Trackunit's successful journey with Apache Flink

Trackunit

Bio

Lasse Nedergaard

System architect and lead developer

Started with Flink 1.2

Worked with other big data projects based on Spark and Mesos.

Build large distributed database systems with Microsoft SQL server.

Trackunit is the world leader for telematics in the construction industry.

And we are hiring!

We provide IoT services for a broad portfolio to optimize the daily operations for our customers

A short history about our pipeline and how it has changed over time

Our Apache Flink history

2016

External consulting started

An external consultant company start to build a real-time streaming pipeline to power our new mobile apps.

Based on Aws and Apache Flink 1.2.

2017

First release ready

Our first release in May.

We upgraded to 1.3.

We began to rebuild our solution

2018

We need more

A number of production and deployment issues.

We consolidated our pipeline.

We added much more functionality.

Moved to DC/OS and Kafka.

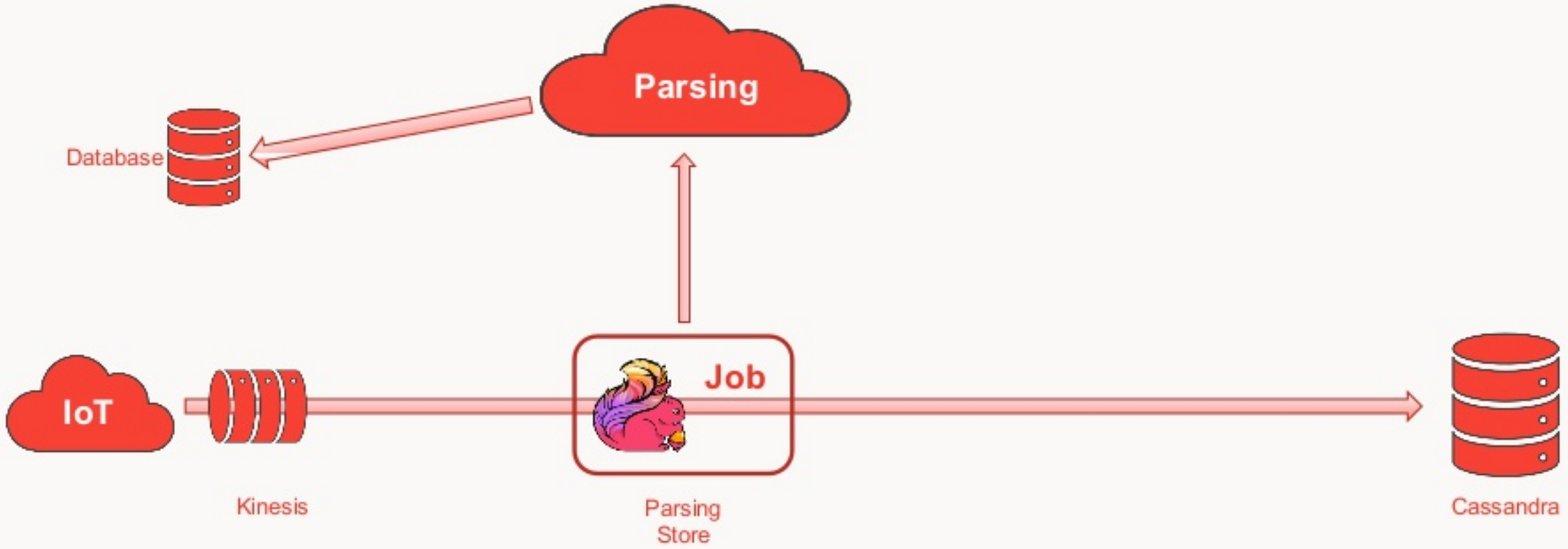
2019

No data loss & Fully auto deployed

Everything scripted and auto deployed.

