



# Apache stack based Infrastructure monitoring and analytics in Big Telco

JongHyok Lee  
SK Telecom



# About Me

- Architect at SK Telecom SW R&D Center
- Leading development of T-CORE, SK Telecom's datacenter and mobile network monitoring and analytics platform
  
- Previously worked as a Senior Architect at IBM Korea Lab
- Mainly focusing on data processing and management related solution implementation



# Agenda

- Telco infra monitoring & analytics
- Real-time processing of monitoring data
- Storing & querying time-series data
- Analytics and real-time inferencing
- Challenges and constraints
- Lessons Learned

---

# Telco infra monitoring & analytics

# SK Telecom and Big data

- SK Telecom

- South Korea's largest telecommunications company
- More than 30 million subscribers
- Business Areas
  - Mobile network operation
  - Internet of Things
  - Automotive
  - Media

- Big data in SK Telecom

- 1000+ nodes hadoop clusters
- 100+ nodes kafka clusters
- Data Pipelines with Hadoop, Hive, Flume, Kafka, HBase, Spark, Druid, Ignite, ElasticSearch, etc.



## Mobile network

Secured 30.16 million subscribers with more than 50 percent of market share in S. Korea (as of September 2017)



## IoT

Commercialized over 40 different IoT devices in S. Korea as a leader of the domestic IoT ecosystem (as of November 2017)



## Automotive

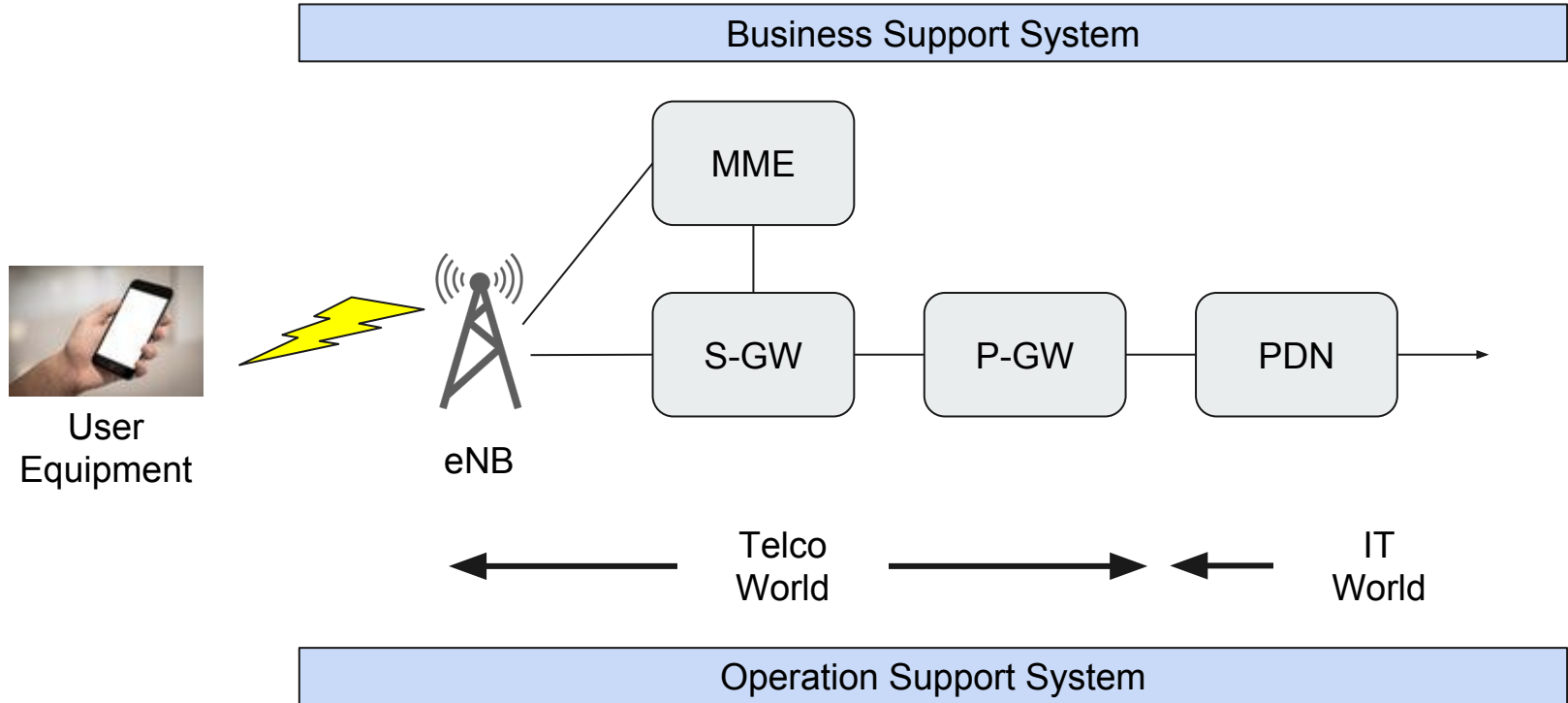
Demonstrated the world's first 5G-based connected car in partnership with BMW Korea and Ericsson (in 2016)



## Media

Reduced latency of live streaming to less than 3 seconds through 'T Live Streaming,' the world's first true real-time streaming technology (in 2016)

# Telco infra - in 1 minute





## 5G is Coming

- Virtualization...
- Virtualization...
- Virtualization...



