
# Real-time Detection Of Anomalies In The Database Infrastructure Using Apache Spark

Prasanth Kothuri, CERN

Daniel Lanza Garcia, CERN

#EUde13

# Outline

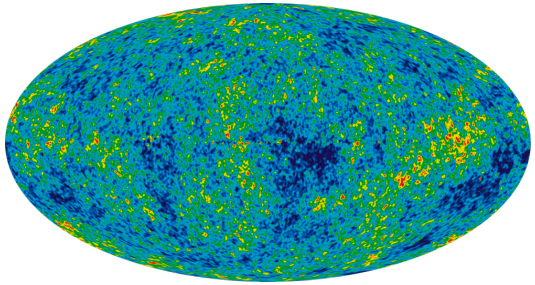
- About CERN
- Apache Spark at CERN
- Data Lake Challenges
- Detecting Anomalous Connections
- Real-time Intelligent Monitoring

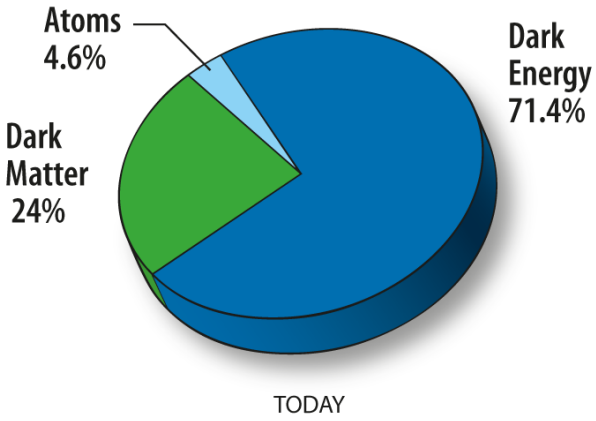
# About CERN

- CERN - European Laboratory for Particle Physics
- Founded in 1954 by 12 Countries for fundamental physics research in the post-war Europe
- Today 21 member states + world-wide collaborations
  - About ~1000 MCHF yearly budget
  - 2'300 CERN personnel
  - 10'000 users from 110 countries

# Fundamental Research

- What is 95% of the Universe made of?
- Why do particles have mass?
- Why is there no antimatter left in the Universe?
- What was the Universe like just after the “Big Bang”





# The Large Hadron Collider (LHC)


**Largest machine** in the world

27km, 6000+ superconducting magnets

**Fastest racetrack** on Earth

Protons circulate 11245 times/s (99.9999991% the speed of light)

**Emptiest** place in the solar system

High vacuum inside the magnets

**Hottest spot** in the galaxy

During Lead ion collisions create temperatures 100 000x hotter than the heart of the sun;

# Apache Spark @ CERN

- Spark is a popular compute engine for Analytics + ETL
  - Deployed on four production Hadoop/YARN clusters
    - Aggregated capacity (2017): ~1500 physical cores, 11 PB
  - Adoption is growing. Key projects involving Spark:
    - Analytics for accelerator controls and logging
    - Monitoring IT Infrastructure, uses Spark + Spark streaming
    - Analytics on aggregated logs
    - Explorations on the use of Spark for high energy physics

# Detection Of Anomalies In Database Infrastructure

# CERN Database Infrastructure

- Over 100 Oracle databases, most of them RAC
  - Running Oracle 11.2.0.4 and **12.1.0.2**
  - 950 TB of data files for production DBs in total (without counting replicas), NAS as storage
  - Oracle In-Memory in production since July 2015
- Examples of critical production DBs:
  - Quench Protection System: 150'000 changes/s
  - LHC logging database: 492 TB, growing 180 TB/year
- But also DB as a Service (single instances)
  - 310 MySQL, 70 PostgreSQL, 9 Oracle, 5 InfluxDB

# Data Lake

- Objective: Build central repository for database audit, performance metrics and logs with the goal of real time analytics and offline analytics

- Use cases
  - Detection of Anomalies
  - Troubleshooting
  - Capacity Planning

# Diverse Data

- Tables
  - Audit
  - Performance Metrics
  - Alert
- Log files
  - Listener

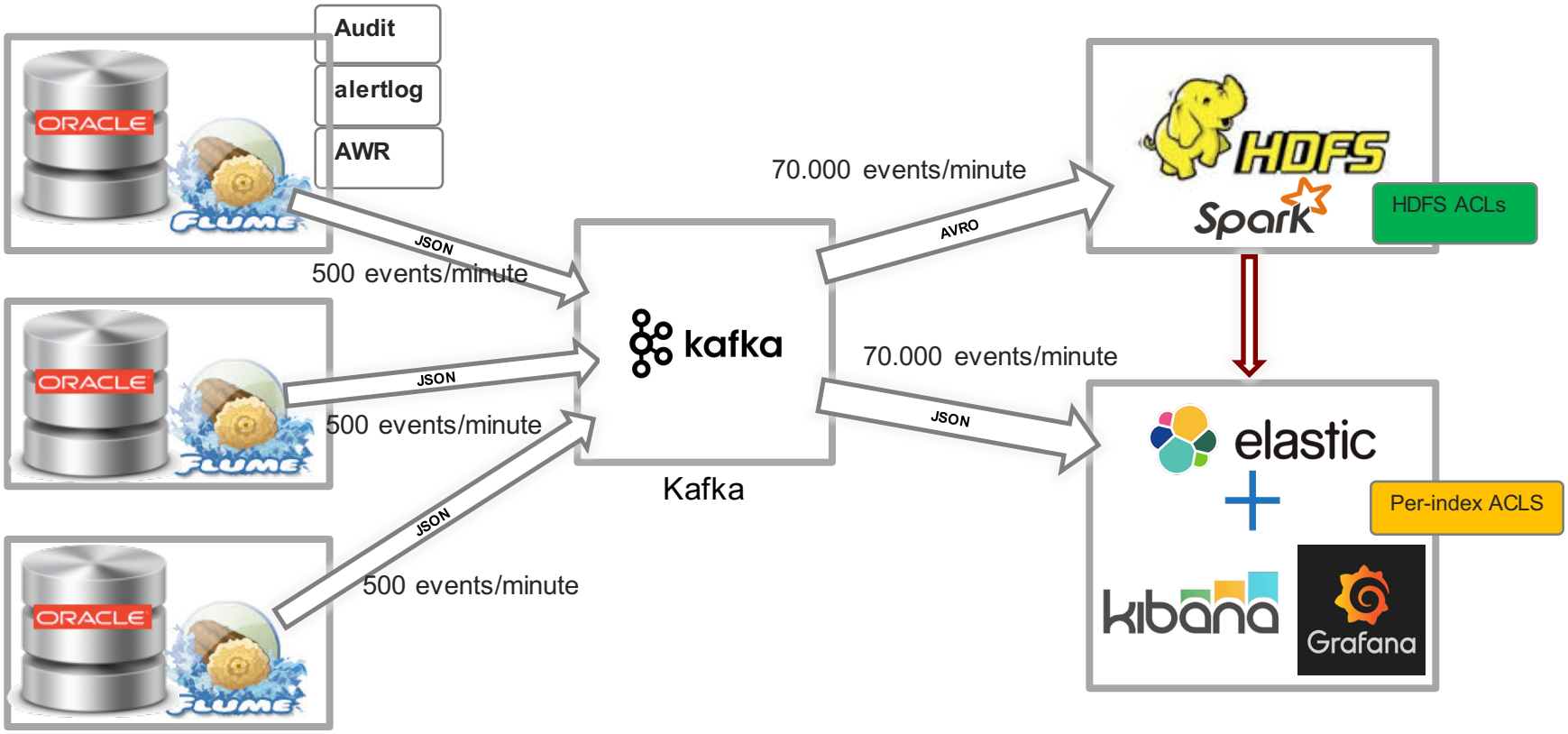
**oracle_sid:** CMSINTR2 **database_type:** oracle **source_type:** alert **hostname:** cmsrac44 **flume_agent_version:** 0.1.4-5.el6 **database_version:** 11.2.0.4.0 **ADDR:** 00007F80FA4511E0 **INDX:** 31,991 **INST_ID:** 2 **ORIGINATING_TIMESTAMP:** November 9th 2016, 09:50:17.000 **NORMALIZED_TIMESTAMP:** - **ORGANIZATION_ID:** oracle **COMPONENT_ID:** rdbms **HOST_ID:** cmsrac44.cern.ch **HOST_ADDRESS:** 10.176.84.61 **MESSAGE_TYPE:** 1 **MESSAGE_LEVEL:** 16 **MESSAGE_ID:** - **MESSAGE_GROUP:** - **CLIENT_ID:** - **MODULE_ID:** - **PROCESS_ID:** 15031 **THREAD_ID:** - **USER_ID:** - **INSTANCE_ID:** - **DETAILED_LOCATION:** -

**oracle_sid:** C2PPSDB2 **database_type:** oracle **producer:** oracle **source_type:** metric **hostname:** itrac1329.cern.ch **flume_agent_version:** 0.1.6-7.el6 **type:** dbconnection **database_version:** 11.2.0.4.0 **DB_NAME:** C2PPSDB **DB_UNIQUE_NAME:** C2PPSDB_RAC13 **INSTANCE_NAME:** C2PPSDB2 **BEGIN_TIME:** June 29th 2017, 11:51:11.000 **END_TIME:** June 29th 2017, 11:52:11.000 **METRIC_NAME:** Enqueue Deadlocks Per Txn **VALUE:** 0B **_id:** AVzzRAX-Nj0vFzICP54W **_type:** log **_index:** itdb_db-metric-2017-06-29 **_score:** -

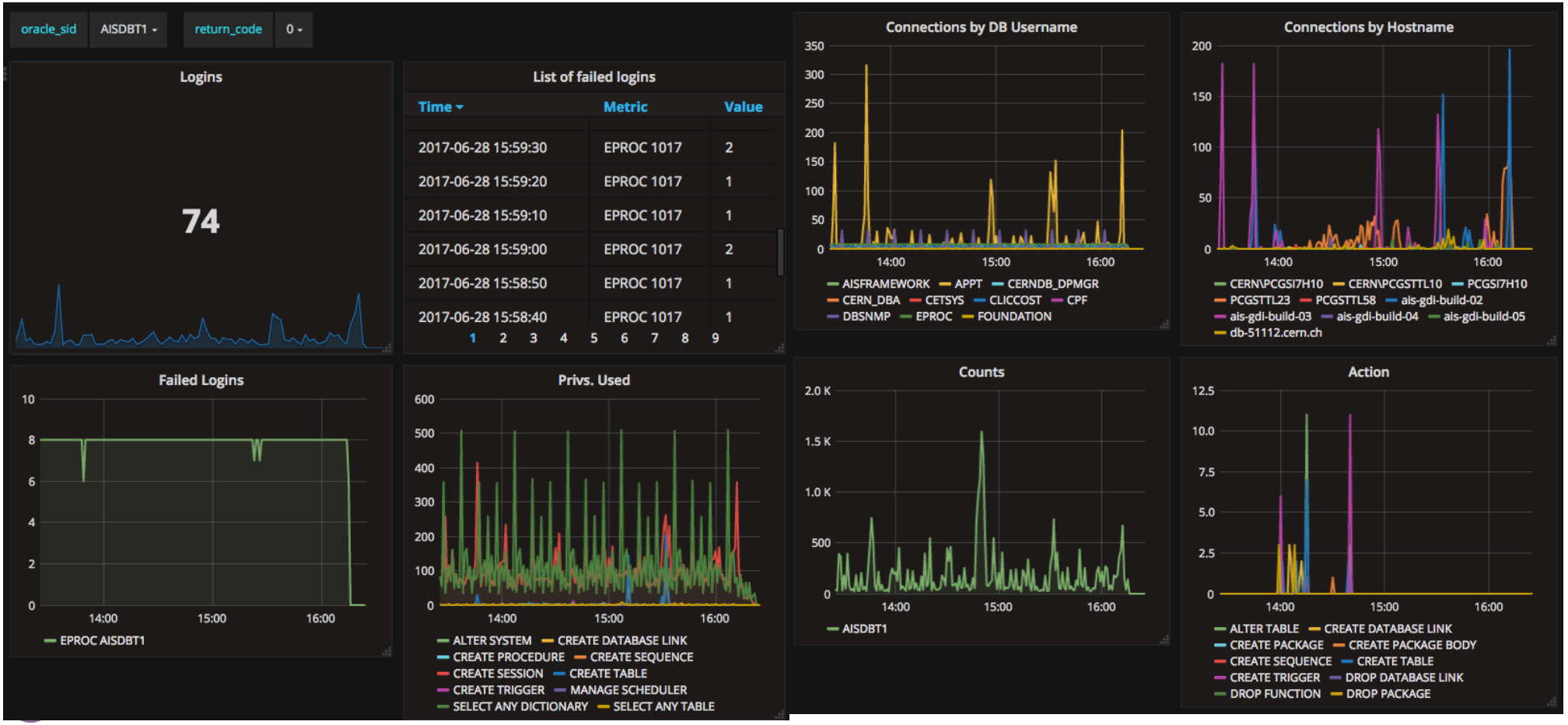
```
29-JUL-2016 15:17:34 * (CONNECT_DATA=(SID=DESFOUND)(CID=(PROGRAM=oracle)(HOST=itrac50035.
29-JUL-2016 15:17:38 * service_update * WCERND * 0
Fri Jul 29 15:18:45 2016
29-JUL-2016 15:18:45 * (CONNECT_DATA=(SID=PAYT)(CID=(PROGRAM=RTSDGN@db-50016)(HOST=db-500
```

| | | |
|---|---|---|
| t | client_host | db-51220.cern.ch |
| t | client_ip | 137.138.161.92 |
| # | client_port | 33,450 |
| t | client_program | perl |
| t | client_protocol | tcp |
| t | client_user | sysctl |