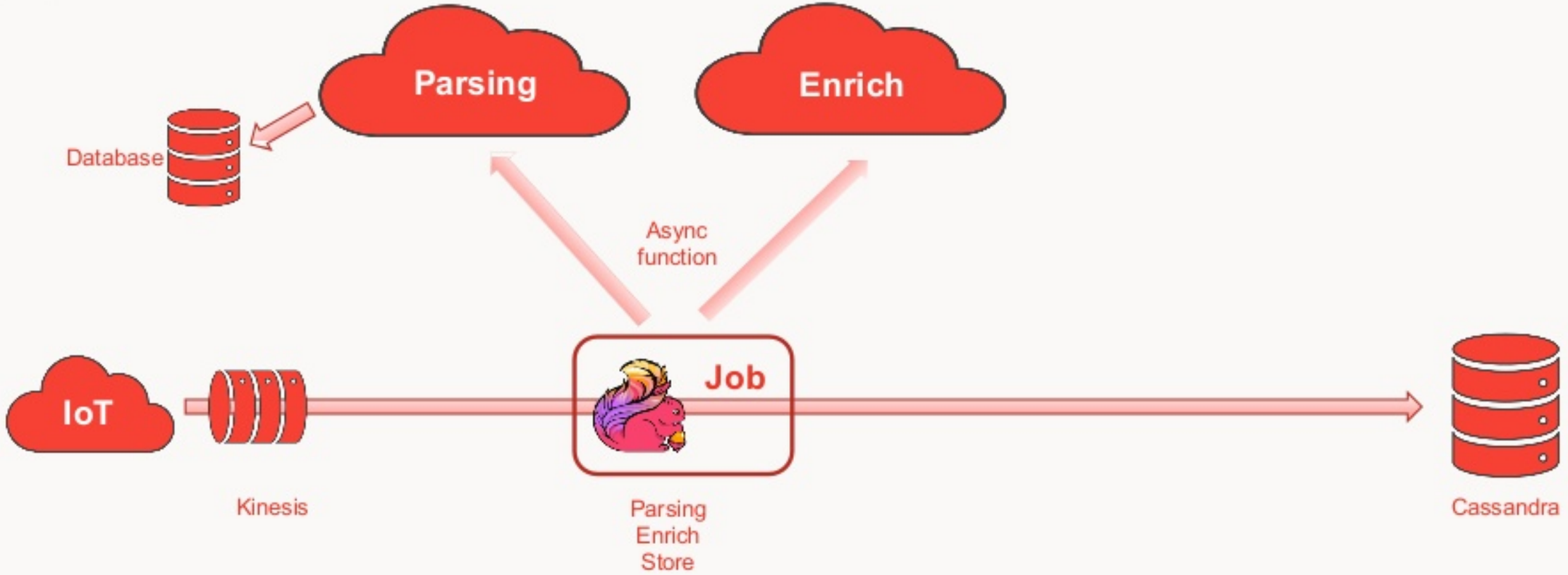
No data loss even in case of structural changes.

More services.


Parsing
Database
IoT
Job
Kinesis
Parsing
Store
Cassandra

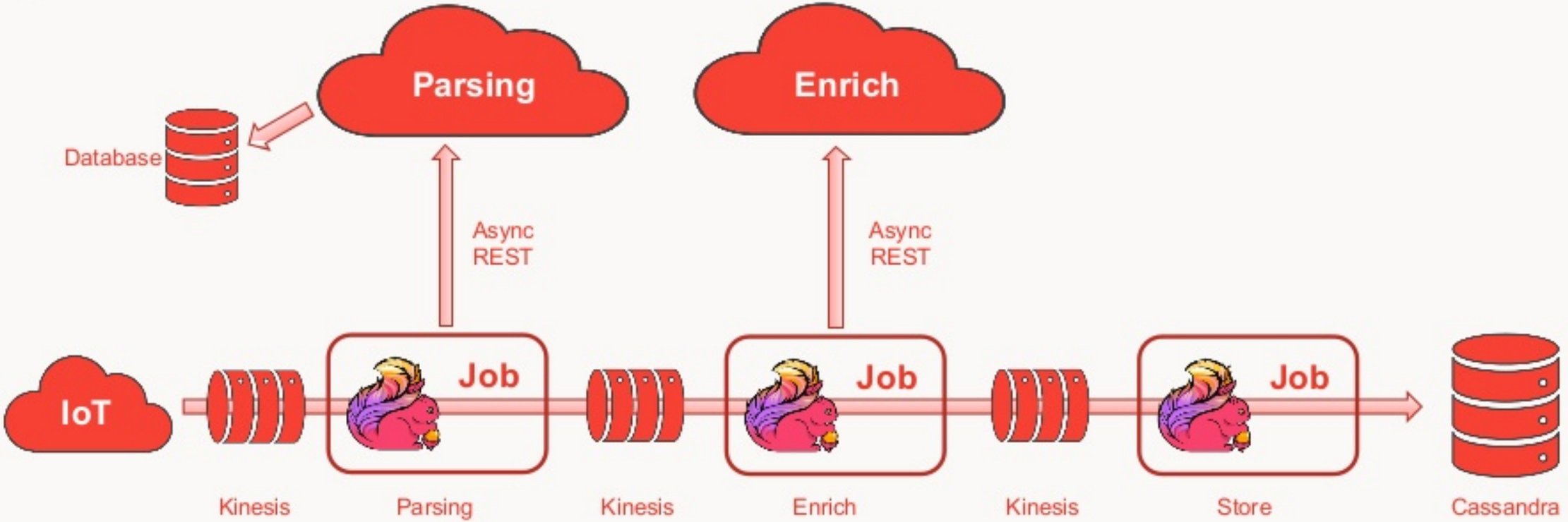
- *Delivered in time.*
- *It worked.*

- *Our overall system lacked performance because of data enrichment at query time.*
- *Reuse of old code base.*


Parsing
Enrich
Database
Async
function
IoT
Job
Kinesis
Parsing
Enrich
Store
Cassandra

