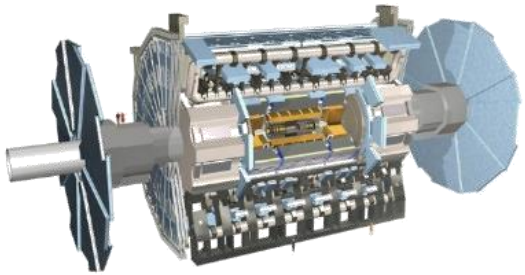
Monitor Apache Spark 3 on Kubernetes using Metrics and Plugins

Luca Canali

Data Engineer, CERN

About Luca



- Data Engineer at CERN
  - Data analytics and Spark service, database services
  - 20+ years with databases and data engineering
  - Passionate about performance engineering

- Repos, blogs, presentations

@LucaCanaliDB
https://github.com/lucacanali
http://cern.ch/canali

Agenda

- Apache Spark monitoring: ecosystem and motivations
- Spark metrics system
- Spark 3 plugins
- Metrics for Spark on K8S and cloud storage
- How you can run a Spark performance dashboard

Performance Troubleshooting Goals

- The key to good performance:
  - You run good execution plans
  - There are no serialization points
  - Without these all bets are off!
    - Attribution: I first heard this from Andrew Holdsworth (context: Oracle DB performance discussion ~2007)

- Spark users care about performance at scale
  - Investigate execution plans and bottlenecks in the workload
  - Useful tools are the Spark Web UI and metrics instrumentation!

Apache Spark Monitoring Ecosystem

- Web UI
  - Details on jobs, stages, tasks, SQL, streaming, etc
  - Default URL: http://driver:4040
  - https://spark.apache.org/docs/latest/web-ui.html

- Spark REST API + Spark Listener @Developer API
  - Exposes task metrics and executor metrics
  - https://spark.apache.org/docs/latest/monitoring.html

- Spark Metrics System
  - Implemented using the Dropwizard metrics library

Spark Metrics System

- Many metrics instrument Spark workloads:
  - https://spark.apache.org/docs/latest/monitoring.html#metrics
  - Metrics are emitted from the driver, executors and other Spark components into sinks (several sinks available)
  - Example of metrics: number of active tasks, jobs/stages completed and failed, executor CPU used, executor run time, garbage collection time, shuffle metrics, I/O metrics, metrics with memory usage details, etc.

Explore Spark Metrics

- Web UI Servlet Sink
  - Metrics from the driver + executor (executor metrics available only if Spark is in local mode)
  - By default: metrics are exposed using the metrics servlet on the WebUI http://driver_host:4040/metrics/json
  - Prometheus sink for the driver and for spark standalone, exposes metrics on the Web UI too
    - *.sink.prometheusServlet.class=org.apache.spark.metrics.sink.PrometheusServlet
    - *.sink.prometheusServlet.path=/metrics/prometheus
- Jmx Sink
  - Use jmx sink and explore metrics using jconsole
  - Configuration: *.sink.jmx.class=org.apache.spark.metrics.sink.JmxSink