# Telco infra monitoring & analytics

## Monitoring

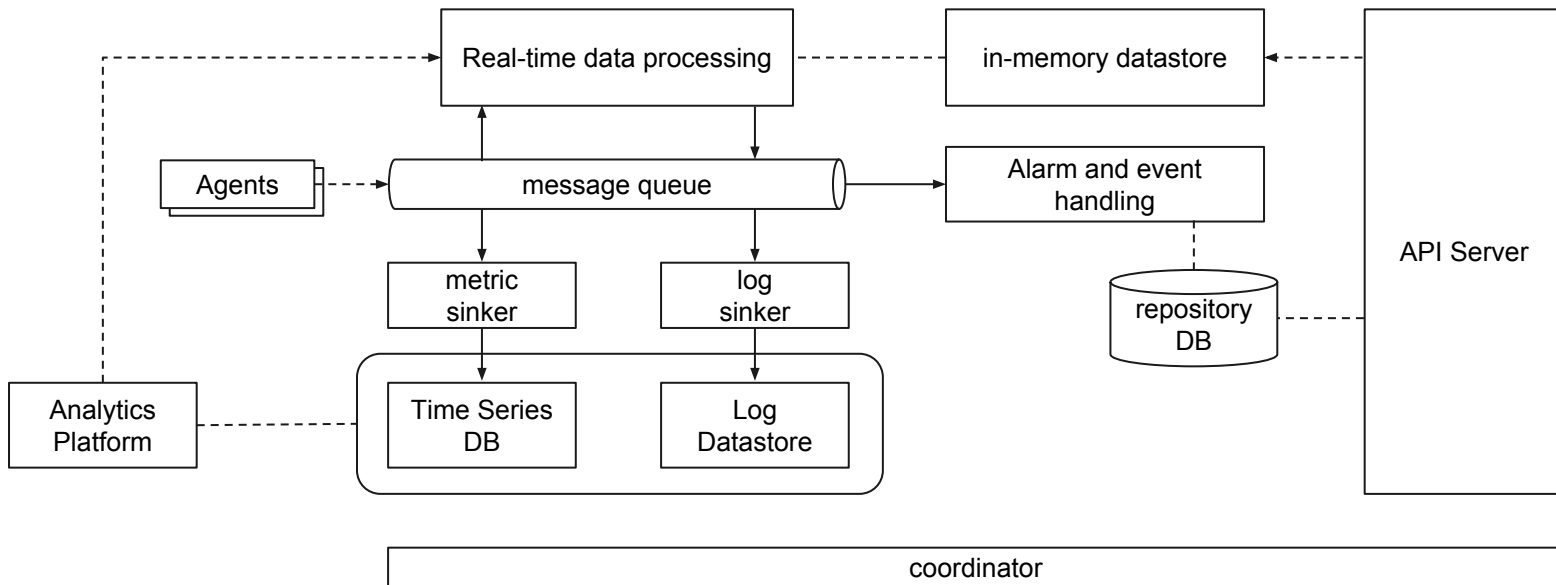
- Virtualized Network Functions and it's Element Management Systems
- Network Function Virtualization Infrastructures (Hypervisors)
- Several data centers and it's facilities
- Network status between data centers and/or regional hubs
- All IT infras including OSSs and BSSs

## Analytics

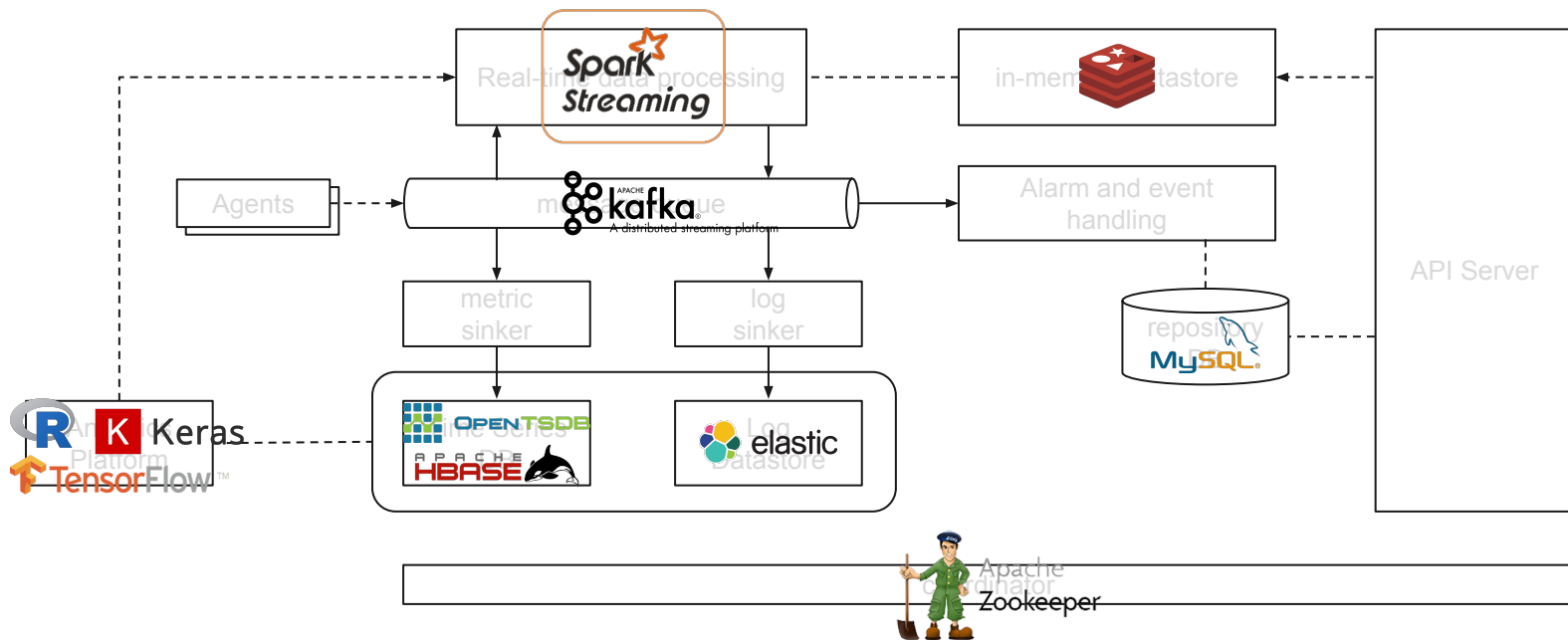
- IT and Network resource failure
- Network anomaly detection
- Resource engineering
- Automated dynamic resource allocation
- Network bandwidth management