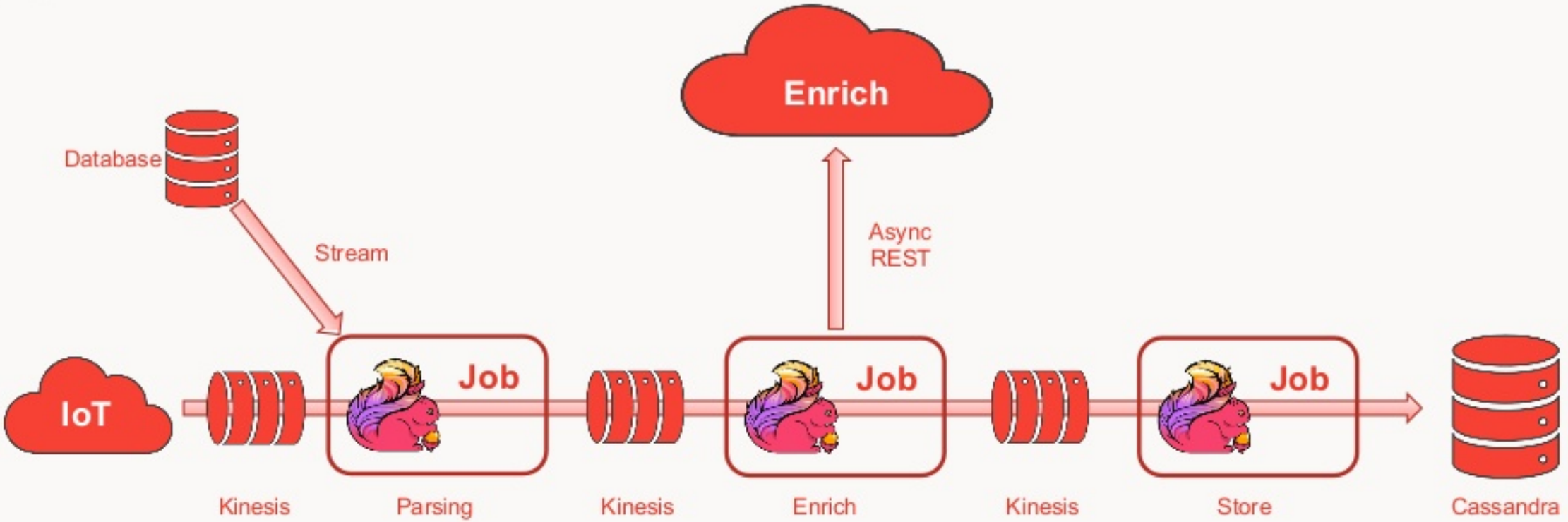
- *Our overall system performance is fine.*

- *Async enrichment caused problems and is nice to have.*
- *Everything was done in one job.*


Parsing
Enrich
Database
Async
REST
Async
REST
IoT
Job
Job
Job
Kinesis
Parsing
Kinesis
Enrich
Kinesis
Store
Cassandra

- *One job now does one thing.*
- *Future-proof for adding more functionality.*

- *Limited Flink throughput.*
- *Increased complexity.*
- *Our Yarn session sometimes dies?*

