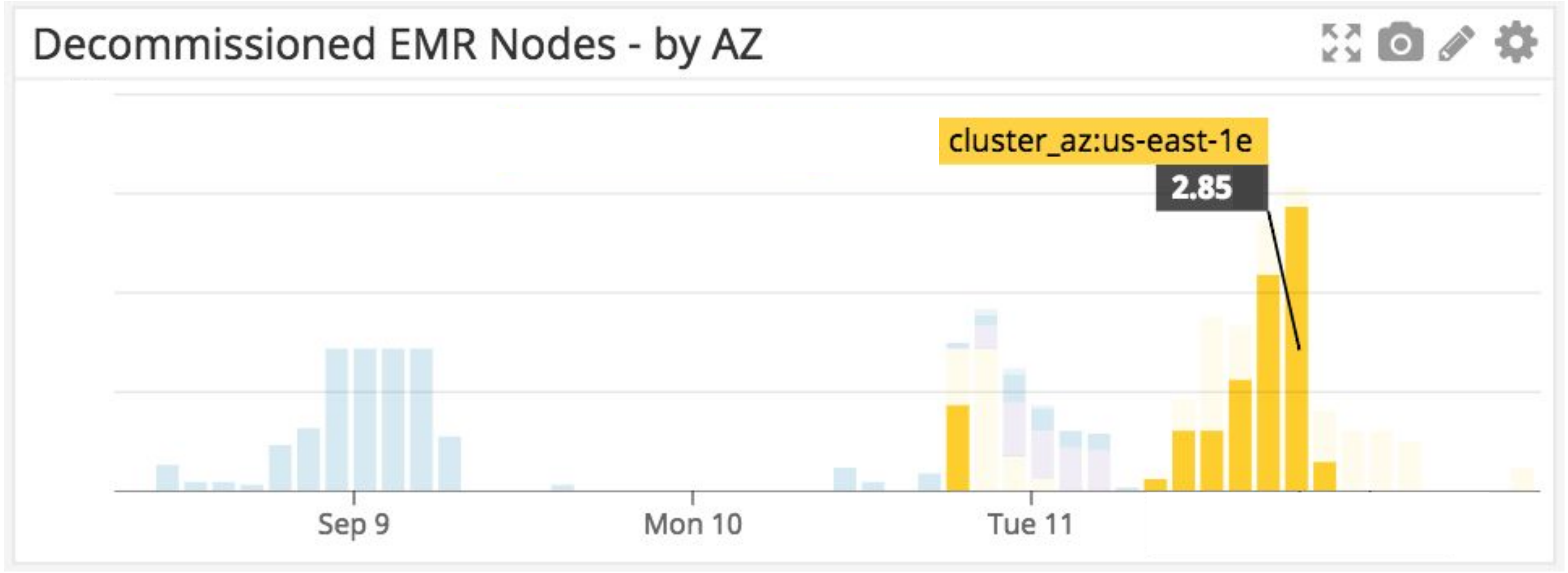
Containers by max heap used

159.51	container_	[redacted]	_01_034267,	[redacted]
156.34	container_	[redacted]	_01_035052,	[redacted]
151.89	container_	[redacted]	_01_103630,	[redacted]
150.94	container_	[redacted]	_01_020806,	[redacted]
150.74	container_	[redacted]	_01_103135,	[redacted]
147.68	container_	[redacted]	_01_020833,	[redacted]
146.04	container_	[redacted]	_01_158636,	[redacted]
145.25	container_	[redacted]	_01_042876,	[redacted]
142.81	container_	[redacted]	_01_158674,	[redacted]
141.29	container_	[redacted]	_01_020811,	[redacted]
141.26	container_	[redacted]	_01_042923,	[redacted]
141.05	container_	[redacted]	_01_104260,	[redacted]