# Architecture - simplified



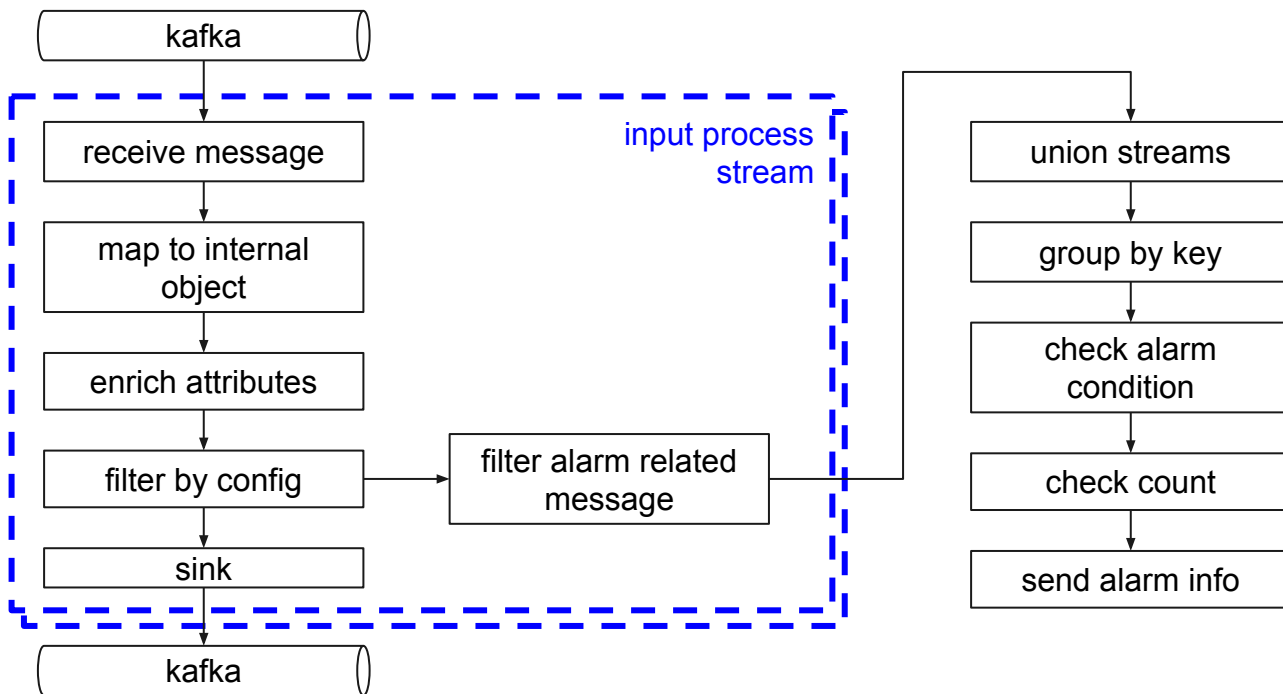
# Software stack



---

# Real-time processing of monitoring data

# Real-time data processing flow



# Real-time Processing Framework considered



**Spark Streaming**

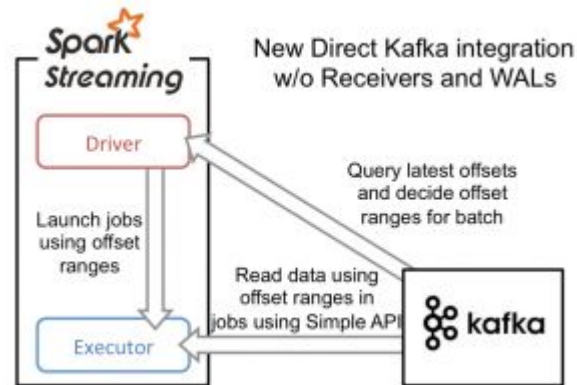
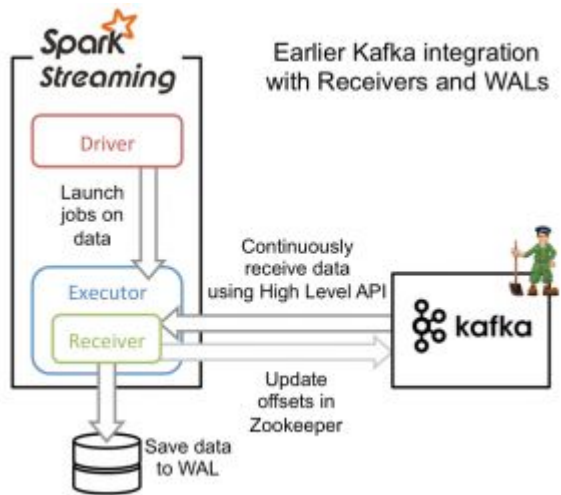