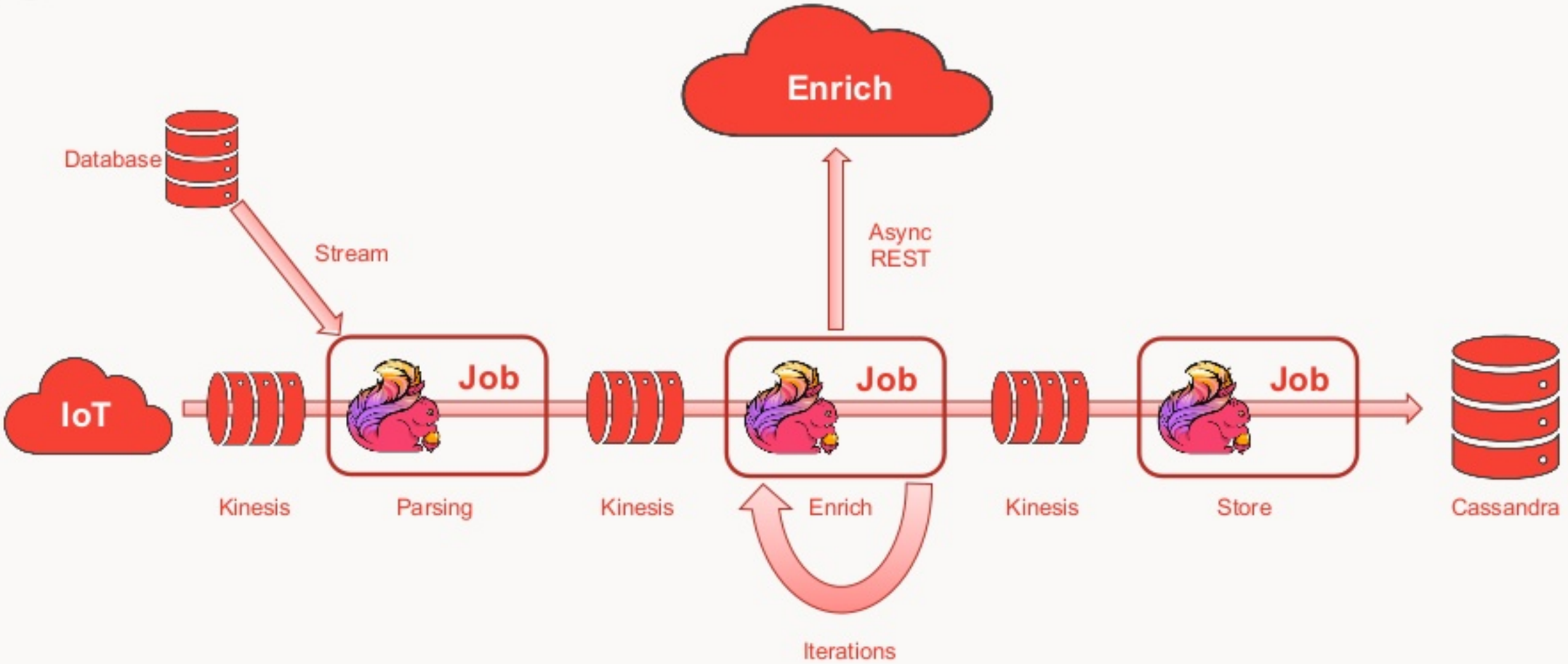

Enrich
Database
Stream
Async
REST
IoT
Job
Job
Job
Kinesis
Parsing
Kinesis
Enrich
Kinesis
Store
Cassandra

Building our own SQL server data source

- Simple by extending RichSourceFunction<T>
- Poll strategy with timestamp
- Store last poll time in state
- Implement ListCheckpointed
- Context.collect<T>

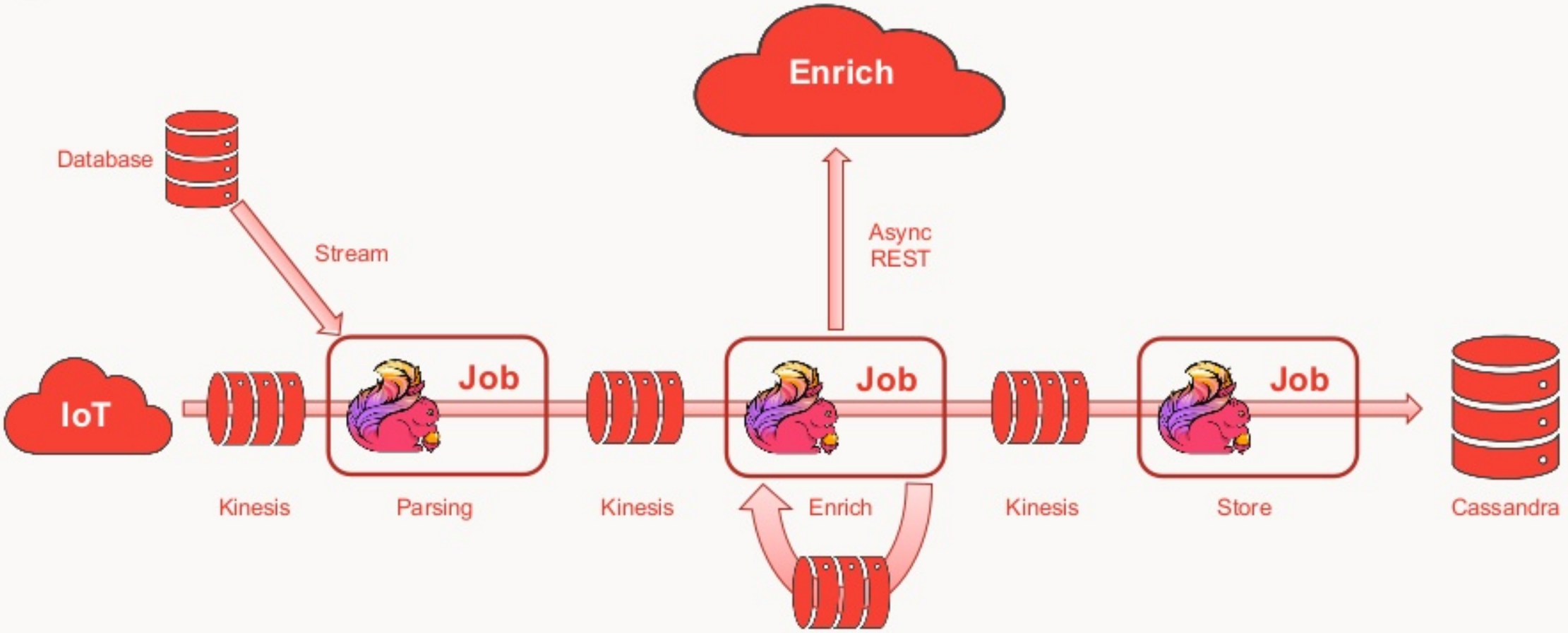
- *Parsing is now fast.*
- *Legacy code has been removed.*

- *Downstream job failed.*
- *Async enrichment caused problems and is nice to have.*


Enrich
Database
Stream
Async
REST
IoT
Job
Job
Job
Kinesis
Parsing
Kinesis
Enrich
Kinesis
Store
Cassandra
Iterations

- *Reduced Async call with 33%*

- *Problems with state TTL (fixed in 1.6)*
- *Caused deadlock because of back pressure ;-)*


Enrich
Database
Stream
Async
REST
IoT
Job
Job
Job
Kinesis
Parsing
Kinesis
Enrich
Kinesis
Store
Cassandra

- *Now it worked.*

- *Still having Async problems.*
- *Problems with catchup as it kills downstream tasks.*
- *Problems with catchup as we read too fast !!!*

Why is the Yarn session dying ????