# Data pipeline Architecture



# Aggregations and Dashboards



# Real-time analysis

- Attempt to detect anomalous connections using Machine Learning techniques
- First approach to use kNN classifier to classify connection as *normal* or *anomalous*
- Anomaly detection suffer from high false positive rate, the challenge is to choose features that best characterize user connections pattern

# Feature Selection

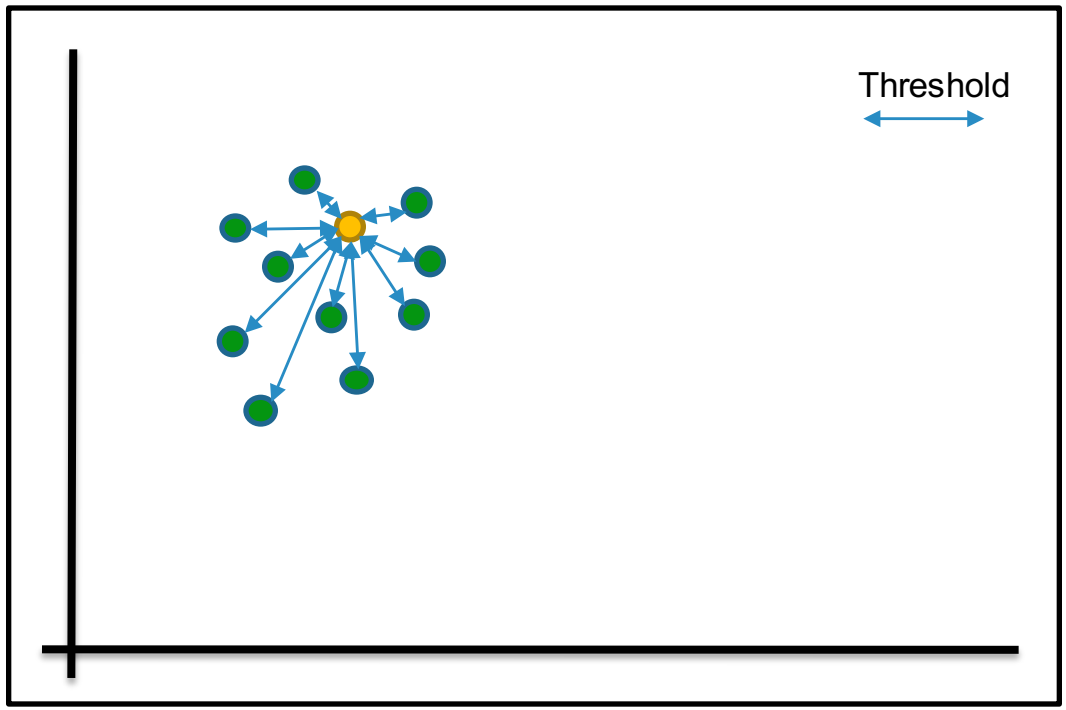
- Data Preparation

- Feature Extraction
  - Client_host
  - Client_user
  - Client_program
  - Service_name
  - Hour_of_the_day
  - Day_of_the_week


```
{
  "oracle_sid" : "XXXX_YYYY",
  "listener_name" : "listener_scan1",
  "database_type" : "oracle",
  "producer" : "oracle",
  "source_type" : "listener",
  "hostname" : "XXXXX.cern.ch",
  "flume_agent_version" : "0.1.6-7.el6",
  "type" : "dbconnection",
  "event_timestamp" : "2017-09-27T04:45:27+0200",
  "CONNECT_DATA_SERVER" : "DEDICATED",
  "CONNECT_DATA_SERVICE_NAME" : "XXXr.cern.ch",
  "client_program" : "python",
  "client_host" : "pxxxxj2.cern.ch",
  "client_user" : "merge",
  "client_protocol" : "tcp",
  "client_ip" : "137.100.100.167",
  "client_port" : 38432,
  "type" : "establish",
  "service_name" : "XXXX.cern.ch",
  "return_code" : 0
}
```