# Architectural issues faced

- Kafka integration with Direct Approach and too many number of partitions
- Integration with multiple kafka sources
- Passing changes to running spark streaming application
- Alarm condition check with Spark's micro-batch

# Kafka Direct Approach and too many number of partitions

Spark streaming's 2 different approaches with Kafka : Receiver-based v.s. Direct approach





# Kafka Direct Approach and too many number of partitions

- Direct Approach : 1 kafka partition = 1 RDD partition = 1 task = 1 cpu core
- Environment
  - Cluster : Standalone mode
  - 2 Worker nodes with SPARK\_WORKER\_CORES = 18 (36 cores total)
  - Data Source
    - 'metrics' topic partition # = 64, 'logs' topic partition # = 90 (means 154 tasks)
    - partition # not adjustable due to support agreement
  - Streaming application configuration
    - spark.executor.cores = 4
    - spark.cores.max = 20 (means spark can run 20 tasks simultaneously)
  - other spark applications are running on the cluster



# Kafka Direct Approach and too many number of partitions

## Sample case

- 100 topic partition
- cluster setting
  - 4 worker, 4 core for 1 worker
- application setting
  - 1 core for 1 executor
  - max 8 core
    - = 8 executors
    - = 8 tasks simultaneously
- Not always bad, but...
- Coalesce or Repartition





# Integration with Multiple Kafka Sources - what to consider

- Single application or dedicated applications?
  - Single application consumes every messages from across all of sources
    - options for limited resource situation but could be slowed down on many-partitioned topics
    - single interval for all sources
    - whole streaming job could be broken by partial failure
  - Dedicated application consumes messages from one source
    - each application can be optimized for the source,
    - resources can be configurable per application
    - less effective for identical batches on separated streaming applications



# Integration with Multiple Kafka Sources - what to consider

- Dynamic source change
  - Source change includes adding new topics/brokers, address change of existing source, removal of consuming source, interval of microbatch of source, etc.
  - Required to meet SLA on production environment (no downtime, etc)
  - What really required is high availability and low latency



# Integration with Multiple Kafka Sources - our approach

- Multiple kafka source consumption feature implementation
- Assigned each of the application handle multiple topics of single kafka broker
- Another options
  - structured streaming
  - vertically separated application

# Integration with Multiple Kafka Sources

```
try {
    List<JavaPairDStream<String, DataEntity>> streamList = new ArrayList<>(collectEntityList.size());
    JavaPairDStream<String, DataEntity> resultStream = null;

    for (CollectEntity collectEntity : collectEntityList) {
        ...

        String brokers = parseBrokerList(collectEntity.getKafkaBrokerList());
        kafkaParams.put("metadata.broker.list", brokers );
        ...

        JavaPairDStream<String, String> directKafkaStream = KafkaUtils.createDirectStream(ssc, String.class, String.class,
            StringDecoder.class, StringDecoder.class, kafkaParams, topics );

        dataStream = directKafkaStream.transformToPair(rdd -> {
            ...

            return rdd.mapPartitionsToPair(new FunctionData(context, collectEntity, mappingTemplateFactory));
        });

        streamList.add(dataStream);
    }

    ...
    resultStream = ssc.union(streamList.get(0), streamList.subList(1, streamList.size()));
    ...
}
```



# Passing configurations to running spark streaming application

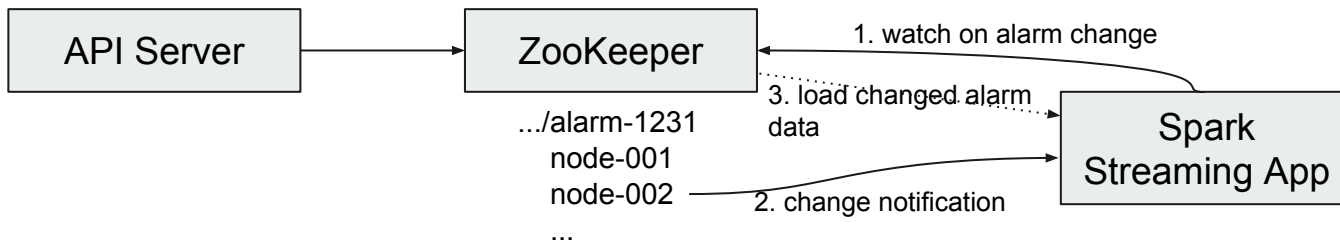
Changes on

- resource information : servers, switches, devices, parts, settings
- monitoring information : metrics, agent configurations, mapping with collecting items
- alarm rule : conditions, targets, checking intervals
- processing rule : mapping and filtering rule
- anomaly detection rule : data and feature generation rule, models
- real-time data supply rule