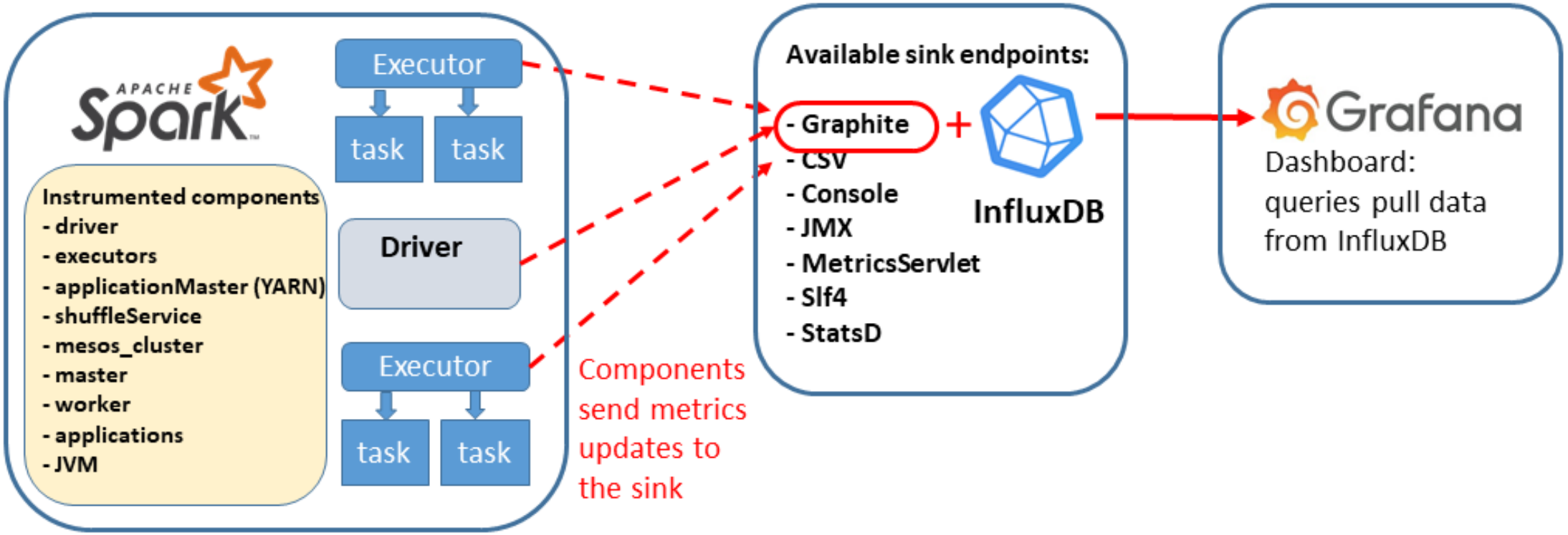
Spark Metrics System and Monitoring Pipeline

Apache Spark Metrics System + InfluxDB + Grafana => Dashboard

Source: Spark components instrumented with dropwizard library **metrics**

Sink: collect and **store** the metrics using InfluxDB

Visualize: using Grafana dashboards



How to Use Spark Metrics – Sink to InfluxDB

- Edit $SPARK_HOME/conf/metrics.properties

```
cat $SPARK_HOME/conf/metrics.properties
*.sink.graphite.class"="org.apache.spark.metrics.sink.GraphiteSink"
*.sink.graphite.host"="<graphiteEndPoint_influxDB_hostName>"
*.sink.graphite.port"=<graphite_listening_port>
*.sink.graphite.period"=10
*.sink.graphite.unit"=seconds
*.sink.graphite.prefix"="lucatest"
*.source.jvm.class"="org.apache.spark.metrics.source.JvmSource"
```

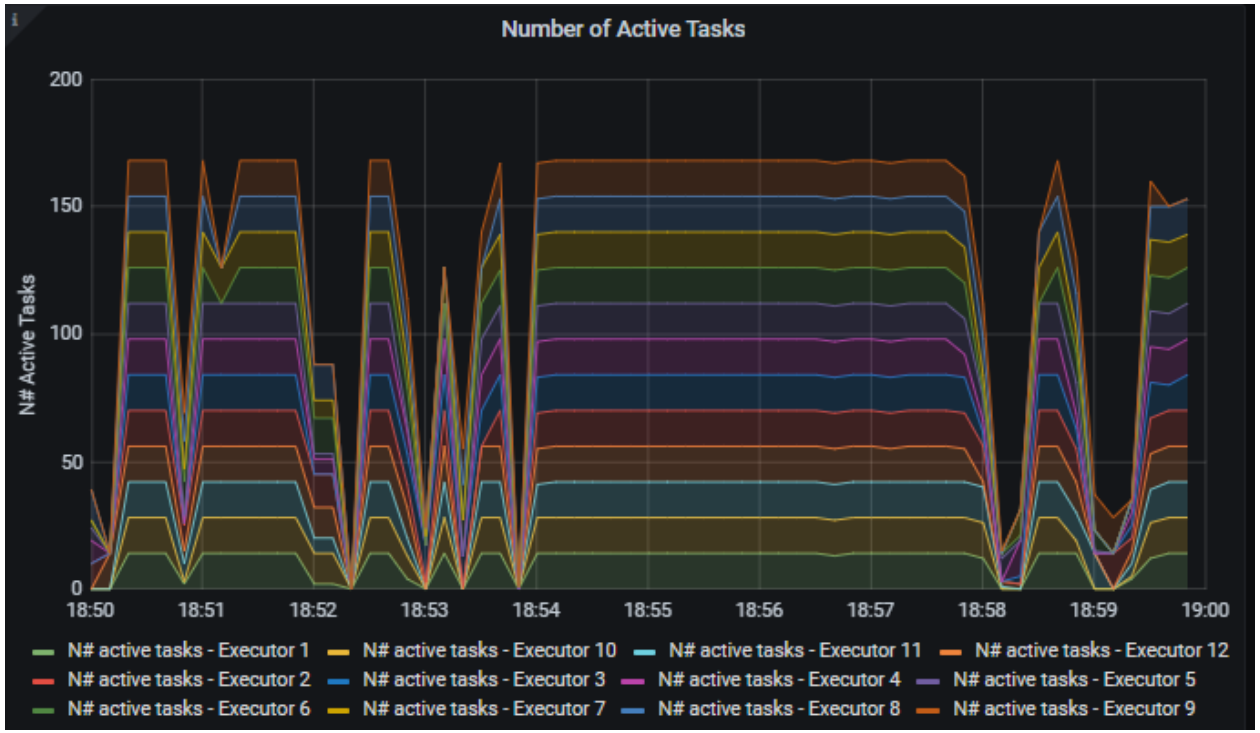
- Alternative: use the config parameters spark.metrics.conf.*
  - Use this method if running on K8S on Spark version 2.x or 3.0, see also SPARK-30985

```
$SPARK_HOME/bin/spark-shell \
--conf "spark.metrics.conf.*.sink.graphite.class"="org.apache.spark.metrics.sink.GraphiteSink" \
--conf "spark.metrics.conf.*.sink.graphite.host"="<graphiteEndPoint_influxDB_hostName>" \
..etc..
```