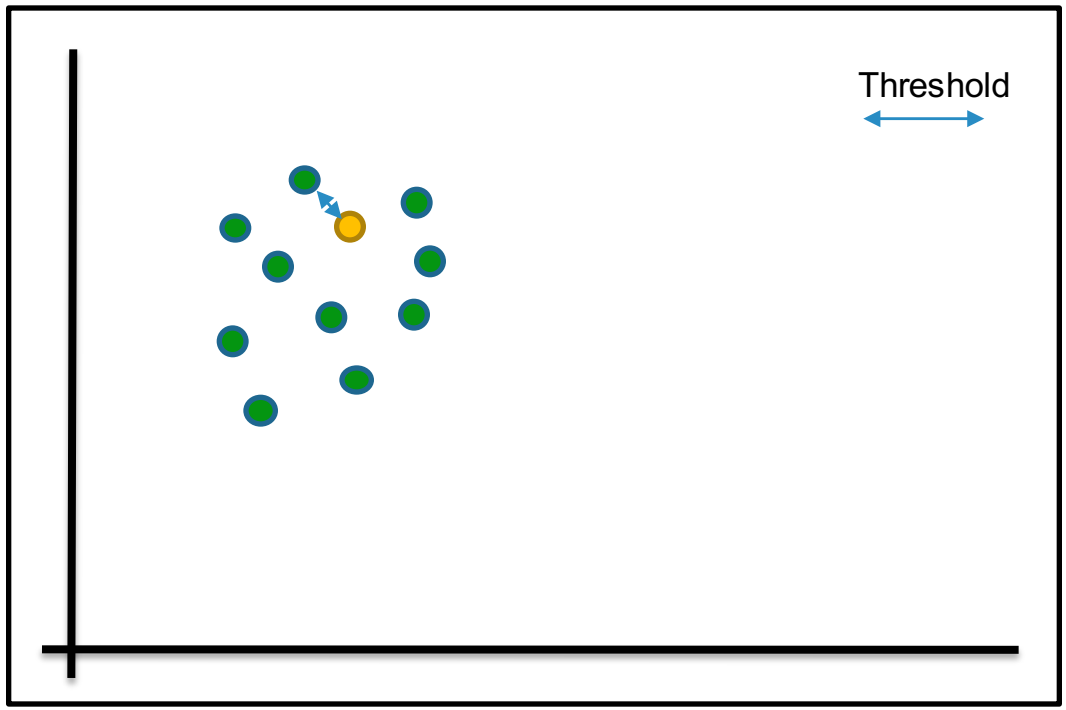
# Detecting anomalous connections job

- Machine learning: kNN



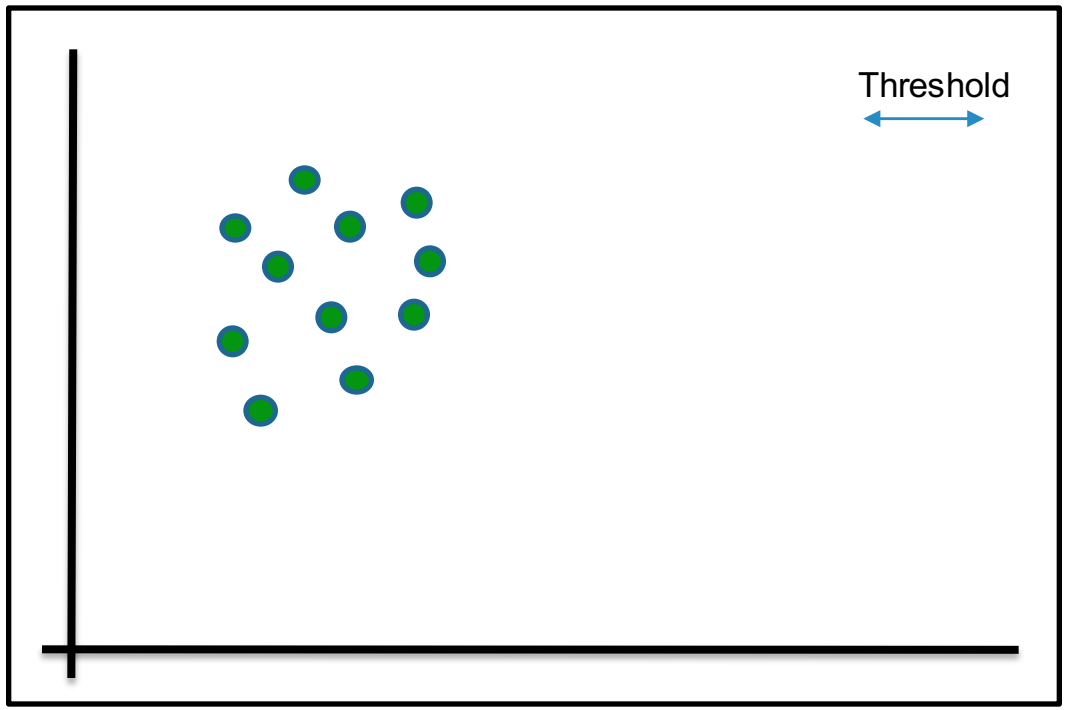
# Detecting anomalous connections job

- Machine learning: kNN



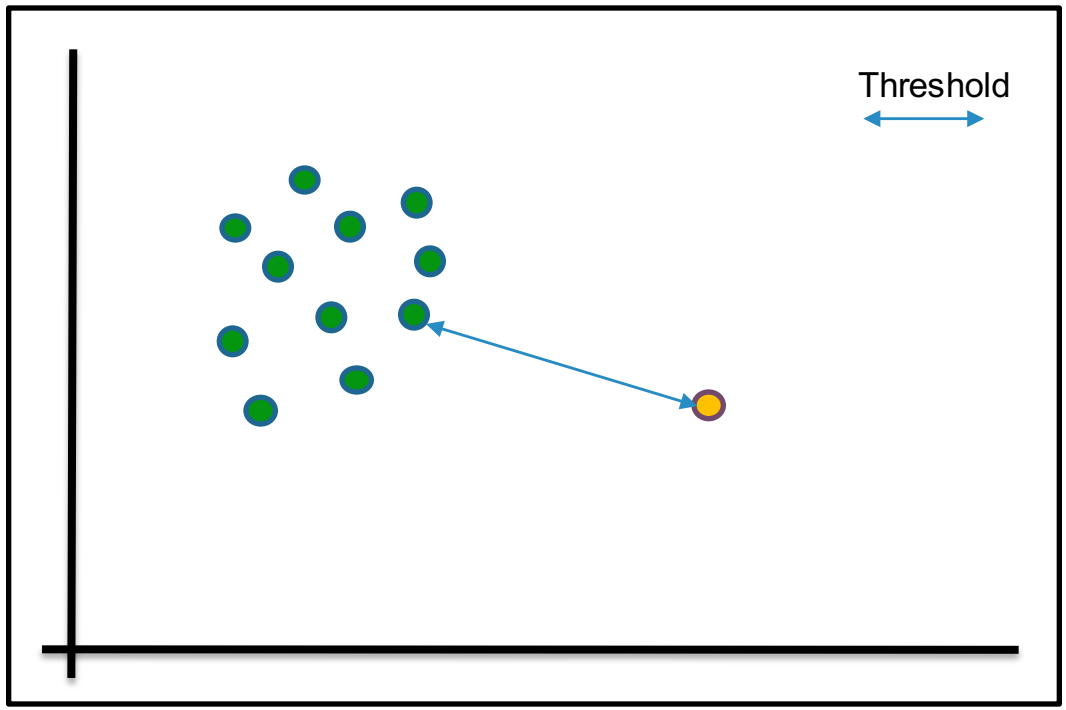
# Detecting anomalous connections job

- Machine learning: kNN



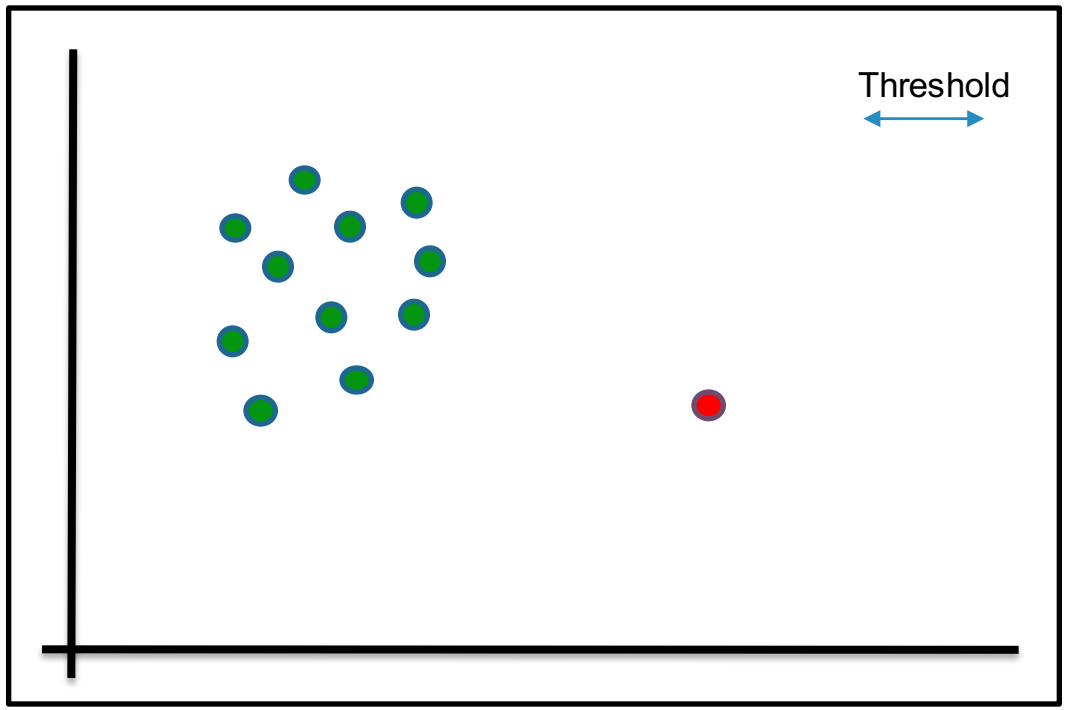
# Detecting anomalous connections job

- Machine learning: kNN



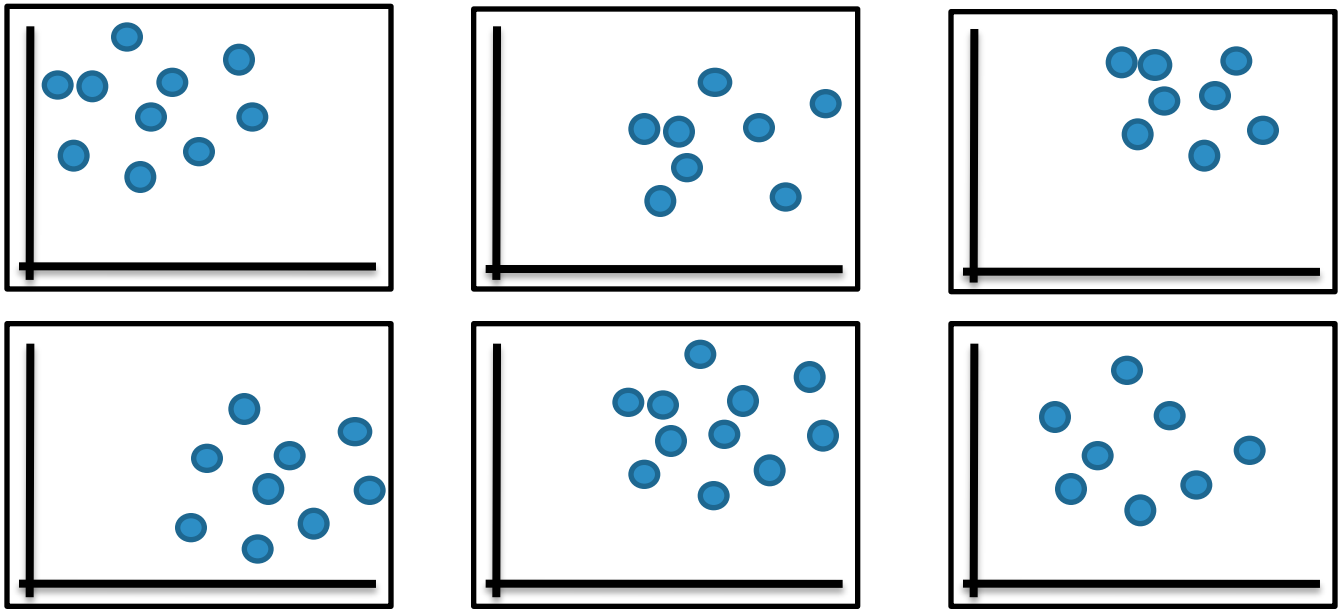
# Detecting anomalous connections job

- Machine learning: kNN



# Detecting anomalous connections job


- Machine learning: kNN
  - Making it scalable, each user should behave similarly



# Detecting anomalous connections job

- Distance Function:

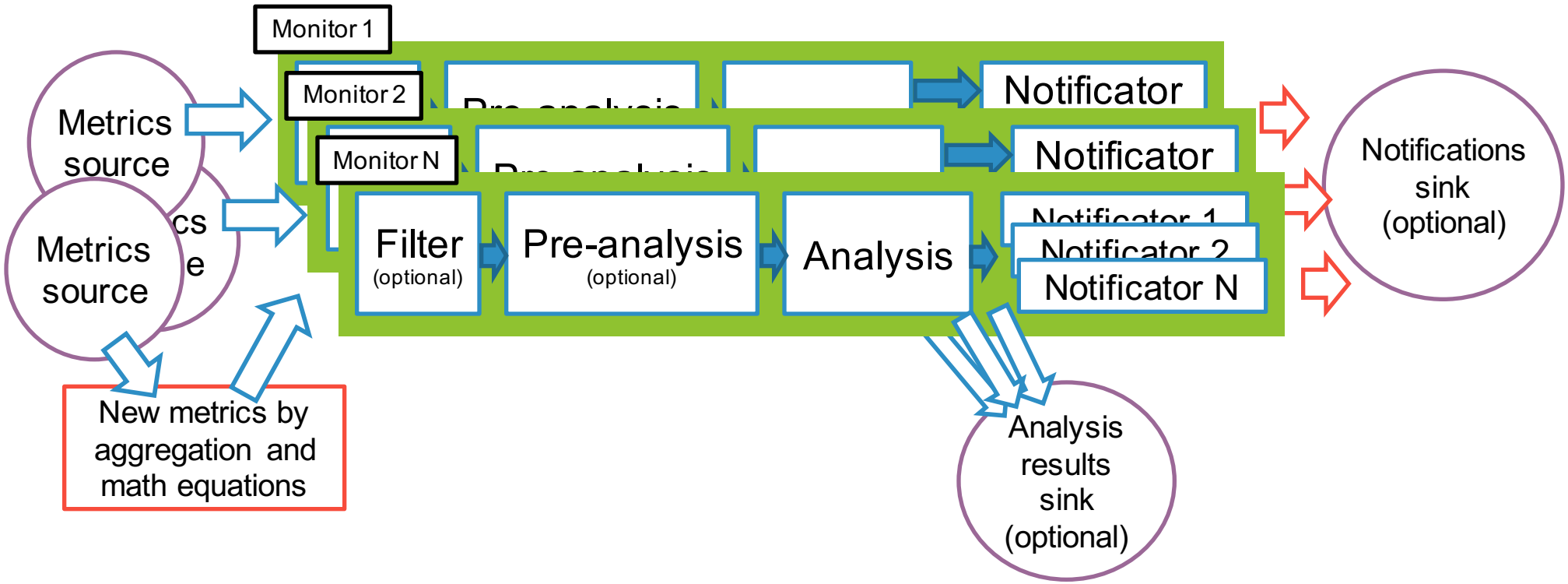
```
for (Connection trustedConnection : connections) {
        float distance = connection.distance(dataMineProperties, trustedConnection);

        if(distance < minimun_distance)
                minimun_distance = distance;
}
```

- Spark Streaming Job processing – 10 million / day
- 6 executors, each with 4 cores and 8GB of memory

