Logisland

Event mining at scale

Thomas Bailet @hurence [2018]

Overview

Logisland

provides a stream analytics solution

that can handle all enterprise-scale

event data and processing

Big picture

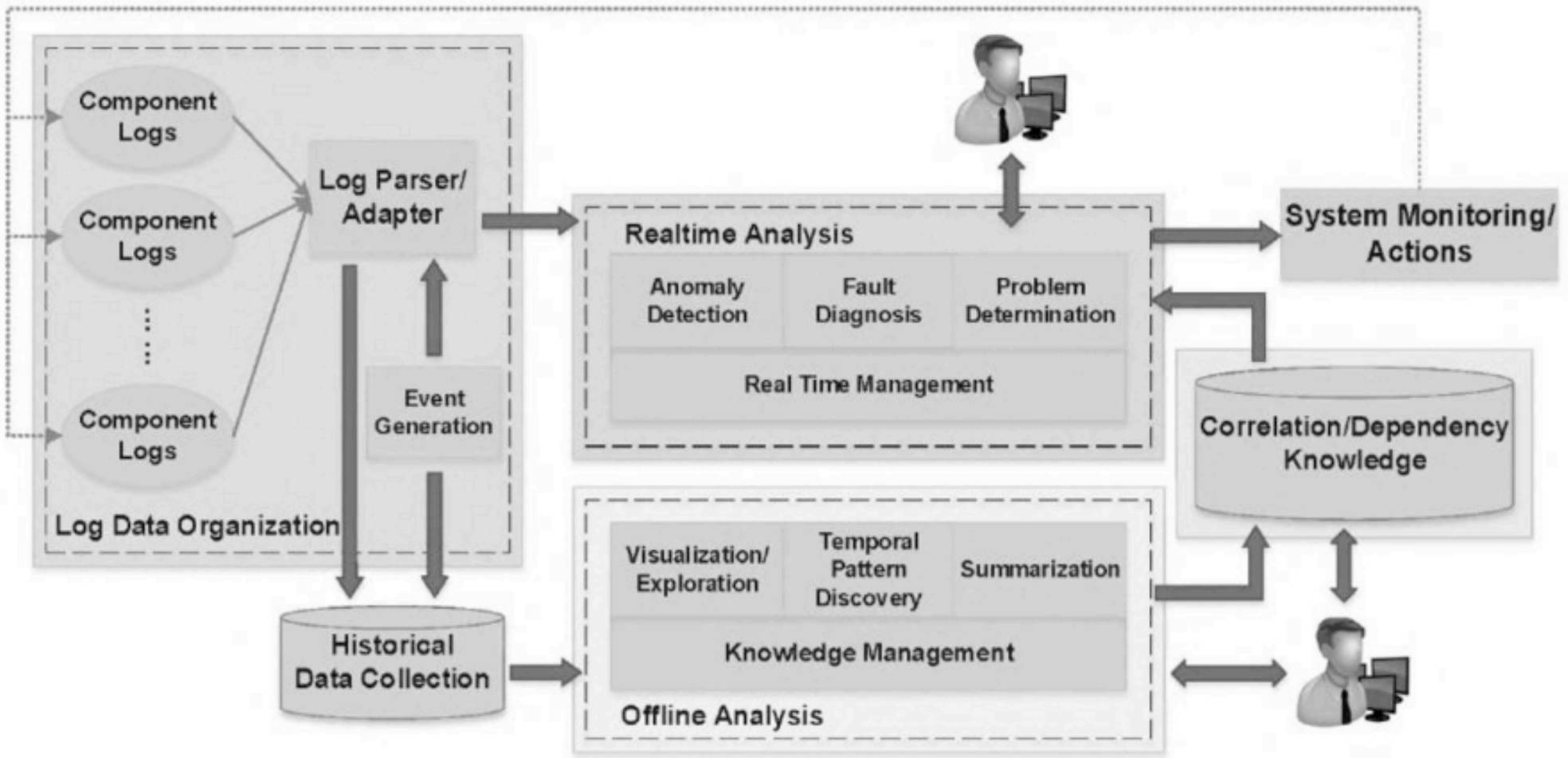
- **Open source**, developed by Hurence
- **High scalability** and **Fault-tolerant**.
- **High throughput** (billions messages / day).
- **Easy** to operate on Hadoop or on lightweight containers
- **Extensible framework** to build high level apps
- Alternative to Splunk, StreamAnalytix, Nifi...