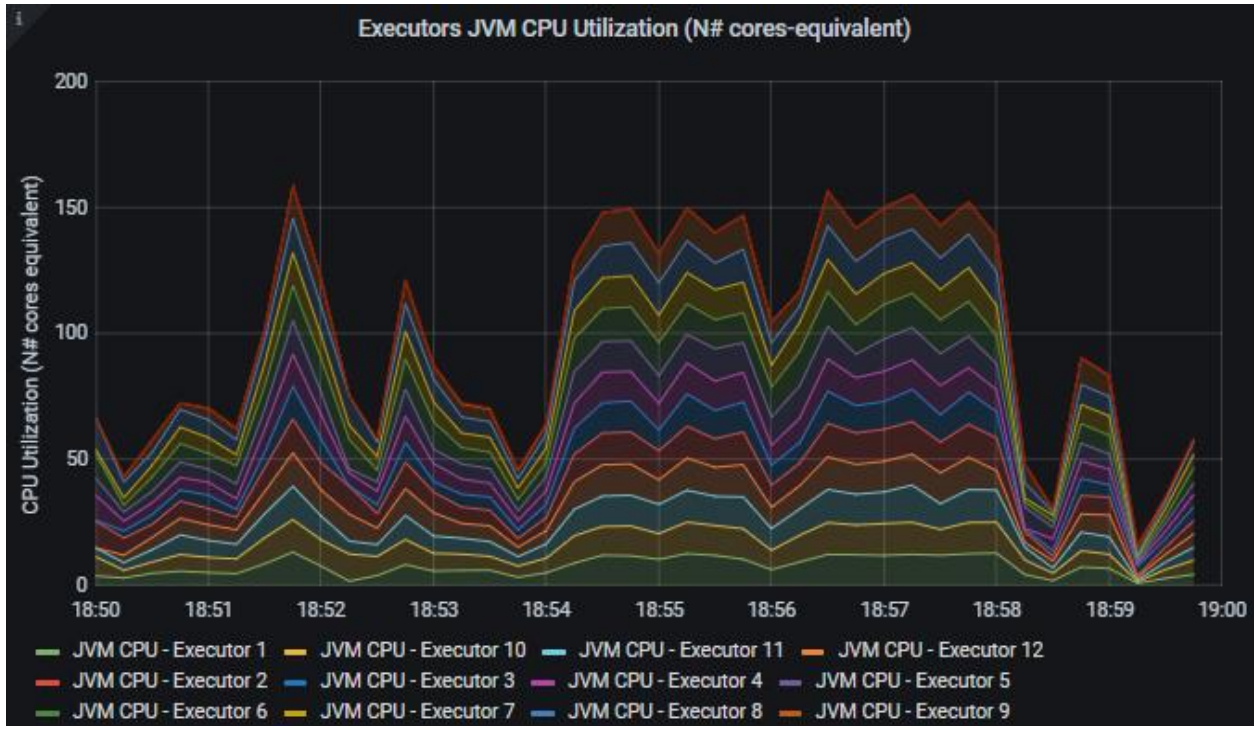
Metrics Visualization with Grafana

- Metrics visualized as time series
  - Metrics values visualized vs. time and detailed per executor