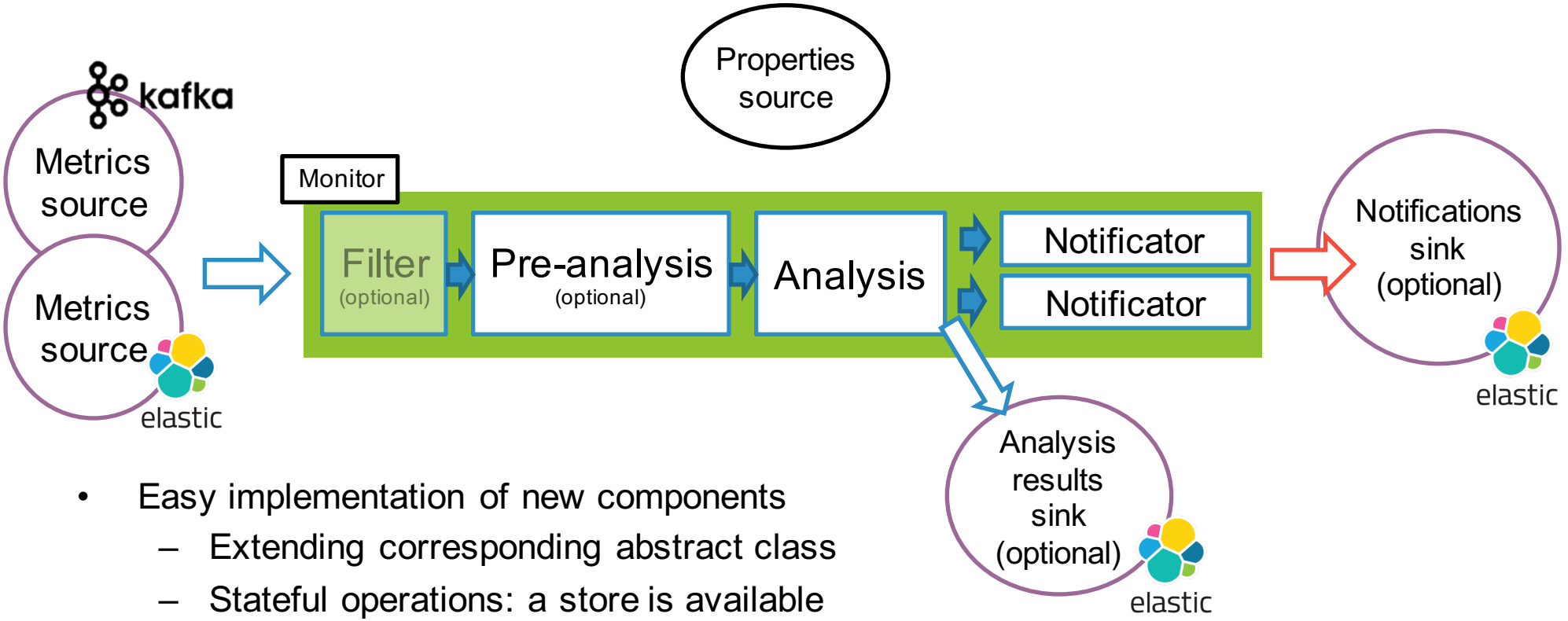
# Metrics monitor job

- A general purpose metrics monitor

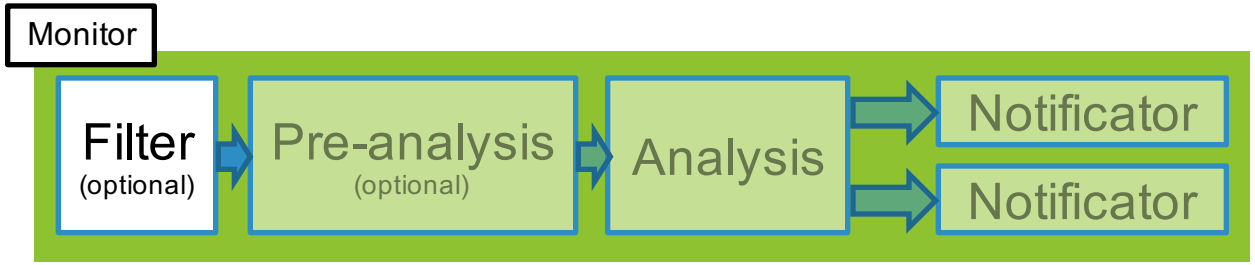
# Metrics monitor job: data flow



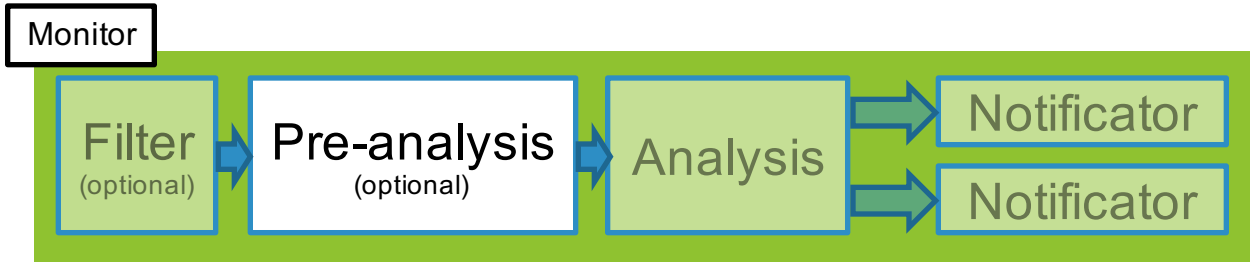
# Metrics monitor job: modularity




- Easy implementation of new components
  - Extending corresponding abstract class
  - Stateful operations: a store is available

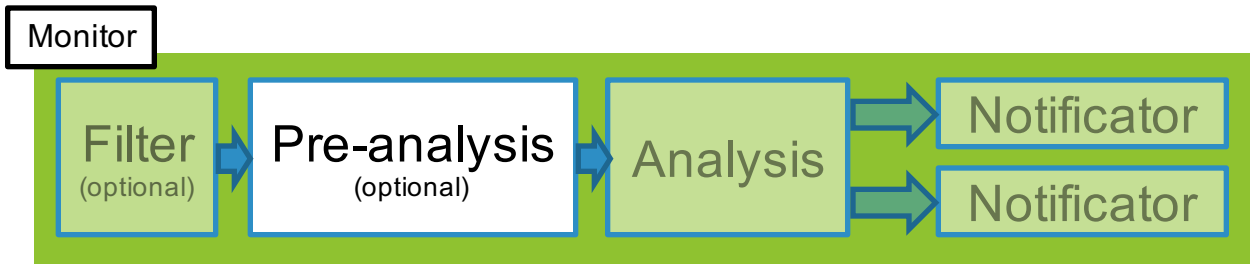



- Which metrics should process this monitor?
  - CPU Usage of all databases, Cache Hit % of MySQL databases, all machines in PROD, …
- Filter metrics by attributes
- Accepts exact value or regular expressions
- If not applied, all metrics are processed

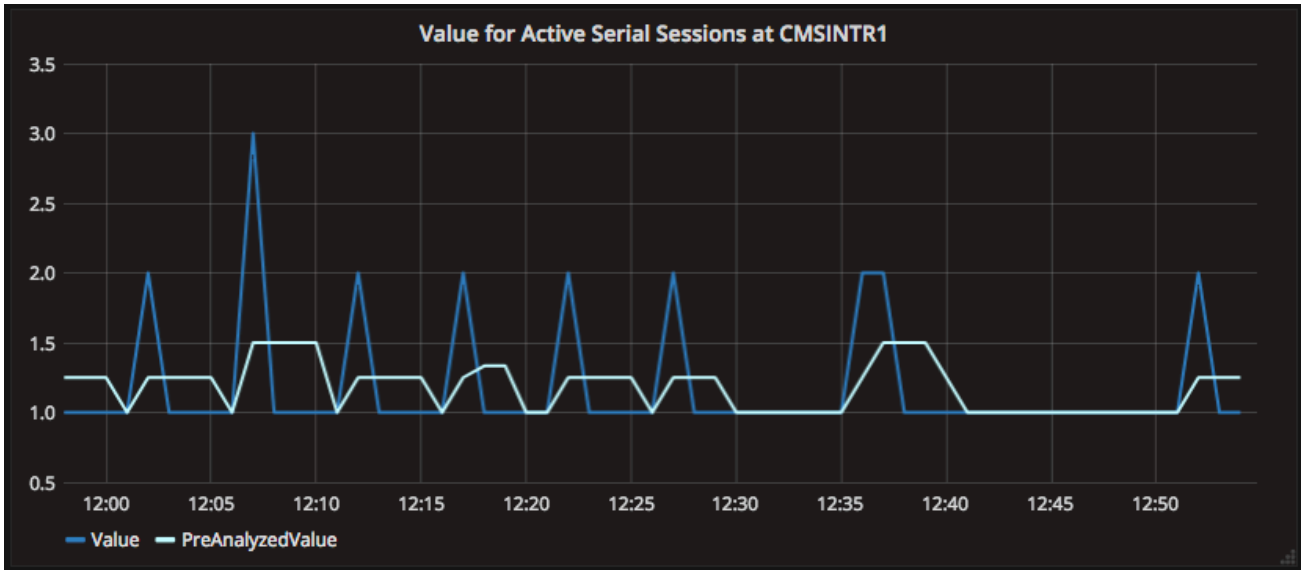


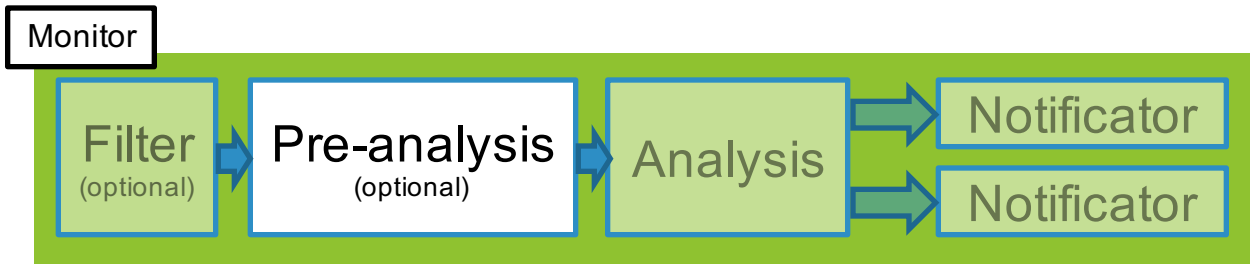

- Massage metric value
  - Average, difference, filter, …

- Internal store for stateful pre-analysis
- If applied, result will be the analyzed value