# Passing configurations to running spark streaming application

How we did before

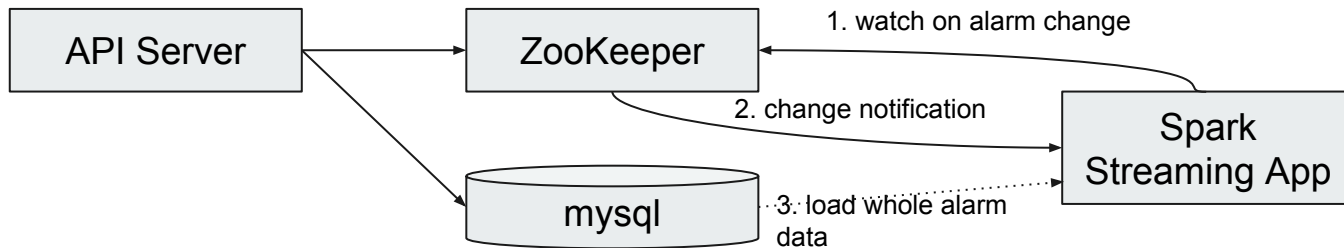
- Zookeeper for sharing data, mysql for storing data
- All applications, including spark streaming application, are subscribed to zookeeper to receive data change notification signal
- Read all of changed information directly from zookeeper
- Worked well for a while. But...




# Passing configurations to running spark streaming application

2nd step

- Notifier only role for Zookeeper
- Directly load changed data from mysql when notified
- Once per microbatch (when change detected) not once per config change







# Passing configurations to running spark streaming application

Now with Structured Streaming

- Redis for data sharing. Redis pub/sub for change notification
- 3 types of approach tested
  - a. Access redis data directly when needed
  - b. Access redis data on change -> store data in broadcast variable -> use UDF for data match/filter
  - c. Access redis data on change -> store in DataFrame -> use join for data match/filter
- Load all or update changes?
- “Always best” approach doesn’t exist

500k rules total 30k tps input data 5k rule changes (every 10s)	Average time to update changes to spark executor	Average delay to detect 0.9M alarms on 30k tps incoming data
a. Access redis directly	N/A	59.8sec
b. use broadcast variable & filter	7.8s	0.5s
c. use DataFrame & join	16.3s, 3.0s when update changes only	1.5s, 1.2s



# Event detection

What to consider

- What type of event patterns will be supported?
- How to organize collected data?
- Which tool is best for checking event conditions?
- How to express checking condition for storing and sharing?

Our tool

- Spring Expression Language (SpEL)
  - Part of Spring framework
  - Supports evaluation



# Event detection

Event patterns (for single target)

- Threshold check
  - `cpu.percent > 90%`
  - `disk.partition.use_percent > 90%` more than 10 minutes
- Log text check
  - “System Halted” in tomcat log message
  - “ERROR” in `/opt/application/log.txt` appears more than 3 times in 10 minutes
- Logical operation of patterns
  - Condition A OR Condition B AND Condition C



# Event detection

```
...
ExpressionParser expressionParser = new SpelExpressionParser();
Expression expression;

for (AlarmRuleTemplate[] templates : alarmRuleEntity.getTemplateList()) {
    ...
    boolean alert = false;
    for (AlarmRuleTemplate template : templates ) {
        templateParser = new MetricTemplateParser(template.getTemplate());
        for (AlarmDataEntity entity : alarmDataEntityList) {
            templateParser.setAttribute(entity.getMetric(), entity.getValue());
        }
        ...
        expression = expressionParser.parseExpression(parsedTemplate);
        alert = expression.getValue(Boolean.class);
        if ( !alert ) break;
    }

    if ( alert ) {
        alarmCheckData.addAlarmCheck(templates);
    }
}
```



# Event detection

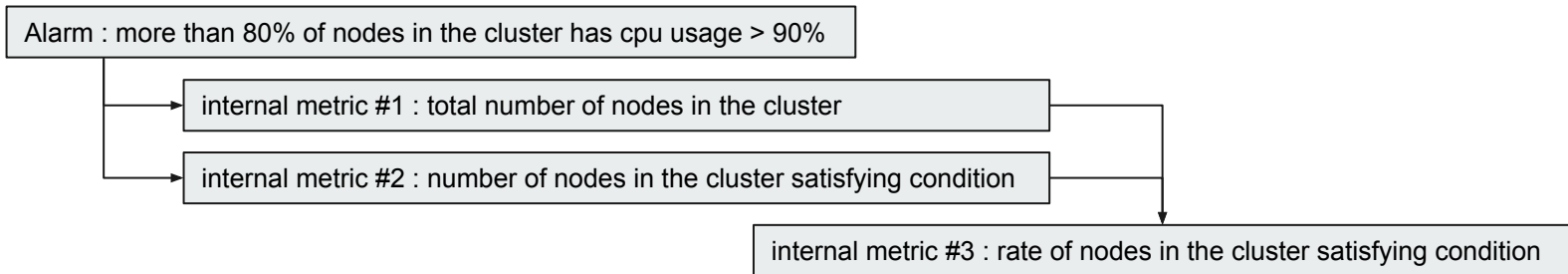
What we're working on

- correlation support
  - Check series of events
    - “ERROR” appeared in application log followed by “Exception” in 10 minutes
  - Aggregation of status
    - more than 70% of “spark cluster” is satisfying condition of `cpu.percent > 90%`
    - one instance of “myprocess” is running on one of 3 nodes
- movement check
  - `memory.percent` keep increasing during last 10 hours with 5 % of exceptions

# Event detection

What we're working on (continued)