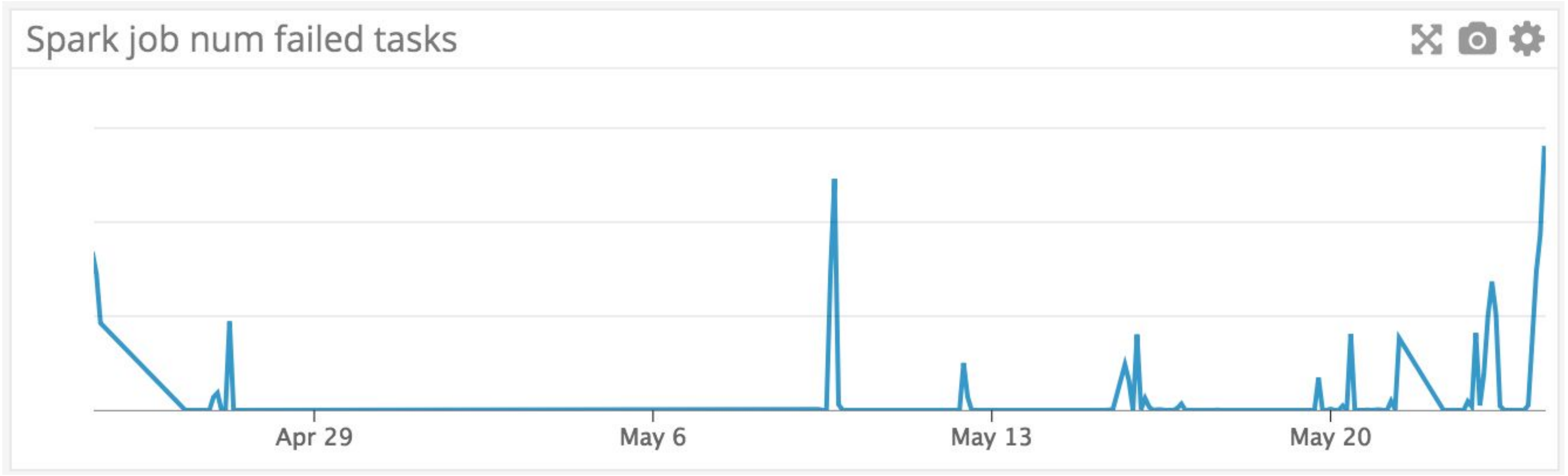
Containers by max process memory

208.92	container_	[redacted]	_01_020798,	[redacted]
203.21	container_	[redacted]	_01_020811,	[redacted]
202.65	container_	[redacted]	_01_020806,	[redacted]
202.26	container_	[redacted]	_01_020854,	[redacted]
201.94	container_	[redacted]	_01_020850,	[redacted]
201	container_	[redacted]	_01_020824,	[redacted]
200.44	container_	[redacted]	_01_020796,	[redacted]
200.43	container_	[redacted]	_01_020855,	[redacted]
199.87	container_	[redacted]	_01_020843,	[redacted]
199.83	container_	[redacted]	_01_042886,	[redacted]
199.59	container_	[redacted]	_01_020833,	[redacted]
199.51	container_	[redacted]	_01_042908,	[redacted]