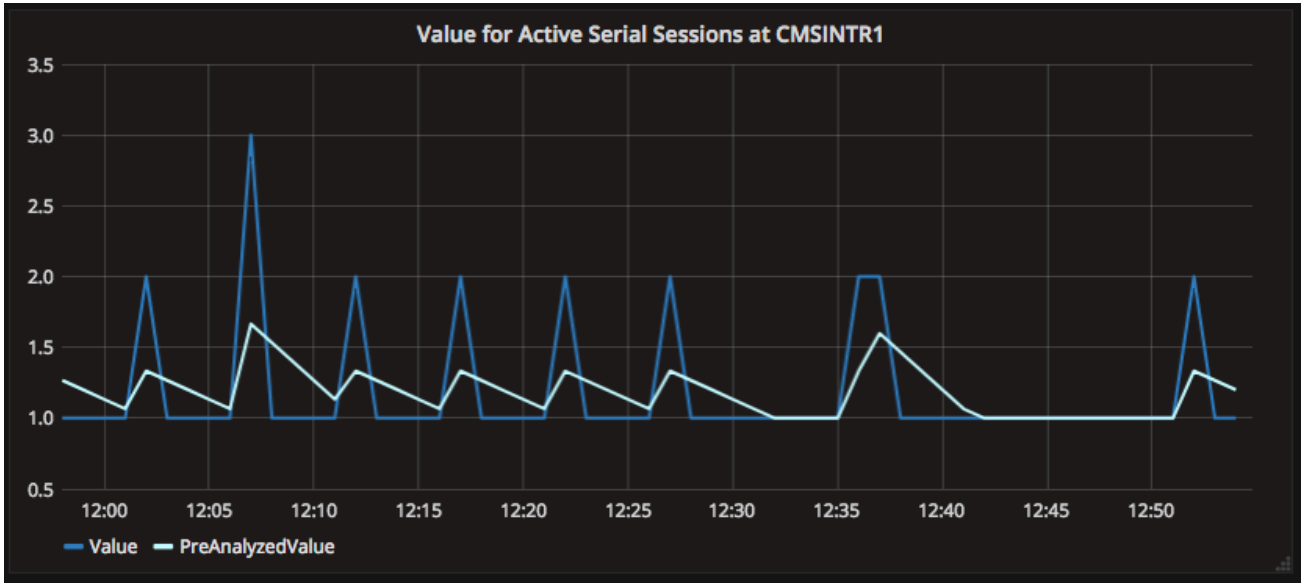


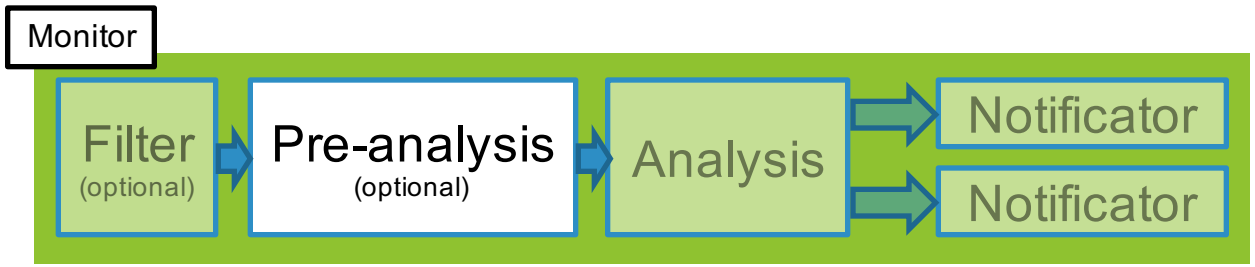

- Type: average value of a period



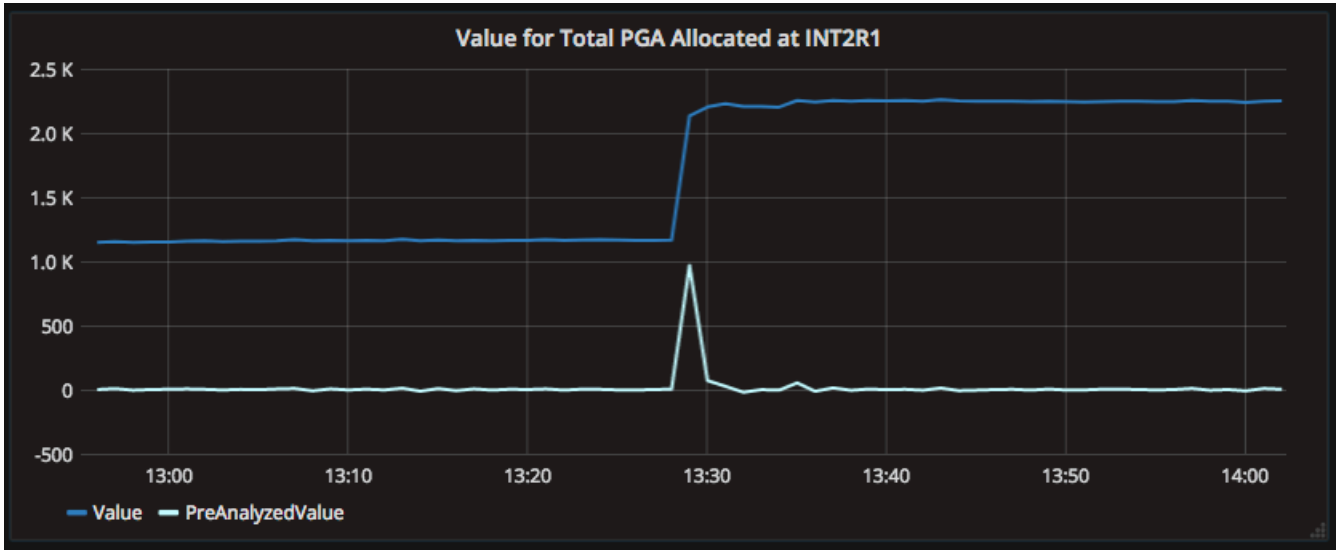


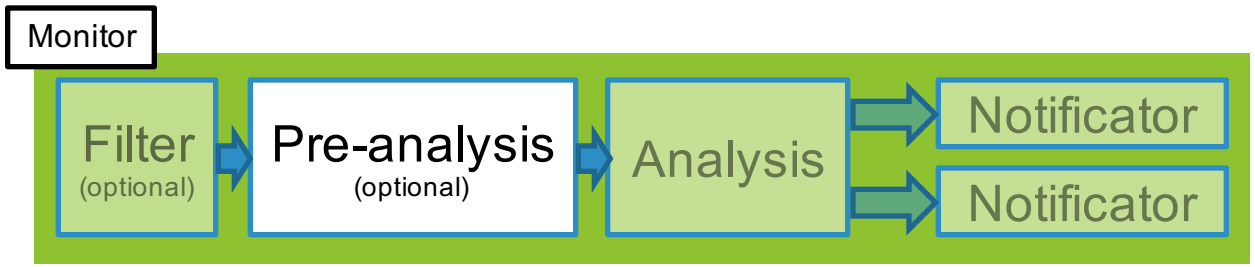

- Type: weighted average value of a period



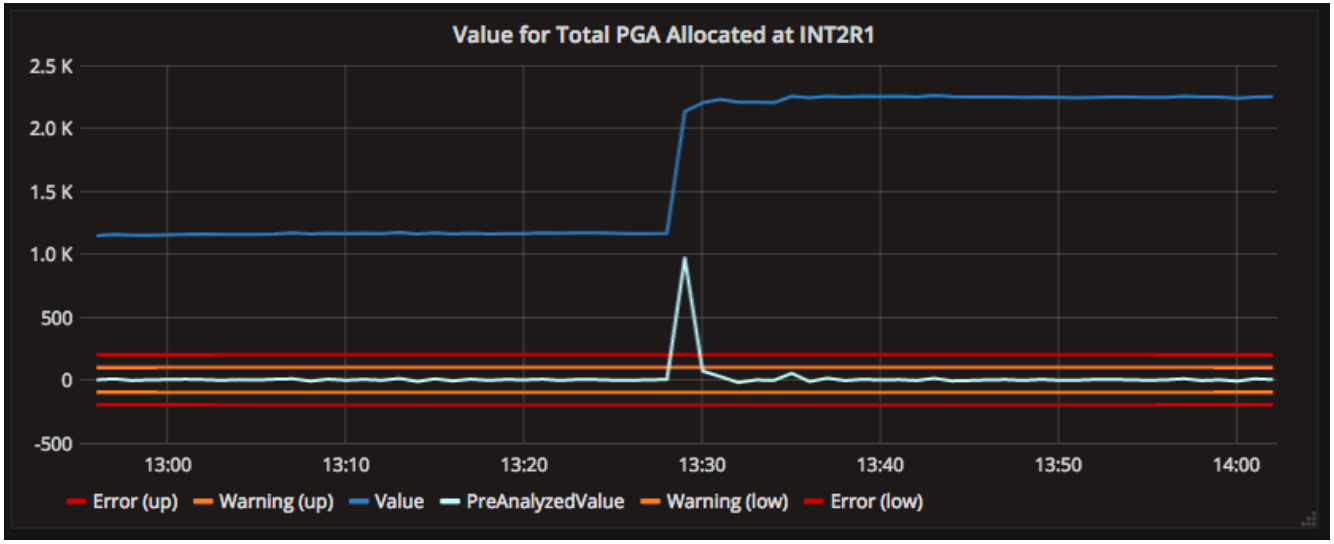


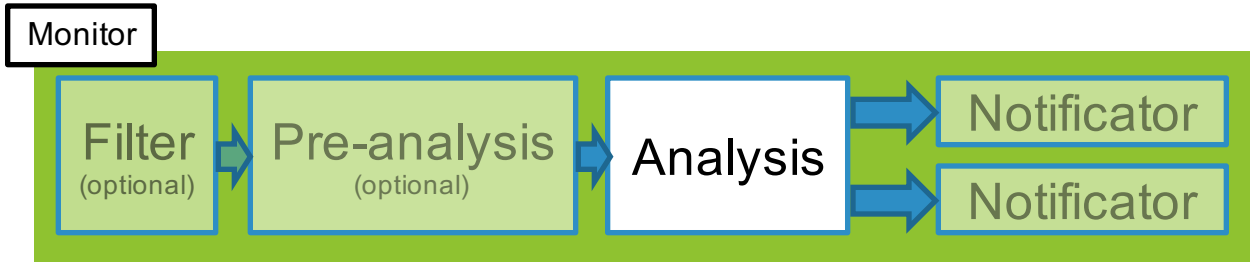

- Type: difference with previous value






- Type: difference with previous value

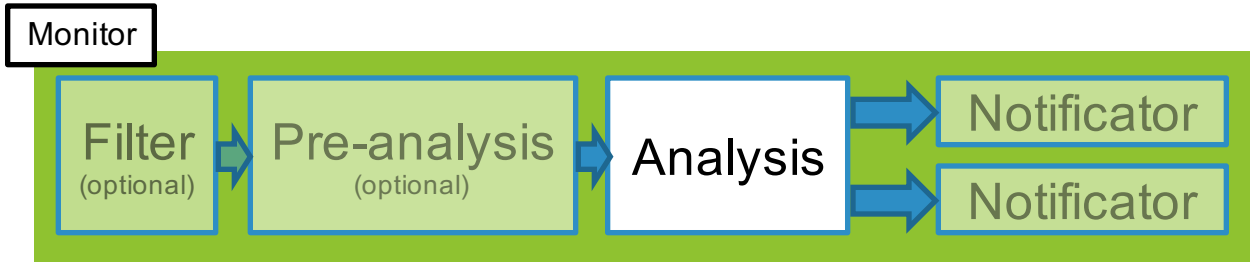




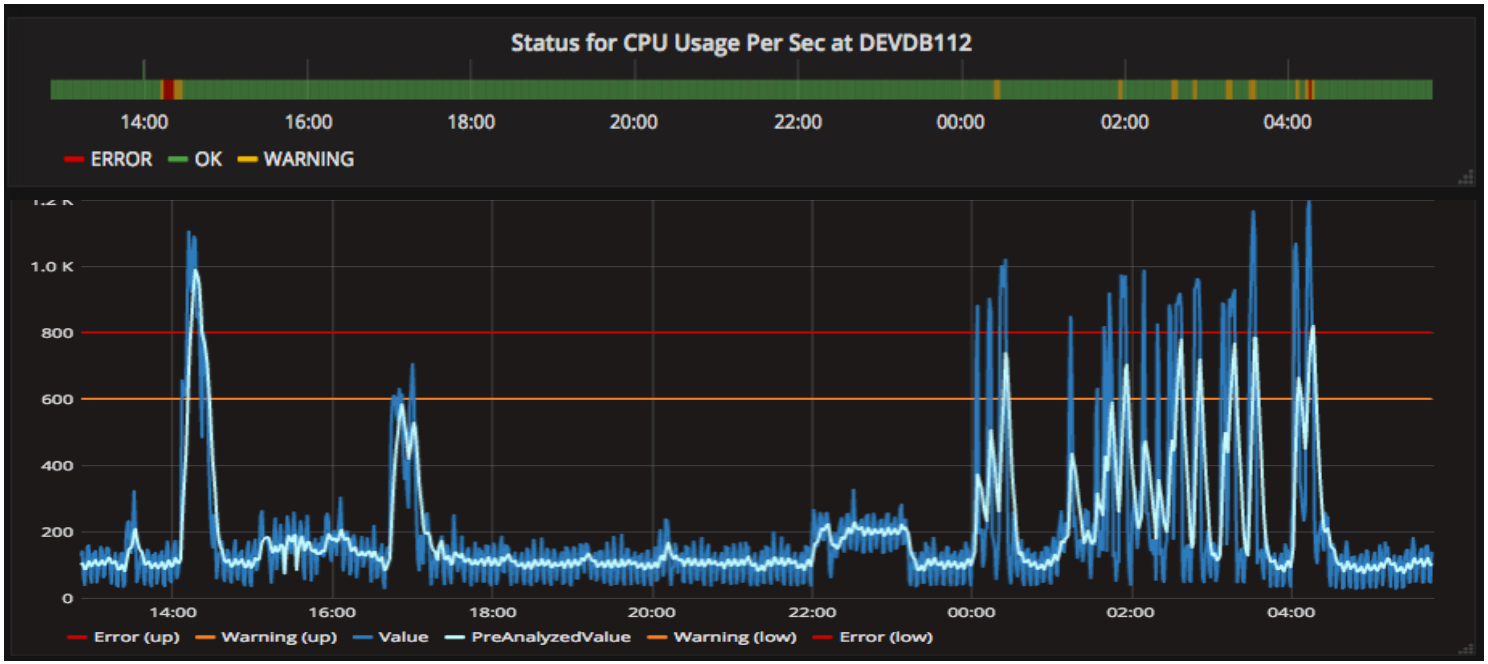

- Is the metric behaving as it should?