- Separating event check stage with alarm check stage
  - Decompose alarm definition to predefined event patterns  
ex) Threshold check pattern : metric comparison\_operator value
  - Event check first. Use results on checking alarm
  - Sometimes aggregation and details of event check result needed to check alarm condition
- Temporary internal metric
  - Temporarily calculated by real-time processing engine for internal purpose
  - Mainly used for correlation alarm check





# Moving to Structured Streaming

- Easier API and SQL support - faster development
- Catalyst optimizer - better than human in most of the cases
- Dynamic query change support - searching for best way

---

# Storing and querying time-series data





# Our Data

- metric data
  - numeric, time-series
  - count, rate (%), time (msec, sec, ...), size (byte, mbyte, ...), throughput (bps, tps, ...), ...
- log data
  - file appended text. 1 line / multi-line text
  - non-periodic metric data parsed from log
- labels and value list
  - non-numeric, time-series
  - process status (dead, alive, ...), network status (listen, time\_wait, close\_wait, ...)
- objects
  - structured objects. mainly JMX objects
- asset information
  - H/W information, periodic update
  - Manufacturer, model, specs, parts, temperature, fan speed, ...



## Time-series datastores considered



OPENTSDDB



*influxdb*



**OpenTSDB**



# OpenTSDB

## Brief introduction

- Time-series datastore for only numeric data
- Started at 2010. Current stable version 2.3.1
- HBase backend (Supports Cassandra backend from 2.3.0)
- Clustering supported by HBase clustering
- Http, telnet API support / Java API
- LGPL v2.1+ (perfectly compatible with Apache license v2) - <http://opentsdb.net/faq.html>



# Data on OpenTSDB

- Data format

- `<metric name> <tag list> <timestamp> <value>`
- Sample

```
cpu.sys_percent host=server001,ip=192.168.10.11,group=main,role=web 1536576293 12.3
```

```
cpu.sys_percent host=server001,ip=192.168.10.11,group=main,role=web 1536576294 13.1
```

```
cpu.sys_percent host=server001,ip=192.168.10.11,group=main,role=web 1536576295 13.4
```

- Target is specified by tag list
- Long / many tag means performance downs

- Our tag list

- `id` : id of target server
- `host` : hostname of target at capture time
- `location` : additional information to specify target (interface, partition, device, port, etc)



# Issues faced

- Slow operation on data store affects real-time processing
- Strange aggregation result
- Tough maintenance
- New requirements and slowed down developer community activity



# Slow operation on data store affects real-time processing

Interface options tested : import cli, http api, Java API

import cli	Http API	Java API
Streaming application sinks to file per micro-batch. bulk load with import cli	Streaming application sinks to http api per micro-batch	Streaming application sinks to opentsdb using Java API (net.opentsdb.core.TSDB class)
<ul style="list-style-type: none"><li>● tight validation with timestamp</li><li>● error handling issue</li></ul>	<ul style="list-style-type: none"><li>● slower than Java API</li></ul>	<ul style="list-style-type: none"><li>● tight validation with timestamp</li><li>● directly affected by status of backend engine (HBase)</li></ul>



# Slow operation on data store affects real-time processing

## OpenTSDB Java API

- provided by TSDB class
  - OpenTSDB Java API
  - interface with HBase directly

net.opentsdb.core

### Class TSDB

[java.lang.Object](#)

└ net.opentsdb.core.TSDB

---

```
public final class TSDB
extends Object
```

Thread-safe implementation of the TSDB client.

This class is the central class of OpenTSDB. You use it to add new data points or query the database.



# Slow operation on data store affects real-time processing

Calling Java API from spark streaming application

- Spark streaming application interacts with HBase directly
  - Delayed by heavy load on HBase (compaction, GC, etc)
  - OpenTSDB itself performs hourly compaction to rearrange data and it takes some time
- Singleton pattern
  - OpenTSDB connection pool per executor

```
dstream.foreachRDD { rdd =>
  rdd.foreachPartition { partitionOfRecords =>
    // ConnectionPool is a static, lazily initialized pool of connections
    val connection = ConnectionPool.getConnection()
    partitionOfRecords.foreach(record => connection.send(record))
    ConnectionPool.returnConnection(connection) // return to the pool for future reuse
  }
}
```