Monitor cluster metrics

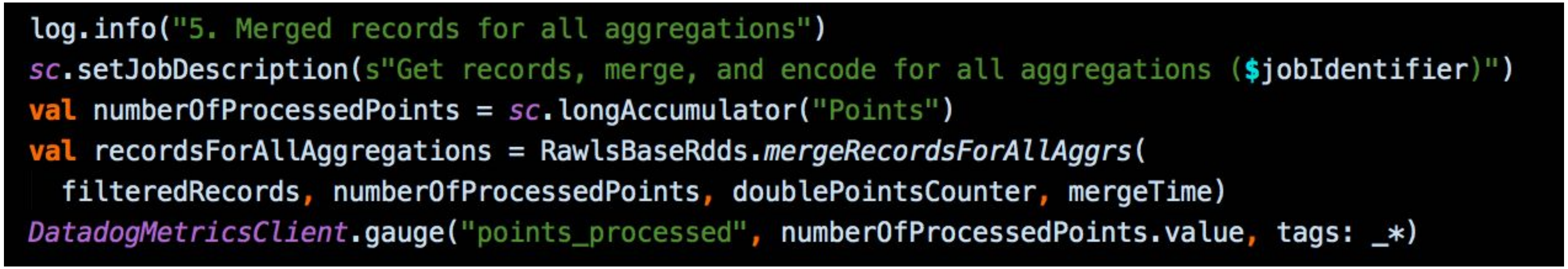


Monitor work metrics



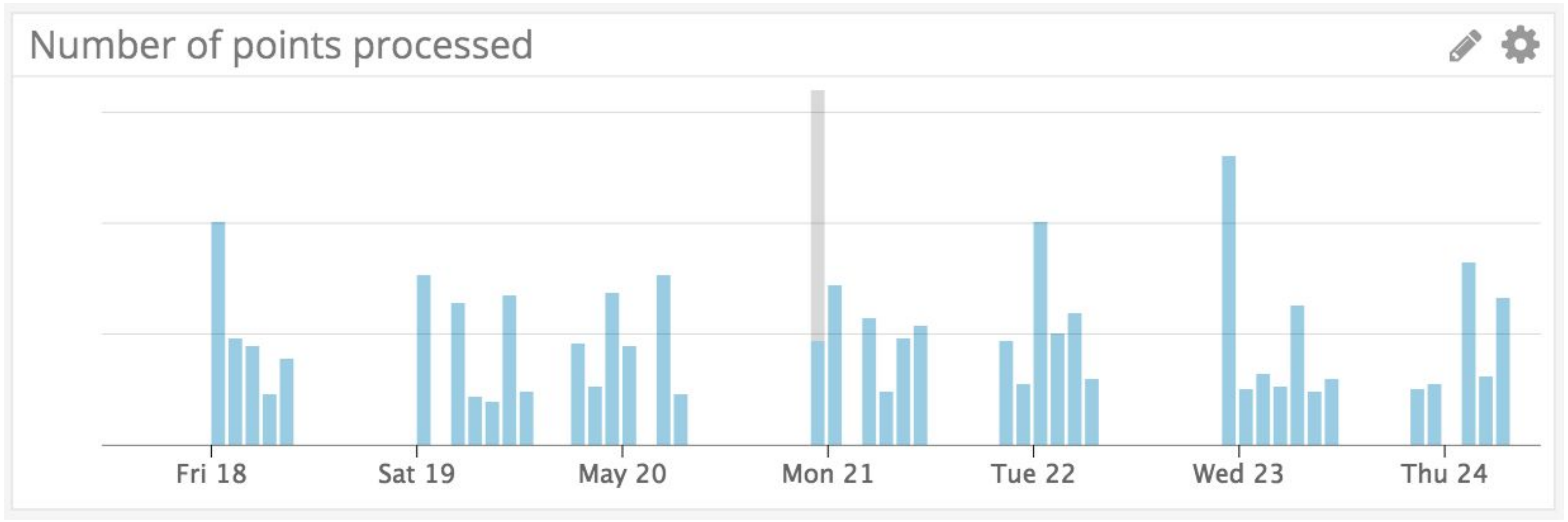
More details: datadoghq.com/blog/monitoring-spark/