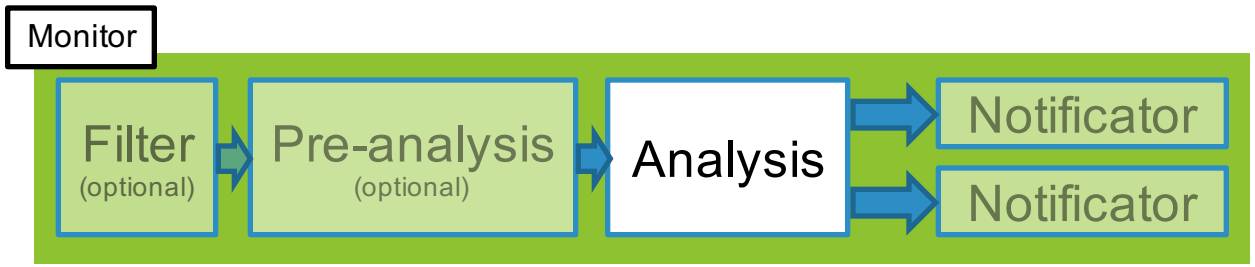
- Determine the status of each incoming metric
  - OK, WARNING, ERROR, EXCEPTION

- Results can be sent to external storage
- Internal store for stateful analysis

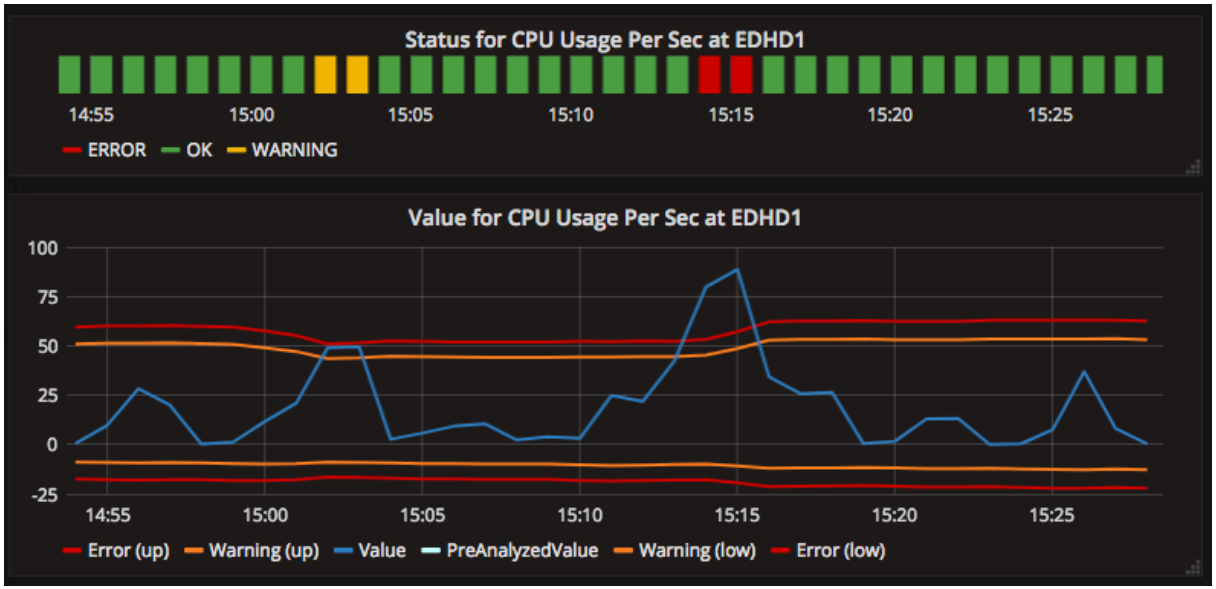


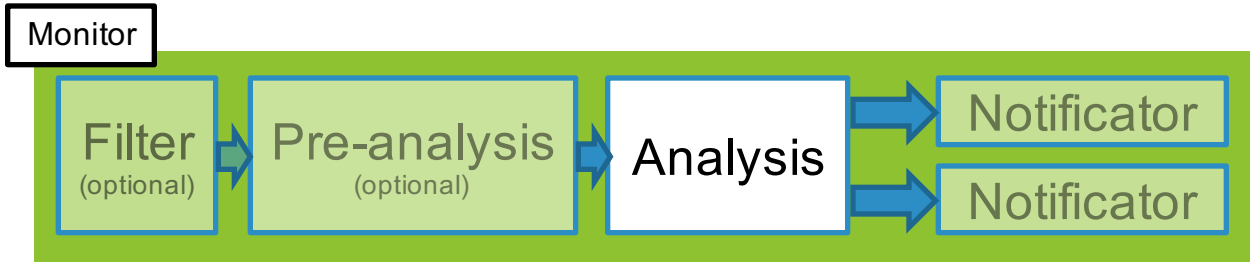

- Type: thresholds fixed manually



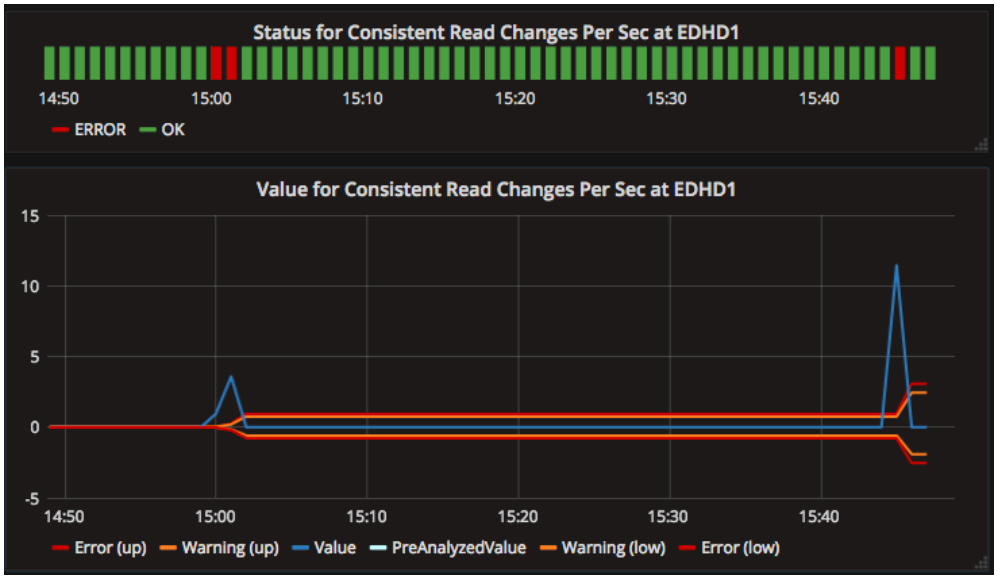


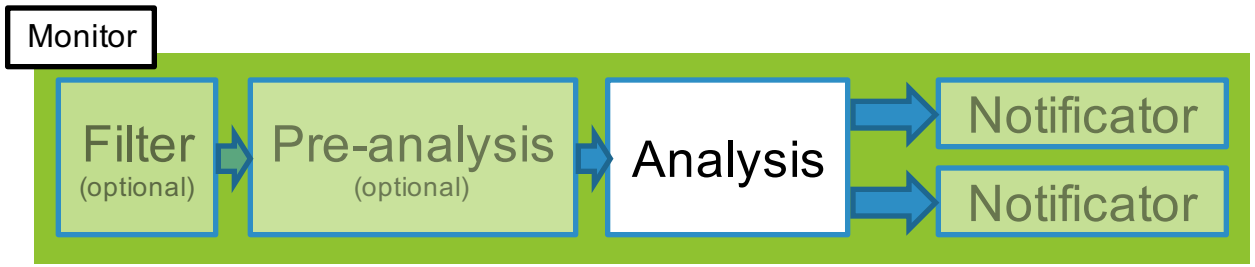

- Type: learning from recent behaviour (average and variance)
  - Errors when metric does not behave as it used to



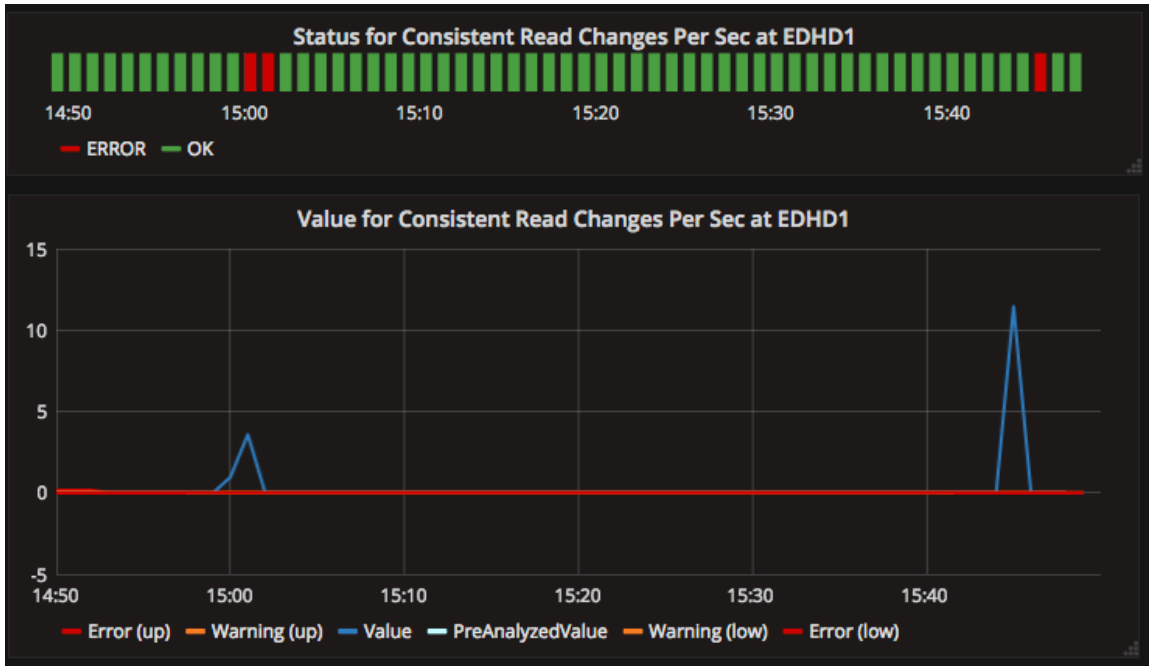


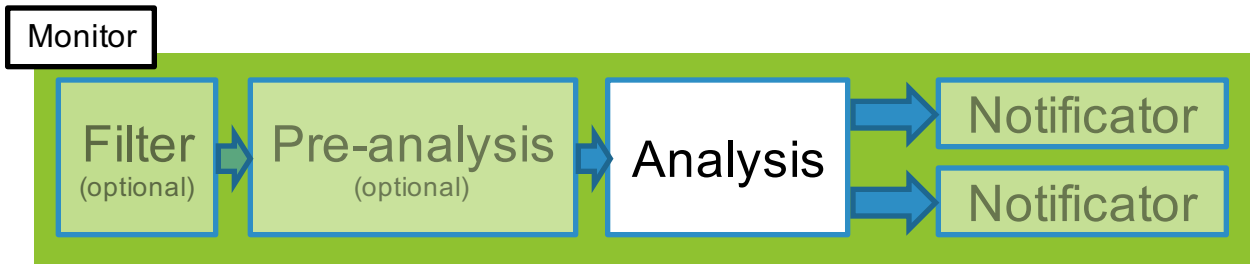

- Type: learning from recent behaviour (average and variance)
  - Easily affected by outliers