Adoption

- **La Française des Jeux** : cybersecu
- **SEB** : (I)IOT
- **Coservit** : IT monitoring / smart alerting
- **IPH (Rubix, Orexad, ...)** : web analytics
- **ST microelectronics** : smart alerting
- **Sodexo** : time-series forecasting
- **IFPEN** : data historian

Purpose

- realtime data-mining
- complex event processing
- patterns finding
- reframing / normalizing / contextualizing
- click stream tracking
- data historian


Component Logs
Component Logs
Component Logs
Log Parser/ Adapter
Event Generation
Log Data Organization
Historical Data Collection
Realtime Analysis
Anomaly Detection
Fault Diagnosis
Problem Determination
Real Time Management
System Monitoring/ Actions
Correlation/Dependency Knowledge
Visualization/ Exploration
Temporal Pattern Discovery
Summarization
Knowledge Management
Offline Analysis

Why ?

- stream processing design pattern (best pratices and code reuse).
- tried most of the concurrent frameworks.
- need something that really scales.
- powerfull log-centric architecture

Stream Processing Patterns

- service injection
- classloading isolation
- metrics collection
- dynamic configuration

Concurrence

- **Beam/Flink** one of the best challenger before StructuredStreaming.
- **ELK** is great to start, but hard to centralize processing, lacks of offline ML & custom advanced analytics.
- **Splunk** is fantastic but clients are not rich enough to afford it ;)
- **NIFI** is a great tool but doesn't play well with distributed processing.
- **Metron**, **Eagle** are too security centric.

Features

- out-of-the-box processors (no code required)
- raw data to structured records conversion
- connectors (Oracle, Tibco, Hadoop, Cassandra, Mongo, OPC, SOLR, blockchain, MQTT ...)
- complex event processing, threshold based query matching
- realtime configuration update
- advanced self-monitoring

Features (advanced)

- high level extensible framework.
- stream governance with schema management.
- SQL aggregations.
- Time series sampling & forecasting.
- Outliers detection.
- Plugins management
- Machine & Deep Learning integration

Paradigm

Logisland continously transforms

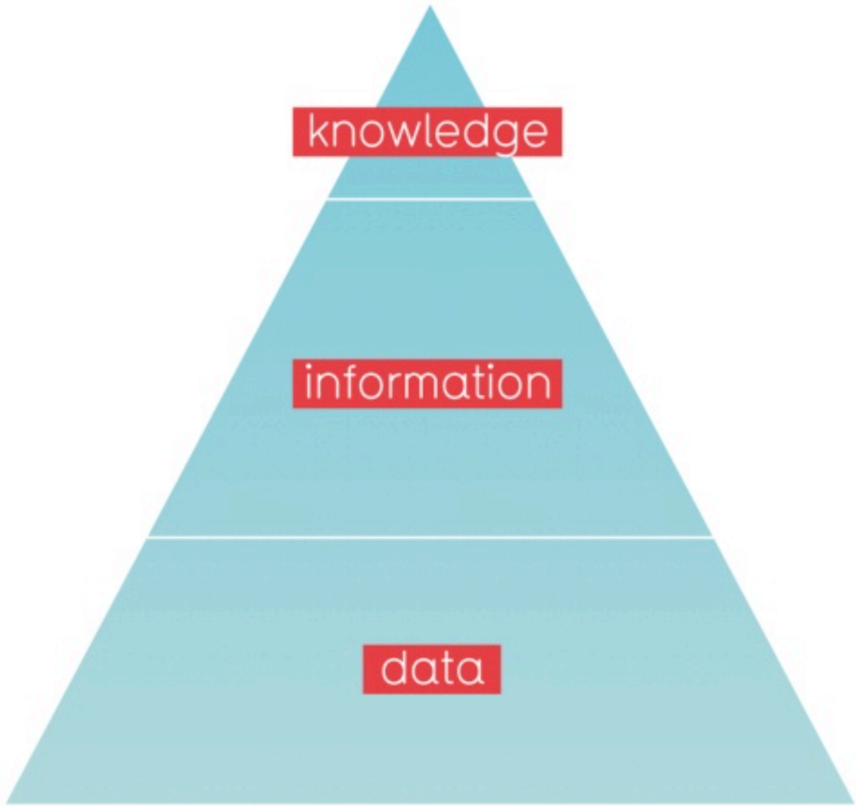
data into information &

information into knowledge

by using asynchronous processing on

increasingly abstract

and meaningfull records.