Monitor work metrics

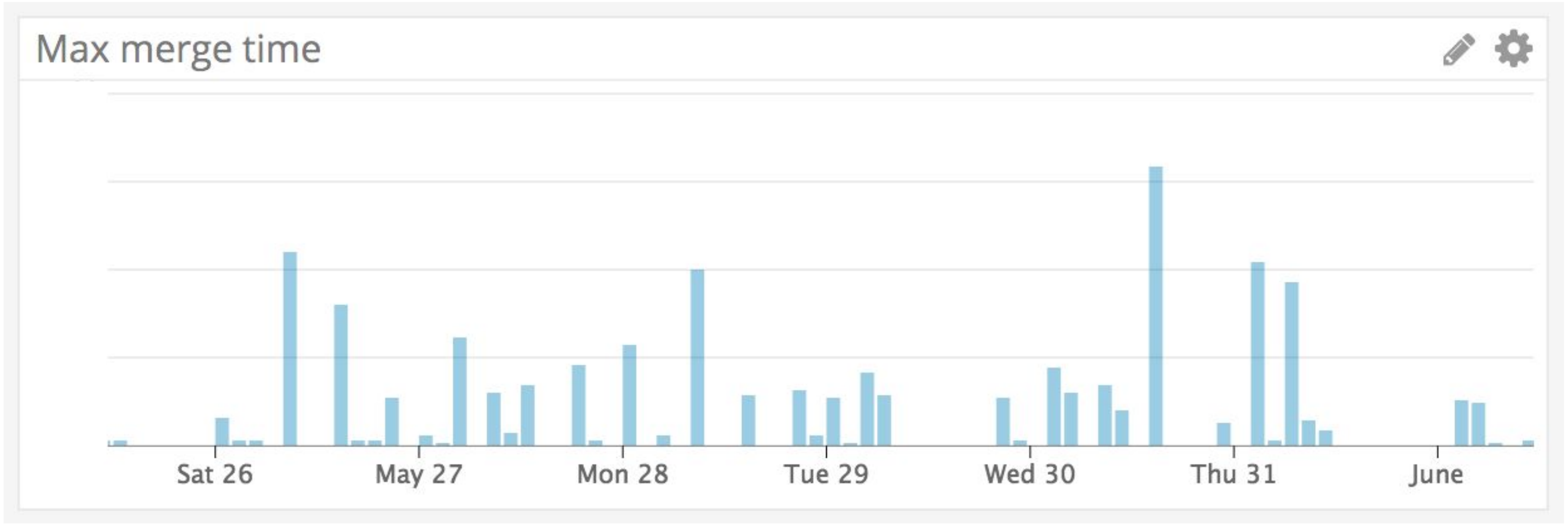


```
log.info("5. Merged records for all aggregations")
sc.setJobDescription(s"Get records, merge, and encode for all aggregations ($jobIdentifier)")
val numberOfProcessedPoints = sc.longAccumulator("Points")
val recordsForAllAggregations = RawlsBaseRdds.mergeRecordsForAllAggrs(
  filteredRecords, numberOfProcessedPoints, doublePointsCounter, mergeTime)
DatadogMetricsClient.gauge("points_processed", numberOfProcessedPoints.value, tags: _*)
```