Number of Active Tasks



Executor JVM CPU Usage



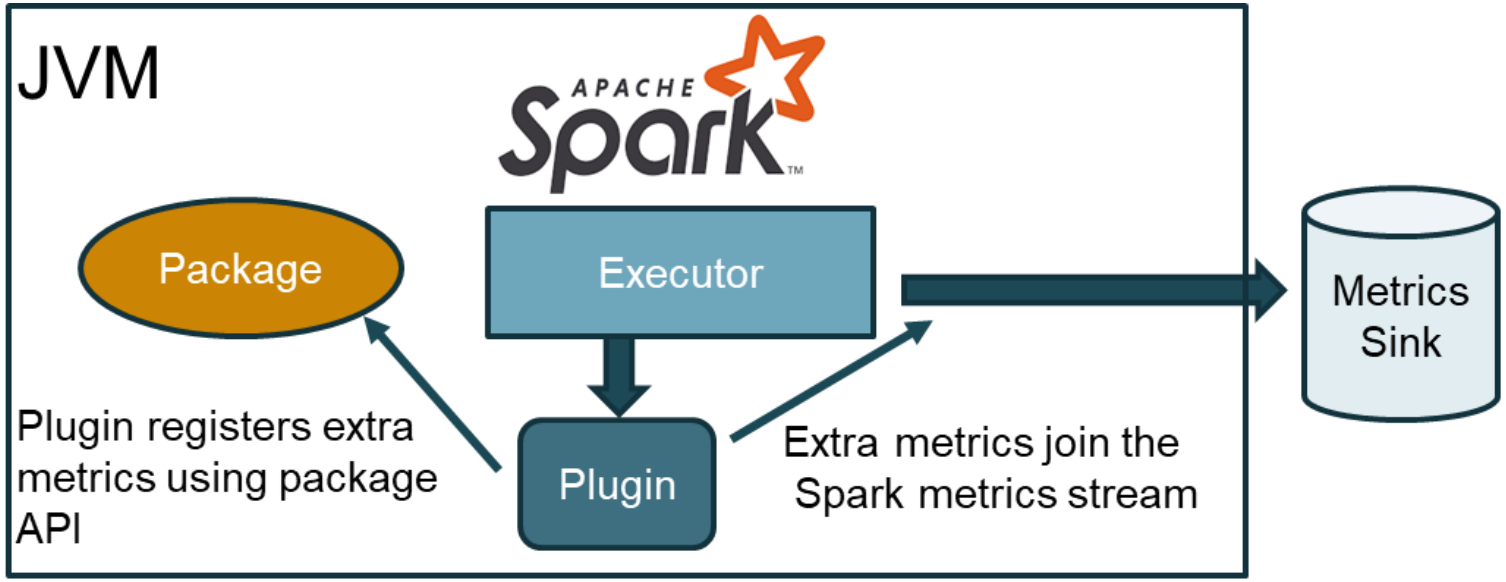
Metrics for Spark on Kubernetes

- Spark on Kubernetes is GA since Spark 3.1.1
  - Running Spark on cloud resources is widely used
  - Need for improved instrumentation in Spark

- Spark 3 metrics system can be extended with plugins
  - Plugin metrics to monitor K8S pods' resources usage (CPU, memory, network, etc)
  - Plugin instrumentation for cloud filesystems: S3A, GS, WASBS, OCI, CERN's EOS, etc)

Spark Plugins and Custom Extensions to the Metrics System

- Plugin interface introduced in Spark 3
  - Plugins are user-provided code run at the start of the executors (and of the driver)
  - Plugins allow to extend Spark metrics with custom code and instrumentation