Use cases

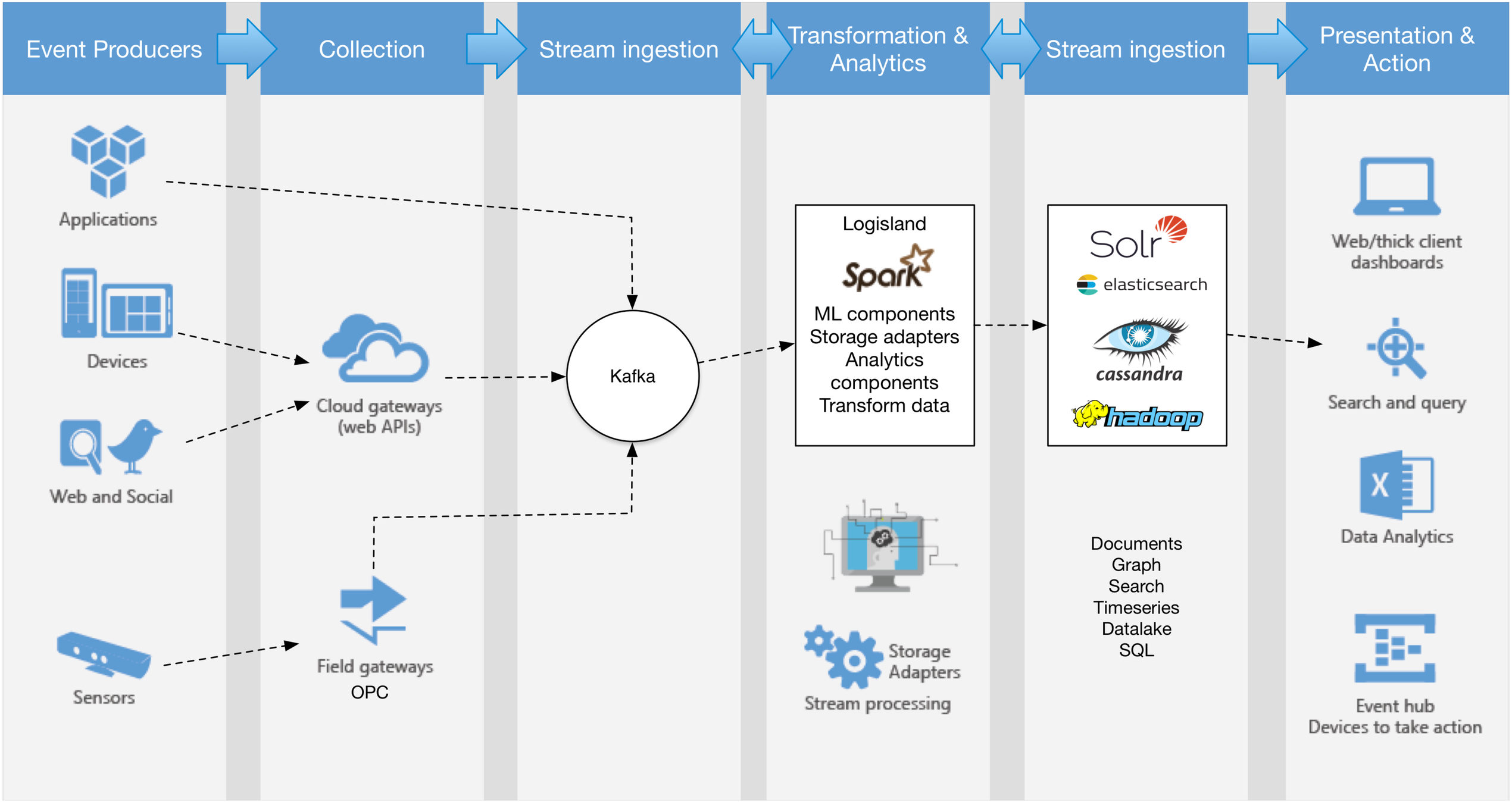
- **IT Monitoring** : data centralization & smart alerting
- **Event processing** : multi-layer processing, enriching, custom Business Rules activation, anomaly detection, usage profiling
- **Click stream tracking** : web analytics and sessionization.
- **Security** : cybersecurity, intrusion & fraud detection.
- **IoT** : alerting, forcasting and M2M communication.