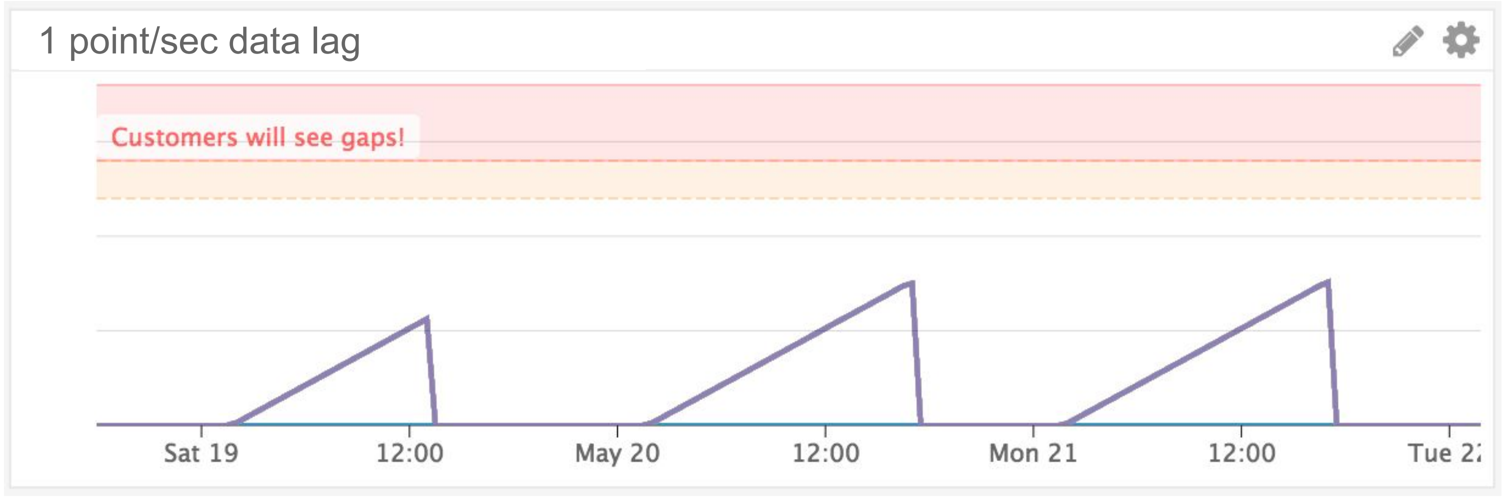
Monitor work metrics

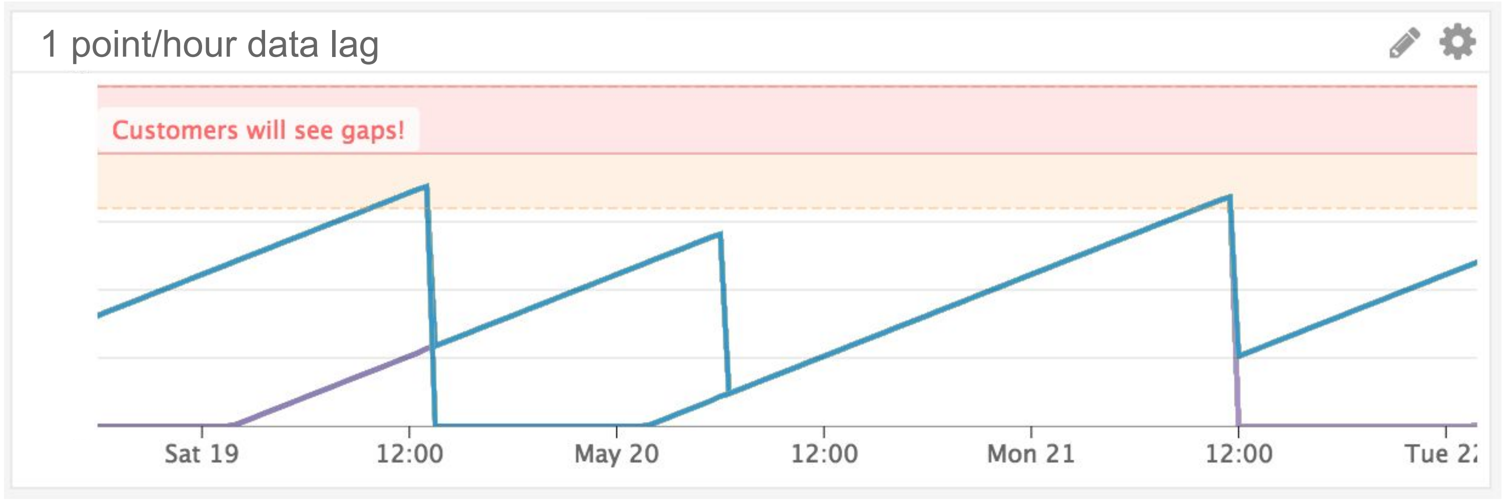


Monitor work metrics



Monitor data lag





Lessons

- Measure, measure and measure!
- Alert on meaningful and actionable metrics.
- High level dashboards.

Highly reliable data pipelines

1. Architecture
2. Monitoring
3. Failures handling

BRACE YOURSELF
FAILURES ARE COMING

Data pipelines will break

Hardware failures

Increasing volume of data

Upstream delays

Bad code changes

Data pipelines will break

1. Recover fast.
2. Degrade gracefully.

Recover fast

- No long running job.
- Switch from spot to on-demand clusters.
- Increase cluster size.

Recover fast: easy way to rerun jobs

- Needed when jobs run but produce some bad data.
- Not always trivial.