A First Look at the Spark Plugins API

- Code snippets for demonstration
  - A Spark Plugin implementing a metric that reports the constant value 42

```
import com.codahale.metrics.{Gauge, MetricRegistry}
import org.apache.spark.api.plugin.{DriverPlugin, ExecutorPlugin, PluginContext, SparkPlugin}
import org.apache.spark.SparkContext

class DemoMetricsPlugin extends SparkPlugin {

// Return the plugin's executor-side component.
override def executorPlugin(): ExecutorPlugin = {
  new ExecutorPlugin {
    override def init(myContext:PluginContext, extraConf:JMap[String, String]): Unit  = {
      // Gauge for testing
      val metricRegistry = myContext.metricRegistry
      metricRegistry.register(MetricRegistry.name( name = "ExecutorTest42"), new Gauge[Int] {
        override def getValue: Int = 42
        })
    }
  }
}
```

Spark 3.x Plugin API

metricRegistry allows to register user-provided sources with Spark Metrics System

‘Kick the Tires’ of Spark Plugins

- From https://github.com/cerndb/SparkPlugins
  - RunOSCommandPlugin - runs an OS command as the executor starts
  - DemoMetricsPlugin - shows how to integrate with Spark Metrics

```
bin/spark-shell --master k8s://https://<K8S URL> \
  --packages ch.cern.sparkmeasure:spark-plugins_2.12:0.1 \
  --conf spark.plugins=ch.cern.RunOSCommandPlugin,\
  ch.cern.DemoMetricsPlugin
```

Plugins for Spark on Kubernetes

- Measure metrics related to pods' resources usage
  - Integrated with rest of Spark metrics
  - Plugin code implemented using cgroup instrumentation
  - Example: measure CPU from `/sys/fs/cgroup/cpuacct/cpuacct.usage`

```
bin/spark-shell --master k8s://https://<K8S URL> \
--packages ch.cern.sparkmeasure:spark-plugins_2.12:0.1 \
--conf spark.kubernetes.container.image=<registry>/spark:v311 \
--conf spark.plugins=ch.cern.CgroupMetrics \
...
```

CgroupMetrics Plugin

- Metrics (gauges), in ch.cern.CgroupMetrics plugin:
  - CPUTimeNanosec: CPU time used by the processes in the cgroup
    - this includes CPU used by Python processes (PySpark UDF)
  - MemoryRss: number of bytes of anonymous and swap cache memory.
  - MemorySwap: number of bytes of swap usage.
  - MemoryCache: number of bytes of page cache memory.
  - NetworkBytesIn: network traffic inbound.
  - NetworkBytesOut: network traffic outbound.

Plugin to Measure Cloud Filesystems

- Example of how to measure S3 throughput metrics
  - Note: Apache Spark instruments only HDFS and local filesystem
  - Plugins uses Hadoop client API for Hadoop Compatible filesystems
  - Metrics: bytesRead, bytesWritten, readOps, writeOps

```
--conf spark.plugins=ch.cern.CloudFSMetrics
--conf spark.cernSparkPlugin.cloudFsName=<name of the filesystem>
 (example: "s3a", "gs", "wasbs", "oci", "root", etc.)
```

Tooling for a Spark Performance Dashboard

- Code and examples to get started:
  - https://github.com/cerndb/spark-dashboard
  - It simplifies the configuration of InfluxDB as a sink for Spark metrics
  - Grafana dashboards with pre-built panels, graphs, and queries
- Option 1: Dockerfile
  - Use this for testing locally
  - From dockerhub: `lucacanali/spark-dashboard:v01`
- Option 2: Helm Chart
  - Use for testing on Kubernetes

Tooling for Spark Plugins

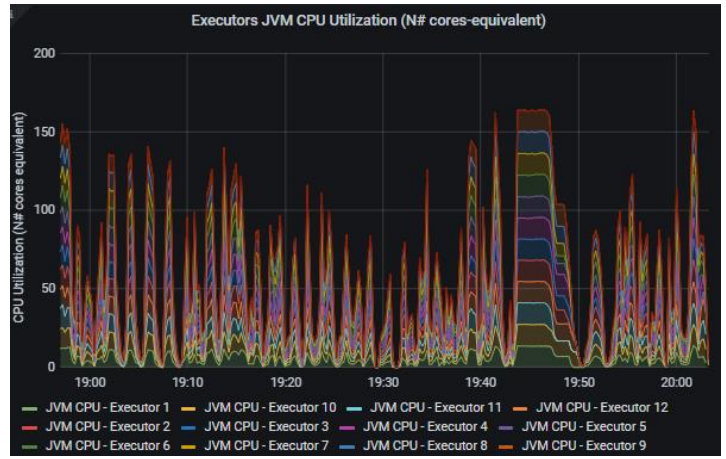
- Code and examples to get started:
  - https://github.com/cerndb/SparkPlugins
  - **`--packages ch.cern.sparkmeasure:spark-plugins_2.12:0.1`**
- Plugins for OS metrics (Spark on K8S)
  - `--conf spark.plugins=`**`ch.cern.CgroupMetrics`**
- Plugins for measuring cloud storage
  - Hadoop Compatible Filesystems: s3a, gs, wasbs, oci, root, etc..
  - `--conf spark.plugins=`**`ch.cern.CloudFSMetrics`**
  - `--conf spark.cernSparkPlugin.cloudFsName=<filesystem name>`