21/06/2018 08:38:35.112	2018-06-21 06:38:35,112 INFO org.apache.flink.yarn.YarnTaskManagerRunnerFactory - RECEIVED SIGNAL 15: SIGTERM. Shutting do	
	host = ElasticMapReduce-slave_10.1.1.112	source = /var/log/hadoop-yarn/containers/application_1529255981579_0001/container_152925598... sourcetype = taskmanager
21/06/2018 08:37:40.601	2018-06-21 06:37:40,601 INFO org.apache.flink.yarn.YarnTaskManagerRunnerFactory - RECEIVED SIGNAL 15: SIGTERM. Shutting do	
	host = ElasticMapReduce-slave_10.1.1.112	source = /var/log/hadoop-yarn/containers/application_1529255981579_0001/container_152925598... sourcetype = taskmanager
21/06/2018 08:36:57.518	2018-06-21 06:36:57,518 INFO org.apache.flink.yarn.YarnTaskManagerRunnerFactory - RECEIVED SIGNAL 15: SIGTERM. Shutting do	
	host = ElasticMapReduce-slave_10.1.1.192	source = /var/log/hadoop-yarn/containers/application_1529255981579_0001/container_152925598... sourcetype = taskmanager

From Flink documentation.

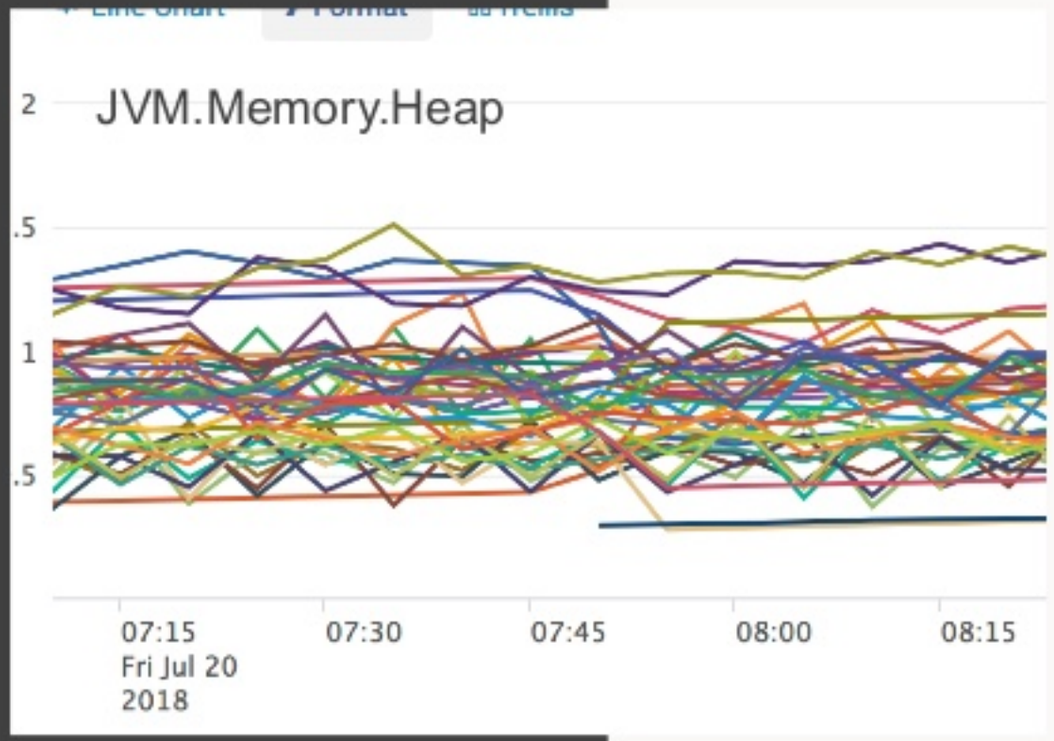
https://ci.apache.org/projects/flink/flink-docs-release-1.5/ops/deployment/yarn_setup.html

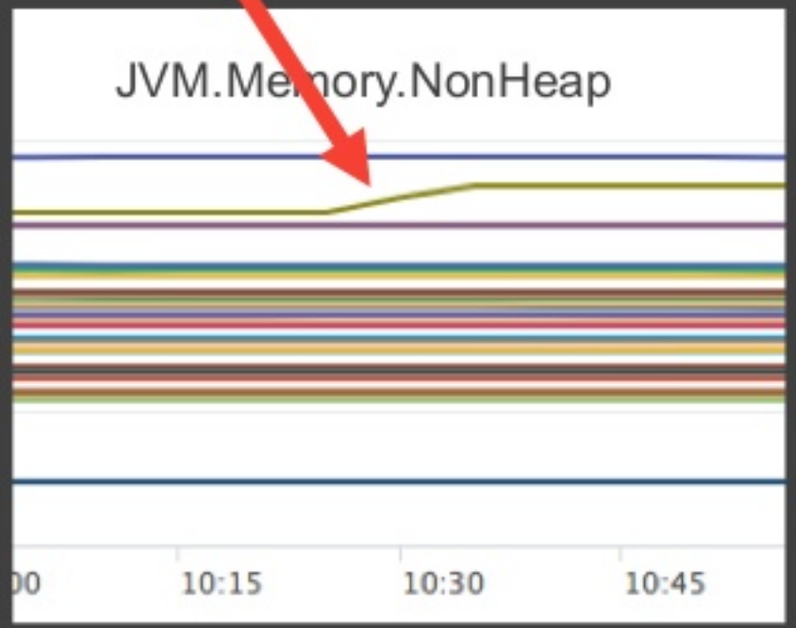
yarn.maximum-failed-containers:
The maximum number of failed containers the ApplicationMaster accepts until it fails the YARN session.
Default: The number of initially requested TaskManagers (-n).