Example: rerun the rollups pipeline

s3://bucket/

- 📂 2018-01
- 📂 2018-02
- 📂 2018-03
- 📂 2018-04
- 📂 2018-05

Example: rerun the rollups pipeline

s3://bucket/2018-05/

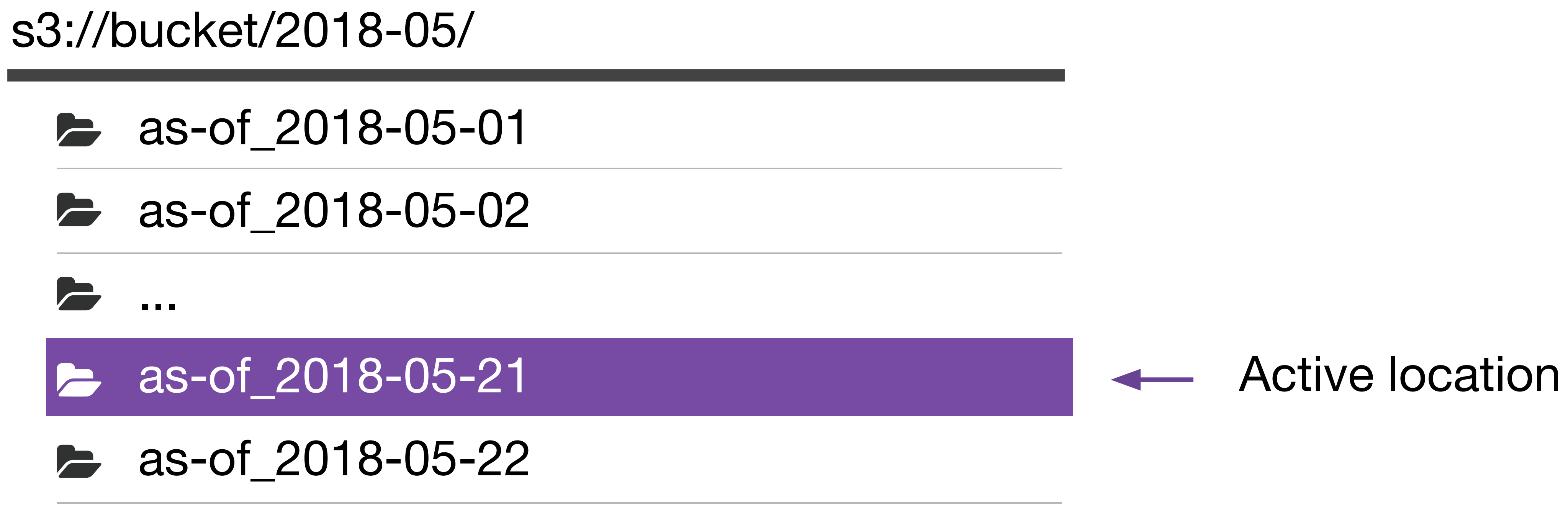
- as-of_2018-05-01
- as-of_2018-05-02
- ...
- as-of_2018-05-21

Example: rerun the rollups pipeline

s3://bucket/2018-05/

- as-of_2018-05-01
- as-of_2018-05-02
- ...
- **as-of_2018-05-21** ← Active location

Example: rerun the rollups pipeline



s3://bucket/2018-05/

- as-of_2018-05-01
- as-of_2018-05-02
- ...
- as-of_2018-05-21 ← Active location
- as-of_2018-05-22

Example: rerun the rollups pipeline

s3://bucket/2018-05/

- 📂 as-of_2018-05-01
- 📂 as-of_2018-05-02
- 📂 ...
- 📂 as-of_2018-05-21
- 📂 as-of_2018-05-22 ← Active location

Example: rerun the rollups pipeline

s3://bucket/2018-05/

- as-of_2018-05-01
- as-of_2018-05-02
- ...
- ✓ **as-of_2018-05-21** ← Active location
- ✗ as-of_2018-05-22