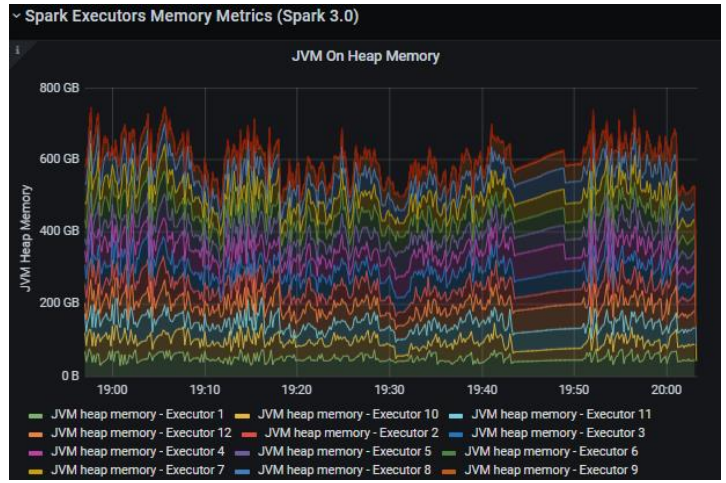
How to Deploy the Dashboard - Example

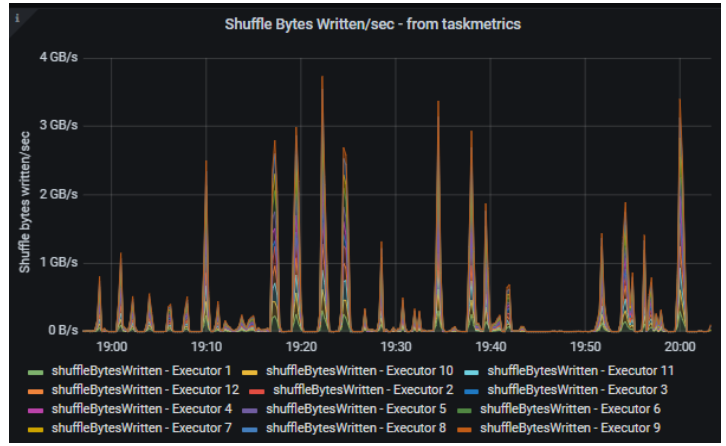
```
docker run --network=host -d lucacanali/spark-dashboard:v01

INFLUXDB_ENDPOINT=`hostname`

bin/spark-shell --master k8s://https://<K8S URL> \
   --packages ch.cern.sparkmeasure:spark-plugins_2.12:0.1 \
   --conf spark.plugins=ch.cern.CgroupMetrics \
   --conf "spark.metrics.conf.*.sink.graphite.class"=
   "org.apache.spark.metrics.sink.GraphiteSink" \
   --conf "spark.metrics.conf.*.sink.graphite.host"=$INFLUXDB_ENDPOINT \
   --conf "spark.metrics.conf.*.sink.graphite.port"=2003 \
   --conf "spark.metrics.conf.*.sink.graphite.period"=10 \
   --conf "spark.metrics.conf.*.sink.graphite.unit"=seconds \
   --conf "spark.metrics.conf.*.sink.graphite.prefix"="luca“ \
   --conf spark.metrics.appStatusSource.enabled=true
   ...
```