# Slow operation on data store affects real-time processing

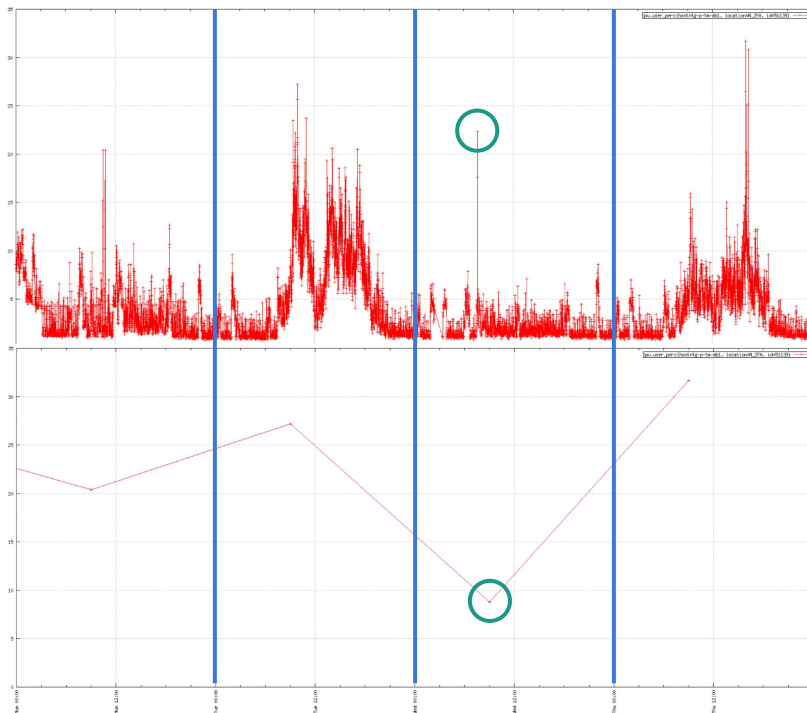
## Our solution

- Separation of concerns
  - Requirement #1 : Immediate alert notification without delay
- New approach
  - Spark streaming application -> kafka -> Data sinker process
  - Assign thread per kafka topic partition to consume metric
  - Write queue to sink data to OpenTSDB
- Problem on data store doesn't cause delay on stream processing anymore

# Strange aggregation result

Query result on same data with same time range

- 1st case : raw data plotting
- 2nd case : daily aggregated max



# Strange aggregation result

OpenTSDB time bucket setting for downsample aggregation

- $\text{timestamp} - (\text{timestamp} \% \text{interval\_ms})$

Our case

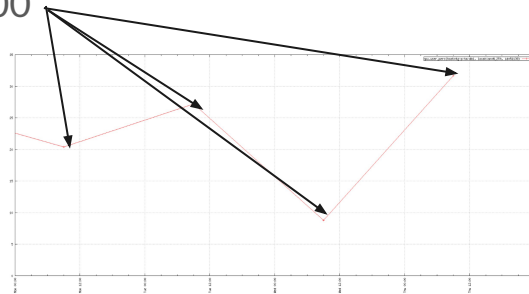
- start = 2017/05/01 (Localtime)  
= 1,493,564,400 in epoch timestamp
- sample\_interval\_ms = 1 day = 86,400,000 ms
- interval\_aligned\_ts = 1,493,510,400 s = 2017/04/30 09:00 (Localtime)
- Daily aggregation performed for every 24 hours beginning from 09:00

Reason

- OpenTSDB doesn't consider timezone while calculating start point
- Policy or issue?

```
long interval_aligned_ts = start;
if (0L != sample_interval_ms) {
    // Downsampling enabled.
    final long interval_offset = (1000L * start) % sample_interval_ms;
    interval_aligned_ts -= interval_offset / 1000L;
}

// Then snap that timestamp back to its representative value for the
// timespan in which it appears.
final long timespan_offset = interval_aligned_ts % Const.MAX_TIMESPAN;
final long timespan_aligned_ts = interval_aligned_ts - timespan_offset;
```





# Tough maintenance and environment

- Huge gap between solution development and production system maintenance
- Too many layers with OpenTSDB
  - HDFS, HBase, and OpenTSDB with ZooKeeper
  - Maintenance complexity increased exponentially
- Not familiar with operational issues
  - Timeout exception between hdfs datanode
  - Writing performance issue during HBase compaction or GC time
  - Configuration optimization and performance tuning
  - Region down and recovery using WAL (Write Ahead Log)
- Limited resources made things harder
  - All modules installed on 3 nodes with 8~16 cores each
    - OpenTSDB, HBase, Hadoop, Spark, Kafka, Zookeeper, Elasticsearch, Redis, ...

# New requirements and slowed down developer community activity

New Requirements with time-series data

- Rollup and pre-aggregation
- Non numeric (label / value list) data type support
- Object data type support

Slow OpenTSDB developer community activity



---

# Analytics and real-time inferencing



# What we're doing

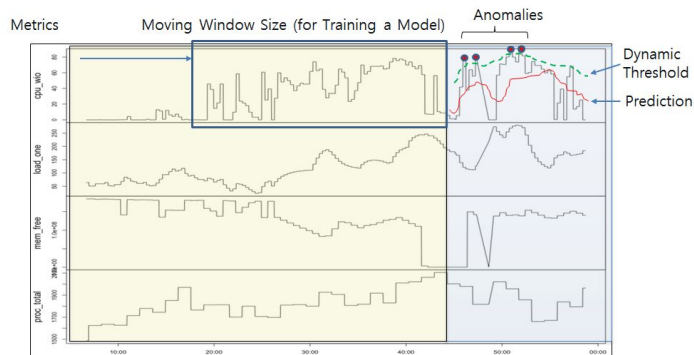
Anomaly detection with

- system metrics
- log clustering

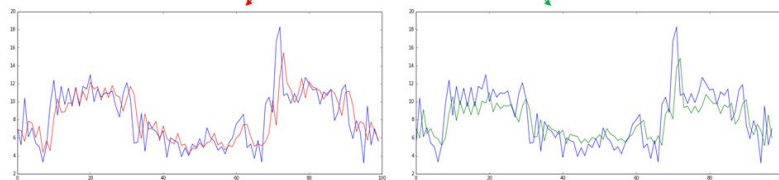
Real-time inferencing

# Metric Analysis

- Goals of metric analysis
  - Predictive maintenance and resource engineering
  - Decrease false alarm
- Topics
  - Anomaly detection with multiple system metrics and application metrics
  - Dynamic threshold with each of forecasted system metric



Resource	Metric	Prediction Model				
		ARIMA (p=5)	ARIMA (p=16)	LSTM (lookback = 1)	LSTM (lookback = 16)	DNN
Tango-A/ tg-s-ad-iris1	CPU_PERC	2.213	2.136	2.28	1.95	2.30