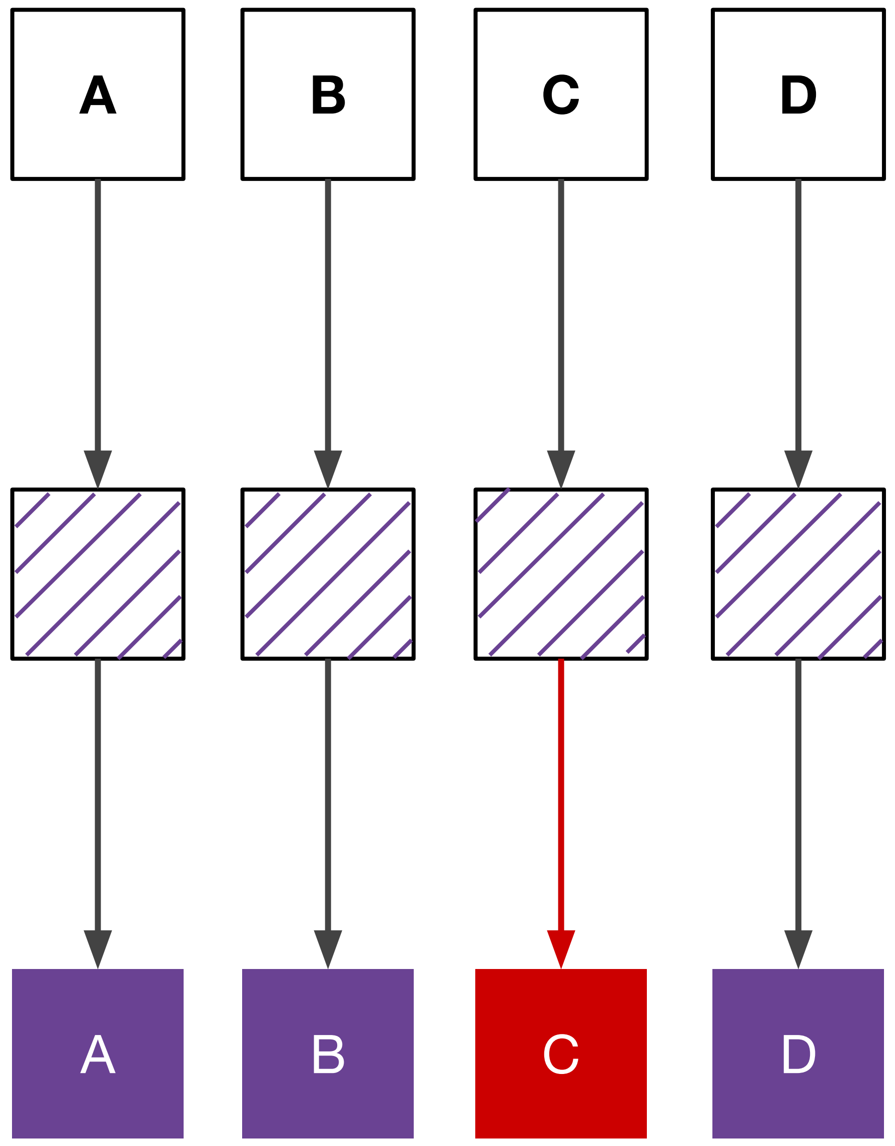
Example: rerun the rollups pipeline

s3://bucket/2018-05/

- as-of_2018-05-01
- as-of_2018-05-02
- ...
- as-of_2018-05-21
- ✗ as-of_2018-05-22
- **as-of_2018-05-22_run-2** ← Active location

Degrade gracefully

- Isolate issues to a limited number of customers.
- Keep the functionalities operational at the cost of performance/accuracy.



Degrade gracefully: skip corrupted files

- Job failure caused by limited corrupted input data.
- Don’t ignore real widespread issues.

Lessons

- Think about potential issues ahead of time.
- Have knobs ready to recover fast.
- Have knobs ready to limit the customer facing impact.

Conclusion

Building highly reliable data pipelines

Conclusion

Building highly reliable data pipelines

- Know your time constraints.

Conclusion

Building highly reliable data pipelines

- Know your time constraints.
- Break down jobs into small survivable pieces.

Conclusion

Building highly reliable data pipelines

- Know your time constraints.
- Break down jobs into small survivable pieces.
- Monitor cluster metrics, job metrics and data lags.

Conclusion

Building highly reliable data pipelines

- Know your time constraints.
- Break down jobs into small survivable pieces.
- Monitor cluster metrics, job metrics and data lags.
- Think about failures ahead of time and get prepared.

Thanks!

We’re hiring!

qf@datadoghq.com
https://jobs.datadoghq.com