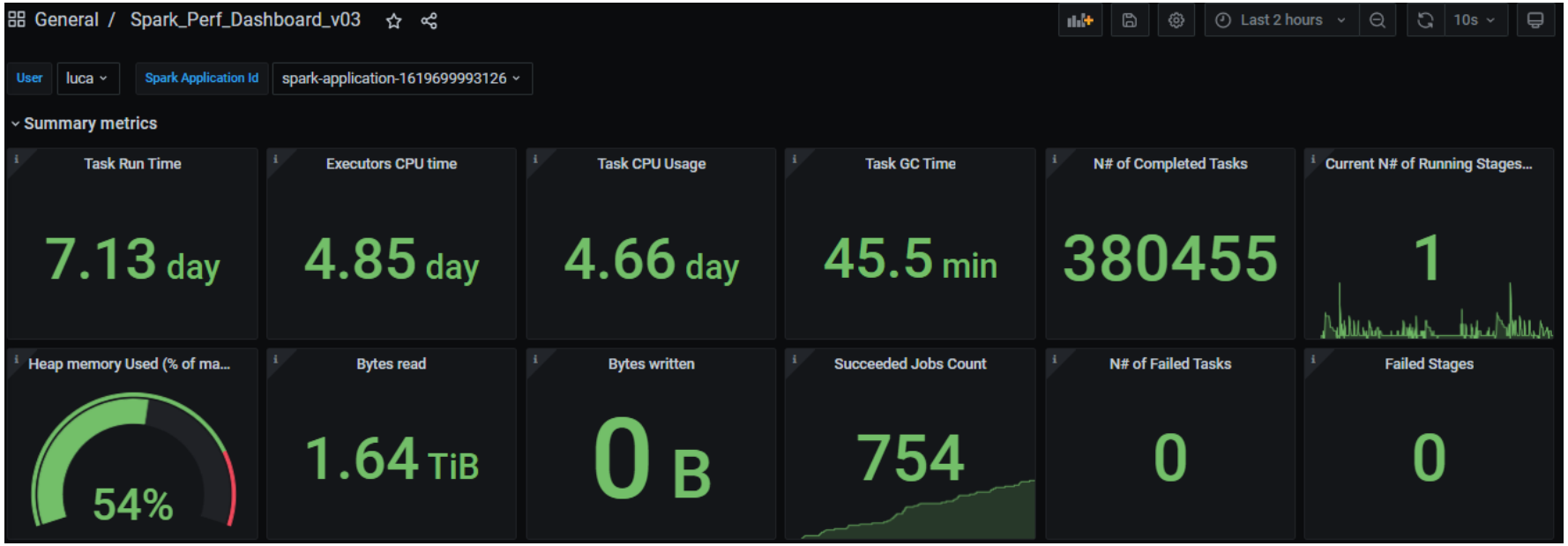
Spark Performance Dashboard

- Visualize Spark metrics
  - Real-time + historical data
  - Summaries and time series of key metrics
  - Data for root-cause analysis