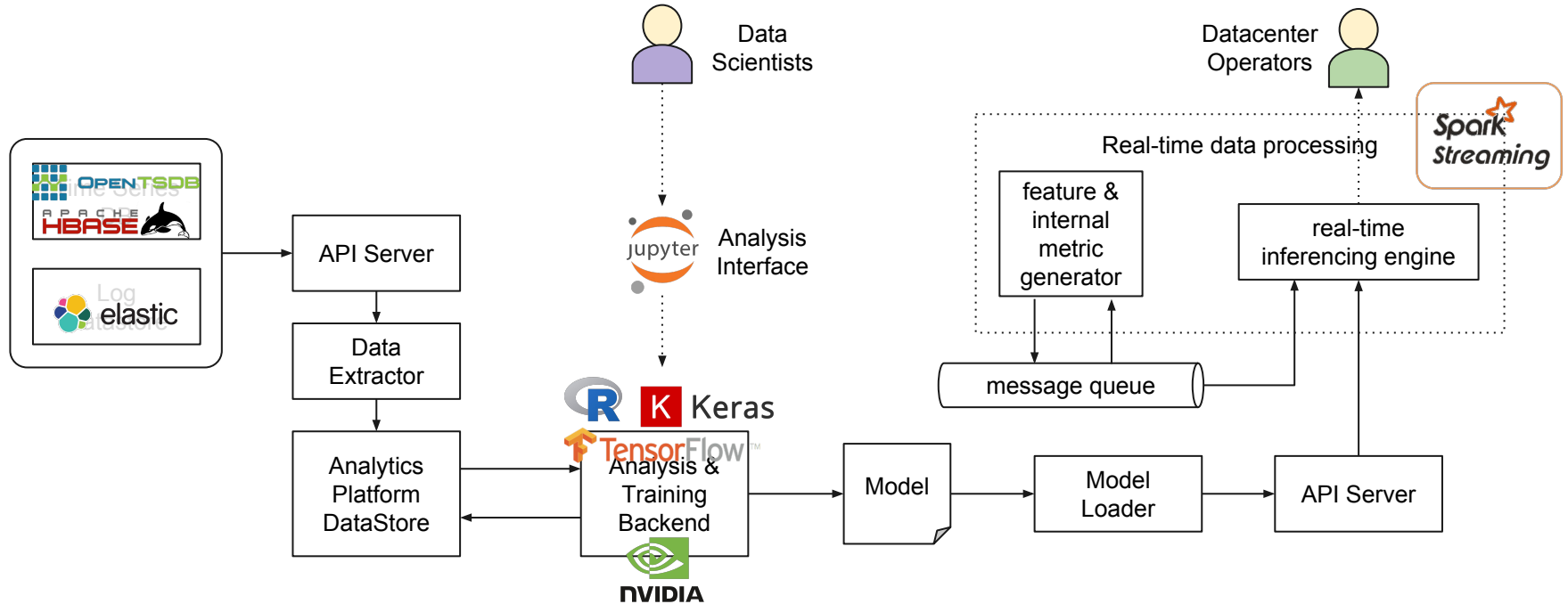


# Log Analysis

- Goals of log analysis
  - Anomaly detection with log pattern changes on time
- Our approach
  - Define distance function for calculating distance of logs
  - Distance for cluster member decision and distance function can be configured

Log cluster	Count
Apr 20 15:37:49 edw-004-08 kernel: EXT4-fs error (device bcache1): <u>mb_free_blocks</u> : double-free of <u>inode 0</u> 's block 48775 0387(bit 31475 in group 14884)	9882
Apr 20 15:37:50 edw-004-08 kernel: EXT4-fs error (device bcache1): <u>mb_free_blocks</u> : double-free of <u>inode 0</u> 's block 48775 0417(bit 31505 in group 14884)	
Apr 20 15:37:50 edw-004-08 kernel: EXT4-fs error (device bcache0): <u>mb_free_blocks</u> : double-free of <u>inode 0</u> 's block 48775 0419(bit 31507 in group 14884)	
....	
Apr 20 16:37:05 edw-004-08 kernel: EXT4-fs error (device bcache5): ext4_mb_generate_buddy: EXT4-fs: group 20565: 13 864 blocks in bitmap, 15544 in <u>gd</u>	99
Apr 20 16:37:05 edw-004-08 kernel: JBD: Spotted dirty metadata buffer (dev = bcache5, <u>blocknr = 0</u> ). There's a risk of <u>filesystem</u> corruption in case of system crash.	6
Apr 21 17:15:32 edw-004-08 kernel: possible SYN flooding on port 13562. Sending cookies.	4
Apr 20 16:43:02 edw-004-08 root: edw-004-08	3
Apr 21 01:00:05 edw-004-08 kernel: EXT4-fs error (device bcache4): ext4_mb_free_metadata: Double free of blocks 22528 (16384 8192)	1
Apr 21 05:07:01 edw-004-08 <u>auditd</u> [3353]: Audit daemon rotating log files	1
Apr 21 23:14:31 edw-004-08 <u>ntpd</u> [3622]: 0.0.0.0 0618 08 <u>no_sys_peer</u>	1

# Analytics Platform





# Summary & Lessons Learned

- Long time of optimization with real data on production environment is essential especially with Streaming application
- Technical expert is essential for both development and stable service
- Community activity and future roadmap should be considered as well as maturity of software



**Thank You!**