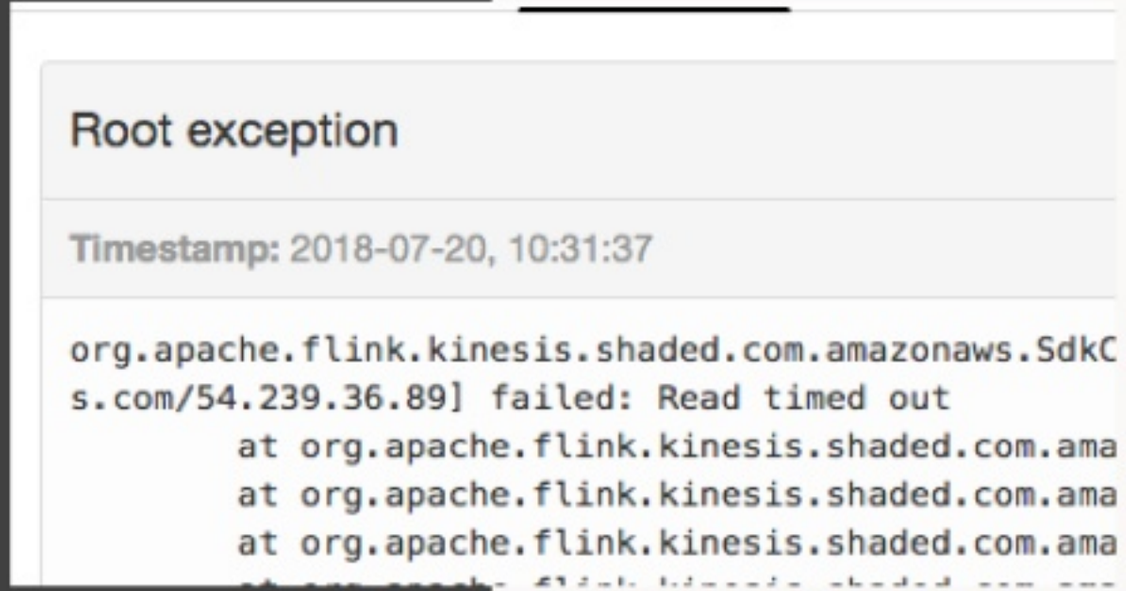
Why are task managers terminated ????

Let’s take a look:

We see a pattern here







Flink handles exceptions with restarts fine.

But it will kill your YARN session over time!

Currently we don’t know why this happens, but we investigate it together with the community.

RocksDb state backend is under suspicion.

Async IO

Async IO timeout exception, why?

- Is not call in a multi-threaded fashion. We have to do it.
- We used an executor service instead of futures.
- We misunderstood timeout and capacity.

Wrong Async IO config

- Our intial configuration
  - Executor service pool size set to 25
  - Capacity set to 10000
  - Timeout set to max REST read timeout (2 sec) + 2 sec.
  - REST connect timeout 350 ms. read timeout 2000 ms.
- =>
  - 10000/25 = 400 msg/thread in queue
  - Avg. max 10 ms to process a message if capacity is max out
  - Normal load 3 msg/thread
  - 333 ms to process a message

- Capacity is max out if
  - Catching up
  - REST service answer slow

Correct Async IO config

- Correct configuration
  - Executor service pool size set to 200.
  - Capacity set to 5000
  - Timeout set to (50 sec) + 2 sec.
  - REST connect timeout 350 ms. read timeout 2000 ms.
- =>
  - 5000/200 = 25 msg/thread
  - Timeout = 25 * 2 sec = 50 sec.

Async IO config the right way

- Use Future instead of executor service
- Use capacity correct as it tells how many calls to handle at the same time
- Set timeout to max Async IO call can process

Throttling

What happens when catching up?