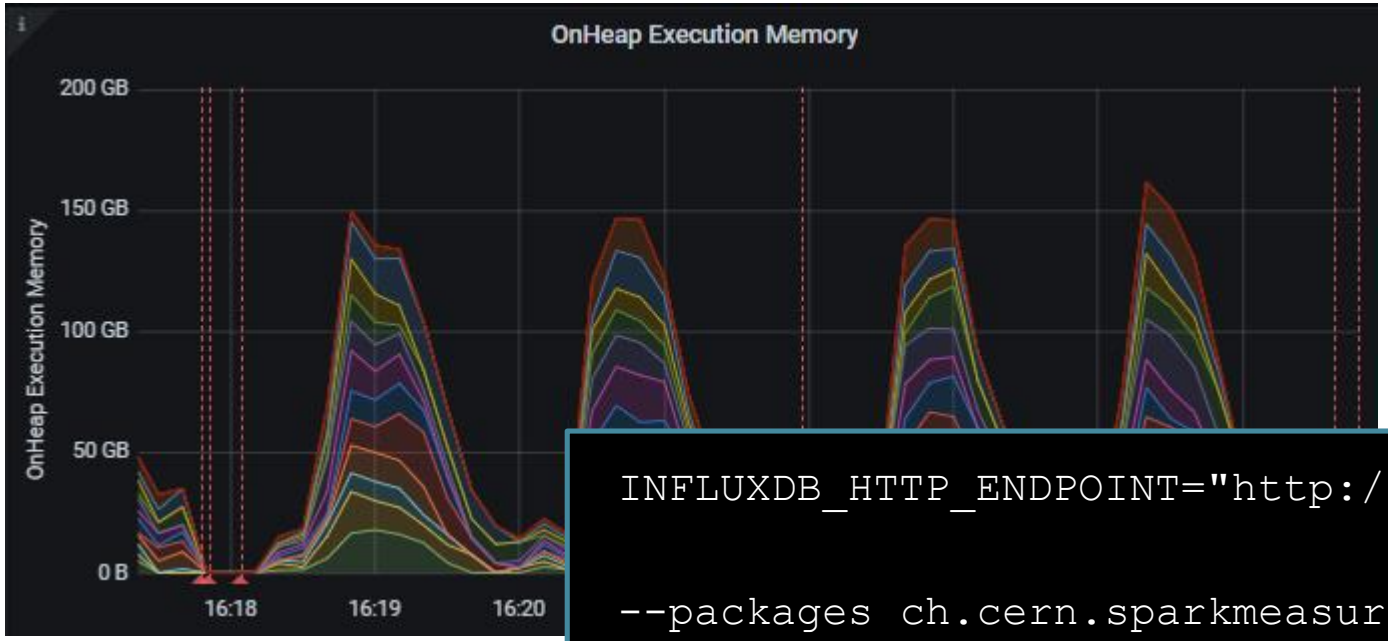


Dashboard Annotations

- Annotations to the workload graphs with:
- Start/end time for job id, or stage id, or SQL id

Dashboard Annotations

- Annotations to the workload graphs with:
- Start/end time for job id, or stage id, or SQL id




```
INFLUXDB_HTTP_ENDPOINT="http://`hostname`:8086"

--packages ch.cern.sparkmeasure:spark-measure_2.12:0.17 \
--conf spark.sparkmeasure.influxdbURL=$INFLUXDB_HTTP_ENDPOINT \
--conf spark.extraListeners=ch.cern.sparkmeasure.InfluxDBSink \
```

Demo – Spark Performance Dashboard (5 min)

Use Plugins to Instrument Custom Libraries

- Augment Spark metrics with metrics from custom code
  - Example: experimental way to measure I/O time with metrics
  - Measure read time, seek time and CPU time during read operations for HDFS, S3A, etc.
    - Custom S3A jar with time instrumentation (for Hadoop 3.2.0, use with Spark 3.0 and 3.1): https://github.com/LucaCanali/hadoop/tree/s3aAndHDFSTimeInstrumentation
    - Metrics: S3AReadTimeMuSec, S3ASeekTimeMuSec, S3AGetObjectMetadataMuSec
    - Spark metrics with custom Spark plugin:
      - `--packages ch.cern.sparkmeasure:spark-plugins_2.12:0.1`
      - `--conf spark.plugins=ch.cern.experimental.S3ATimeInstrumentation`

Lessons Learned and Further Work

- Feedback on deploying the dashboard @CERN
  - We provide the Spark dashboard as an optional configuration for the CERN Jupyter-based data analysis platform
  - Cognitive load to understand the available metrics and troubleshooting process
  - We set data retention and in general pay attention not to overload InfluxDB
- Core Spark development
  - A native InfluxDB sink would be useful (currently Spark has a Graphite sink)
  - Spark is not yet fully instrumented, a few areas yet to cover, for example instrumentation of I/O time and Python UDF run time.

Conclusions

- Spark metrics and dashboard
  - Provide extensive performance monitoring data. Complement the Web UI.
- Spark plugins
  - Introduced in Spark 3.0, make it easy to augment the Spark metrics system.
  - Use plugins to monitor Spark on Kubernetes and cloud filesystem usage.
- Tooling
  - Get started with a dashboard based on InfluxDB and Grafana.
  - Build your plugins and dashboards, and share your experience!

Acknowledgments and Links

- Thanks to CERN data analytics and Spark service team
- Thanks to Apache Spark committers and Spark community
  - For their help with PRs: SPARK-22190, SPARK-25170, SPARK-25228, SPARK-26928, SPARK-27189, SPARK-28091, SPARK-29654, SPARK-30041, SPARK-30775, SPARK-31711

- Links:
  - https://github.com/cerndb/SparkPlugins
  - https://github.com/cerndb/spark-dashboard
  - https://db-blog.web.cern.ch/blog/luca-canali/2019-02-performance-dashboard-apache-spark
  - https://github.com/LucaCanali/sparkMeasure