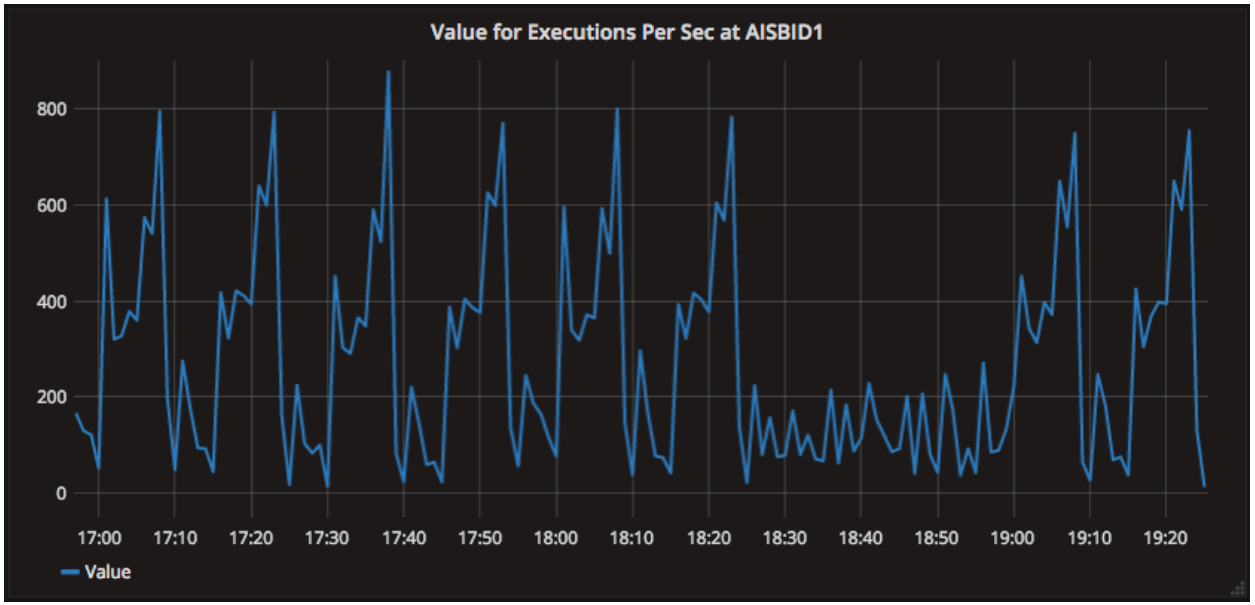


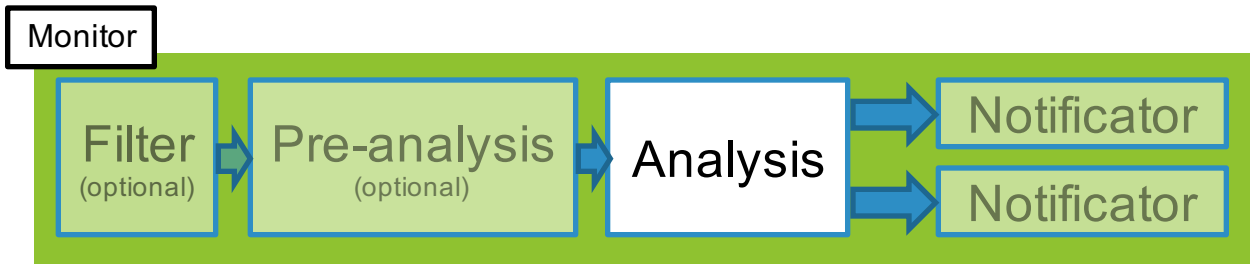

- Type: learning from recent behaviour (percentiles)



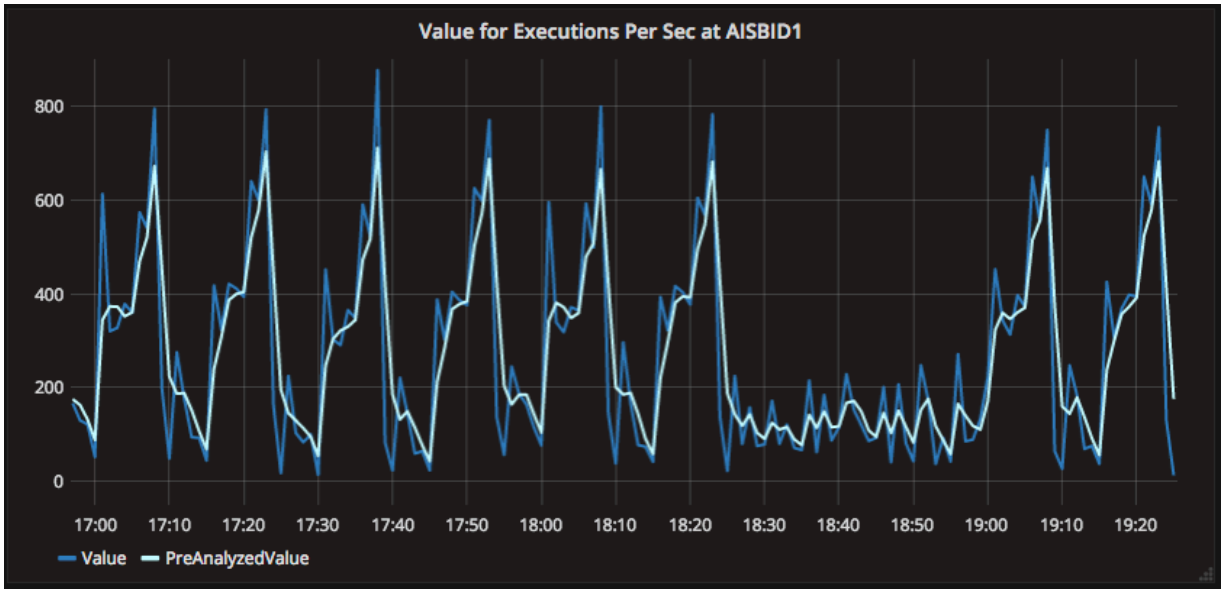


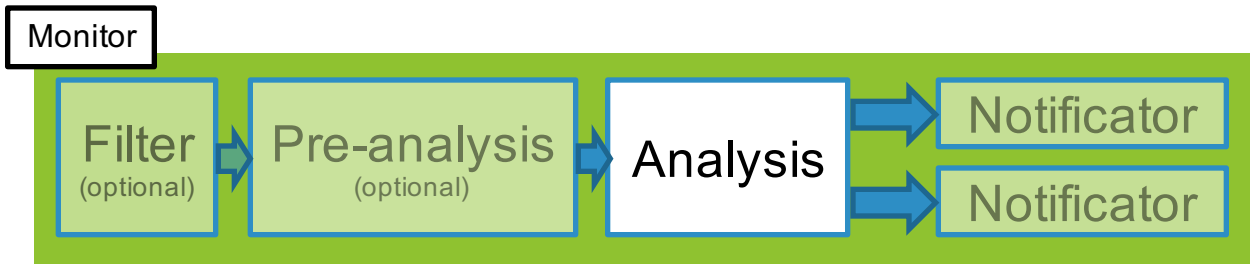

- Type: learning a season (hour, day, week)



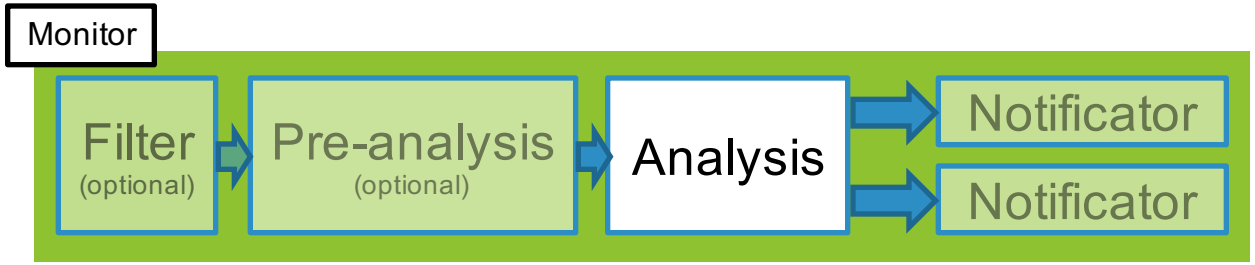



- Type: learning a season (hour, day, week)

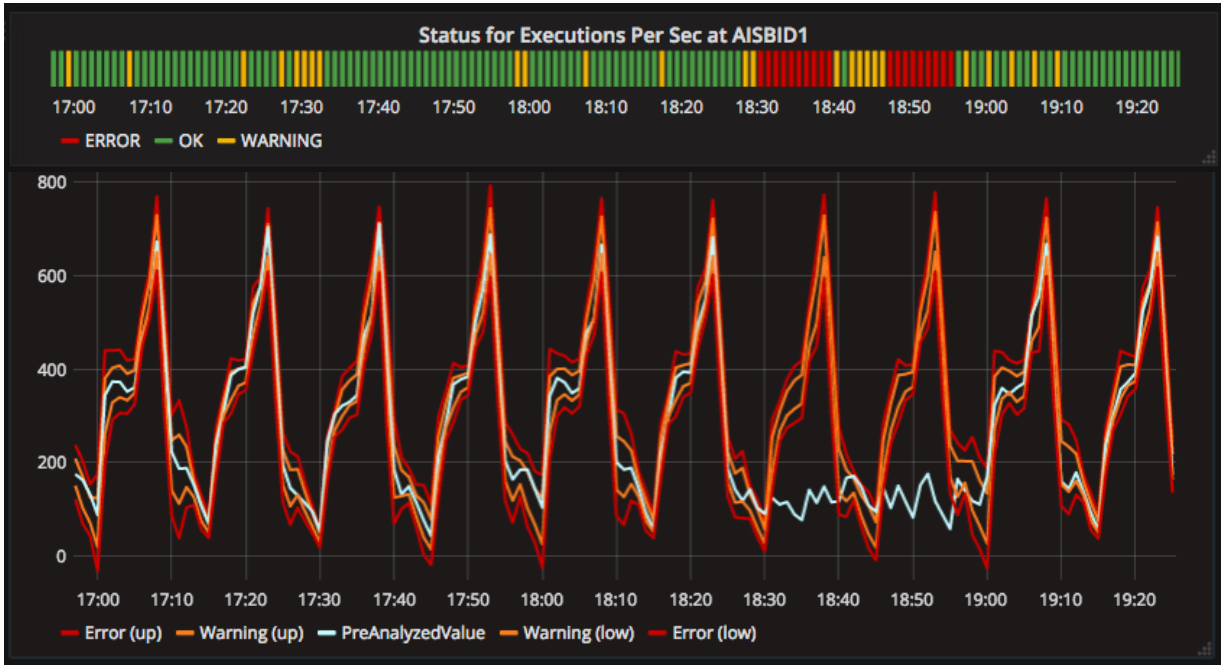


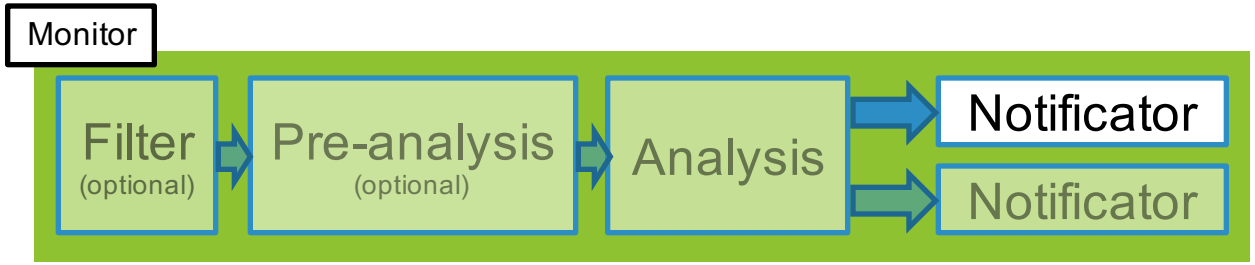



- Type: learning a season (hour, day, week)



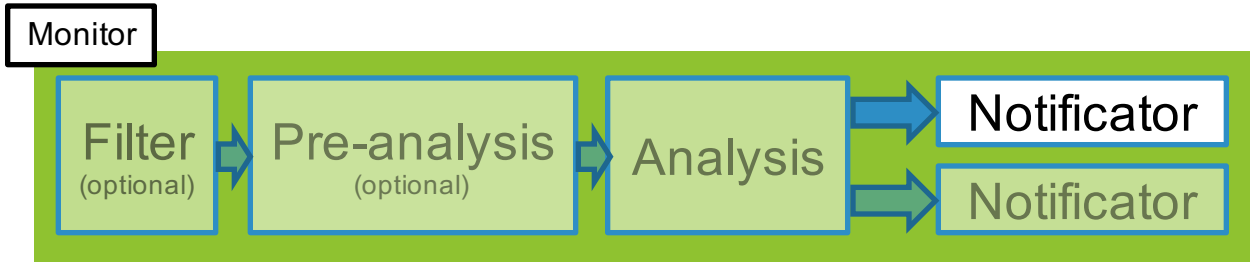

- Type: learning a season (hour, day, week)