- If many jobs were catching up as fast as possible, the read and write throughput against Kinesis and write throughput for Cassandra exceeded the limitations
- For Kinesis: Read/Write exceeded exception
- For Cassandra: Quorum exceptions during high CPU load

- Kinesis can read 2 times faster than write
- Write Exceeded => Exception and not back pressure

- High load in Cassandra => Quorum exception and not back pressure

Solution

- Throttling read
- The maximum number of records to try to get each time we fetch records from a AWS Kinesis shard
- SHARD_GETRECORDS_MAX = **"flink.shard.getrecords.maxrecordcount"**

- *The interval between each getRecords request to a AWS Kinesis shard in milliseconds*
  SHARD_GETRECORDS_INTERVAL_MILLIS = **"flink.shard.getrecords.intervalmillis"**

- The maximum number of getRecords attempts if we get ProvisionedThroughputExceededException
- SHARD_GETRECORDS_RETRIES = **"flink.shard.getrecords.maxretries"**

- **See https://docs.aws.amazon.com/streams/latest/dev/service-sizes-and-limits.html**

Solution

- Test your pipeline with max load:
  - To avoid surprises
  - To detect bottlenecks in your system
  - To have metrics for everything
  - To sleep without any concerns ;)

- Throttle sources
  - To ensure all downstream systems don’t break
- Contribute back to the community back pressure implementations for sinks

Flink On EMR

Job restart fails with “blob server file not found exception”?

- We used EMR on Amazon’s Linux AMI
- We didn't change the default blob server location (/tmp)
- Default a cron job cleaning up in /tmp

- Solution change blob server location with **blob.storage.directory**

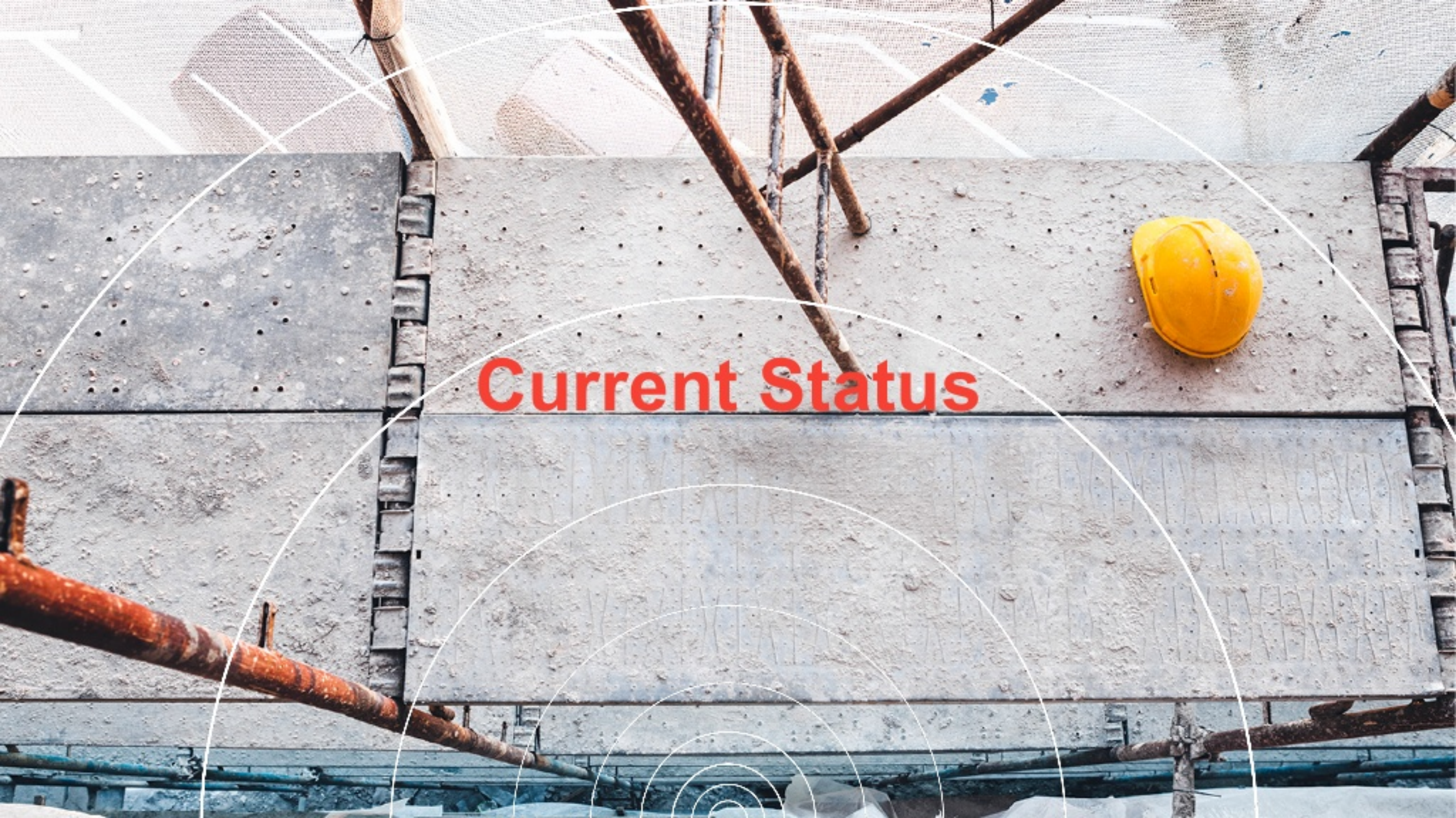
Yarn on EMR failed to start?

- We stored savepoints on HDFS
- We created backup to restore from prev. state
  - but forgot to clean up
- => out of disk space

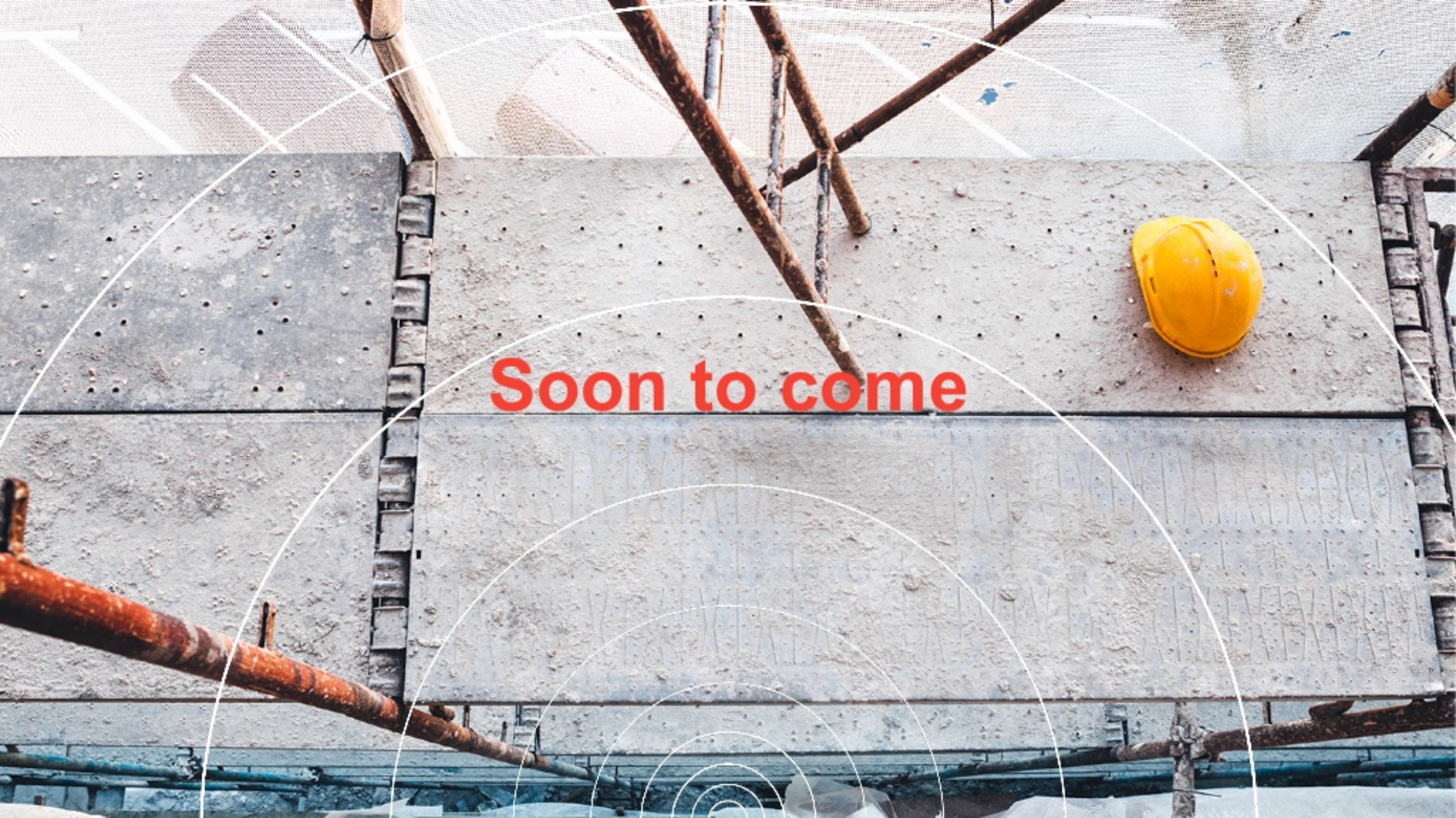
- Metrics was implemented as log reporter
- In debug level
- Stream.print output to debug level
- => all data in log file
- => out of disk space

Scaling Flink before 1.5

- We have to redeploy the Yarn session
- Redeploy all jobs
- Delay in data processing
- Catch up


Current Status

- Running Flink 1.6 on DC/OS (mesos)
  - Dynamic task allocation
  - Separate Flink and Flink savepoints
- Replaced Kinesis with Kafka
  - Don’t have to deal with commercial limitations
- Metrics for everything
  - Use Promethues and Grafana


Soon to come

- Flink desired state
  - Automatic tool to upload and upgrade Flink jobs
- Many new applications build on Flink
  - To deliver new value faster to the business
- More metrics
  - To get deeper insights

01

Avoid job restarts.

As it can cause container termination.

02

Metrics.

Without you are blind. It's the best way to monitor your solution and start debugging.

03

Automate everything.

Without automation you make mistakes and it helps you during the nighttime.

04

Prepare for worst case.

It will happen one day.

05

Test and test worst case.

Automated test but also test catchup scenarios and know what your system can handle.

06

Logging.

Collect logs from Flink and the eco-system around it. Together with metrics you have a chance to find out what's going on.

"

At Trackunit we are not stupid, but we didn't know better at the time.

We do now!

Lasse Nedergaard