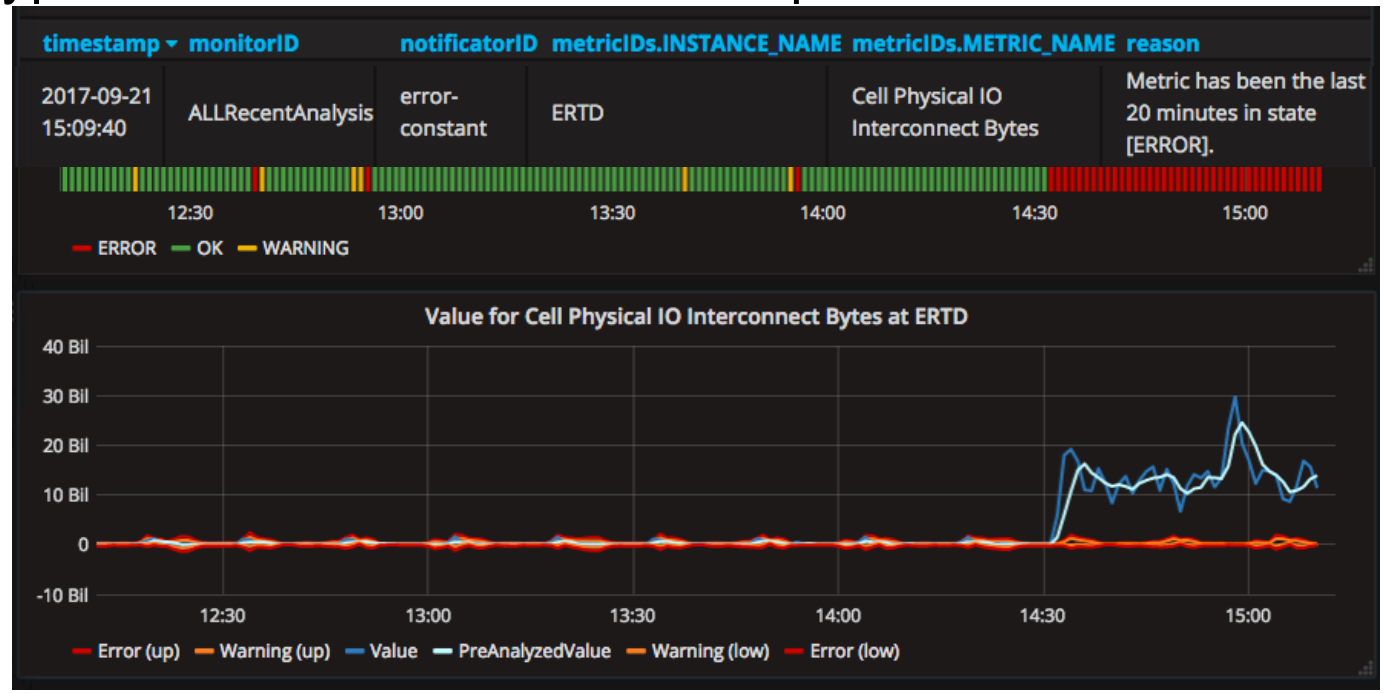


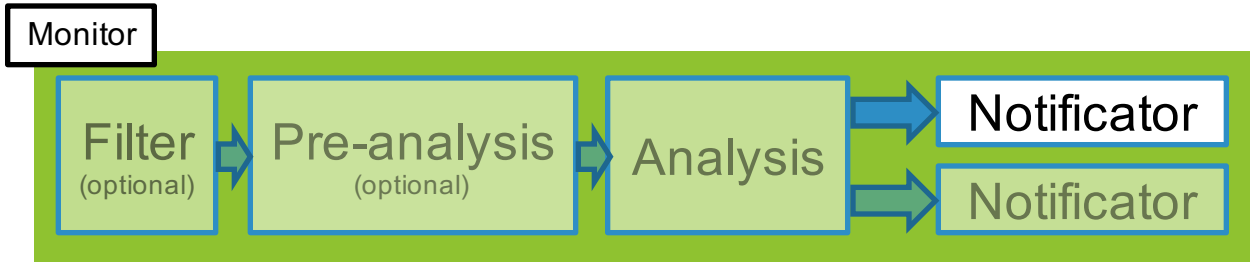

- Should we inform anyone?
  - Metric has been in ERROR for 20 minutes…

- Determines when a notification is raised
- Based on metric statuses
- Notifications are consumed by Notifications sink

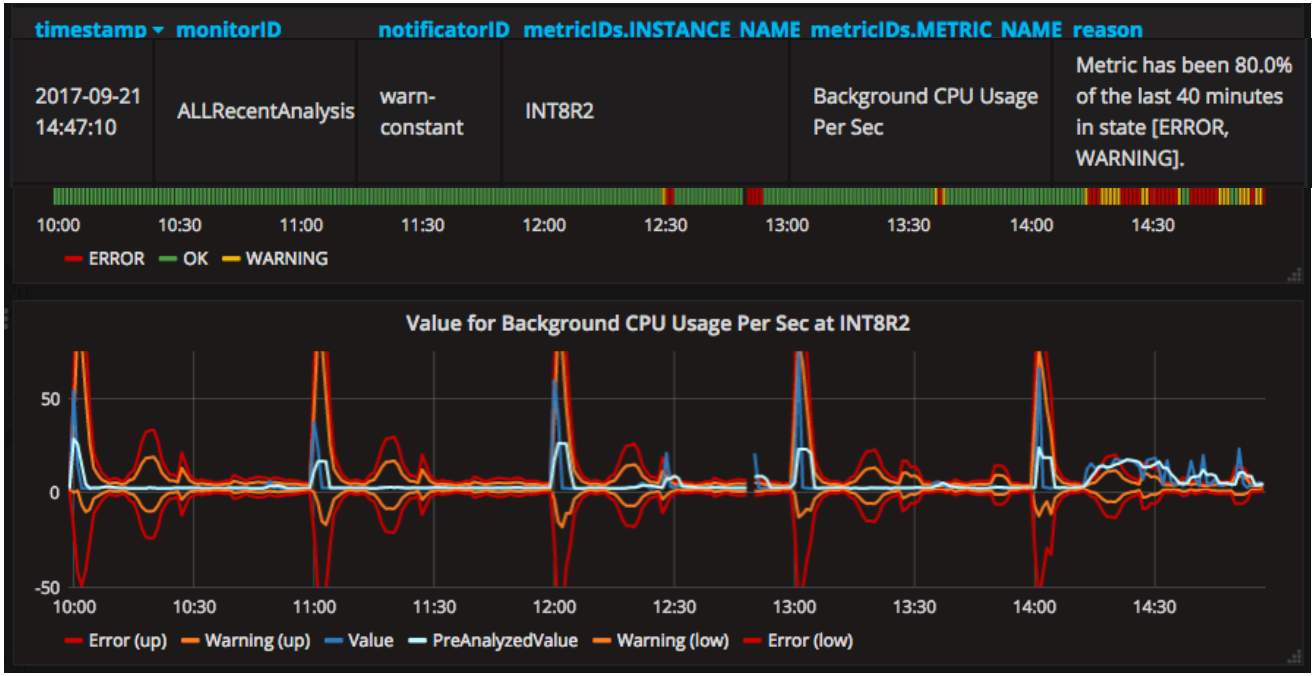



- Type: constant status for certain period



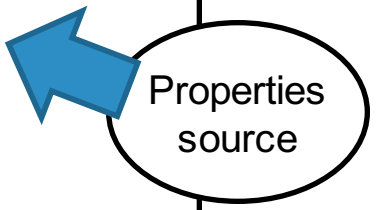



- Type: percentage of the period in certain statuses

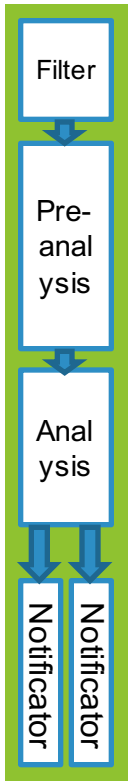


# Metrics monitor job: configuration

```
checkpoint.dir = <path_to_store_stateful_data> (default: /tmp/)
spark.batch.time = <seconds> (default: 30)

metrics.source.kafka-prod.type = kafka
metrics.source.kafka-prod.consumer.bootstrap.servers = server:port,server:port
metrics.source.kafka-prod.topics = db-logging-platform
metrics.source.kafka-prod.parser.attributes = INSTANCE_NAME METRIC_NAME
metrics.source.kafka-prod.parser.value.attribute = VALUE
metrics.source.kafka-prod.parser.timestamp.attribute = END_TIME

results.sink.type = elastic
results.sink.index = itdb_db-metric-results/log
notifications.sink.type = elastic
notifications.sink.index = itdb_db-metric-notifications/log

# Monitors
monitor.<monitor-id-1>.<confs>...
monitor.<monitor-id-2>.<confs>...
monitor.<monitor-id-n>.<confs>...
```



# Metrics monitor job: monitor configuration




- Configuration can be updated while running
  - Add/remove monitor or change any parameter

```
# Monitor CPU of all instances in production
monitor.CPUUsage.filter.attribute.INSTANCE_NAME  = regex:prod-.*
monitor.CPUUsage.filter.attribute.METRIC_NAME  = CPU Usage Per Sec
monitor.CPUUsage.missing.max-period  = 3m
monitor.CPUUsage.pre-analysis.type  = weighted-average
monitor.CPUUsage.pre-analysis.period  = 5m
monitor.CPUUsage.analysis.type  = seasonal
monitor.CPUUsage.analysis.season  = hour
monitor.CPUUsage.analysis.learning.ratio  = 0.2
monitor.CPUUsage.notificator.error-perc.type  = percentage
monitor.CPUUsage.notificator.error-perc.statuses  = ERROR
monitor.CPUUsage.notificator.error-perc.period  = 10m
monitor.CPUUsage.notificator.warn-perc.type  = percentage
monitor.CPUUsage.notificator.warn-perc.statuses  = WARNING
monitor.CPUUsage.notificator.warn-perc.period  = 20m
```

# Metrics monitor job

- We are monitoring:
  - Around 30K metrics per minute (day season analysis)
  - Processing time = 30 seconds/minute
  - On 3 executors (YARN)

- Repo, user's and developer's manual, contributions, issues:
  - https://github.com/cerndb/metrics-monitor

# Tips and Suggestions

# Java 8: import java.time.*;

```
private List<DatedValue> values;

private Duration period = Duration.ofMinutes(30);

public void purge(Instant time) {
    Instant oldest_time = time.minus(period);

    values.removeIf(value -> value.getInstant().isBefore(oldest_time));
}
```

# Java 8:
## Extend RDDs and Streams + Optional + lambdas + + java.streaming.* + reference methods = ❤

```
Stream<Metric> metrics = metricSources.stream()
                                      .map(source -> source.createStream(ssc))
                                      .reduce((str, stro) -> str.union(stro)).get();

metrics = metrics.union(metrics.mapS(definedMetrics::generate));

Stream<AnalysisResult> results = metrics.mapS(monitors::analyze);

analysisResultsSink.ifPresent(results::sink);

Stream<Notification> notifications = results.mapS(monitors::notify);

notificationsSink.ifPresent(notifications::sink);
```

# Not using Spark check-pointing feature

- **Pro**: develop/update the job without loosing stateful data
- **Cons**: implement save/load

```
JavaRDD<Tuple2<K, S>> initialStates = RDD.<Tuple2<K, S>>load(getSparkContext(), id);

StateSpec<K, V, S, R> statusSpec = StateSpec
                                    .function(updateStatusFunction)
                                    .initialState(initialStates.rdd())
                                    .timeout(getDataExpirationPeriod());

StatusStream<K, V, S, R> statusStream = StatusStream.from(asJavaPairDStream().mapWithState(statusSpec));

statusStream.getStatuses().save(id);
```

- Note: for stateful operations checkpoint needs to be enable
  - But when restarting, data is removed

# Conclusion

- Building the data lake opens up for several kinds of analysis
- Implementation of anomalous connections helps us to identify potential intrusions to the database infrastructure
- Intelligent monitoring tool learns and adapts the threshold
- Intelligent monitoring tool is open source and available for everyone
- Apache Spark Streaming helps us to perform machine learning at scale