La française des jeux sample

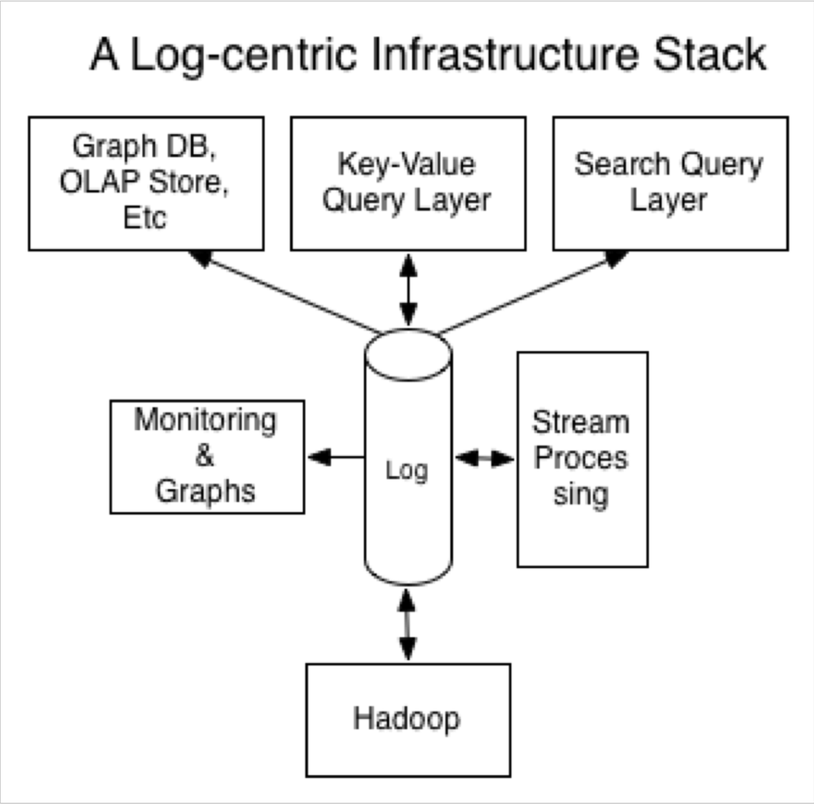
Example of one production cluster

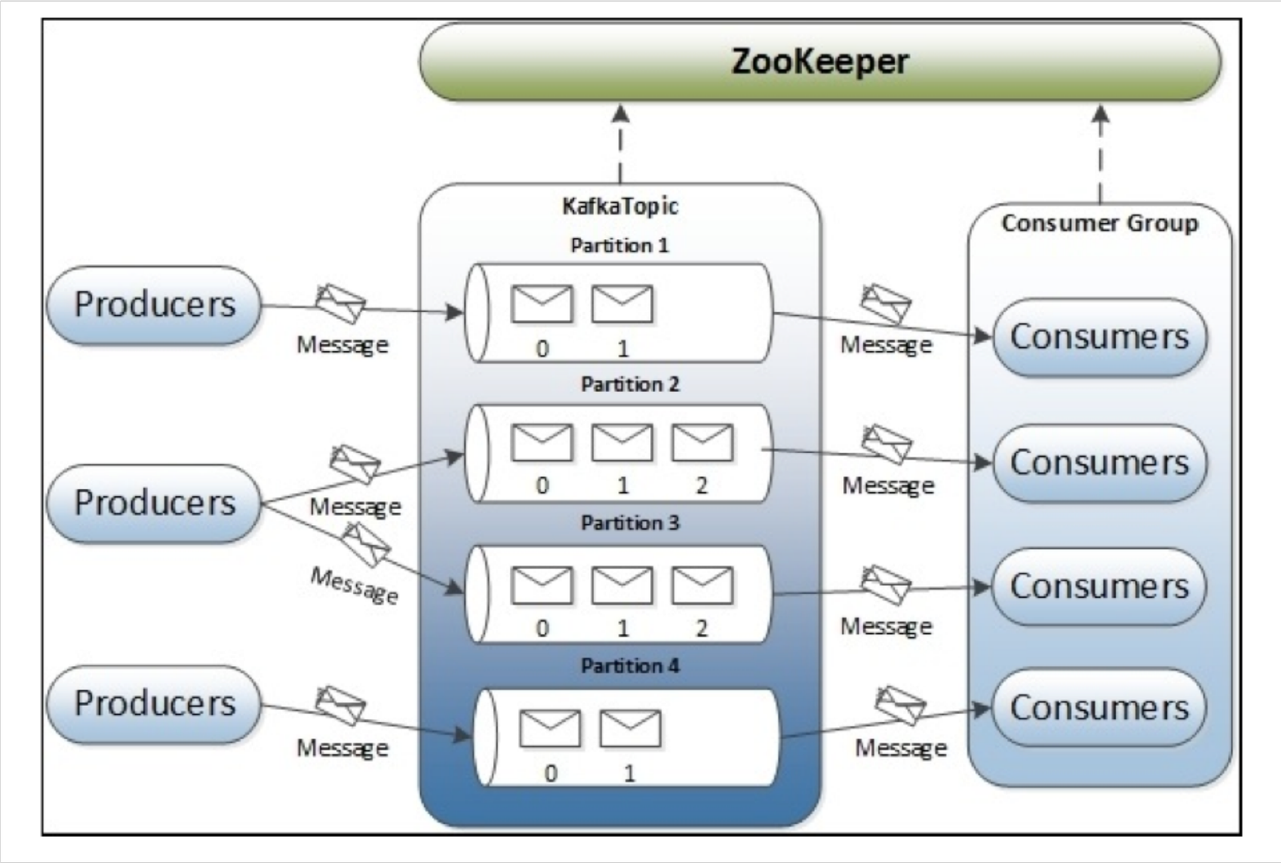
- 5 brokers
- 2000 partitions (replication factor 3)
- 100 000 msg/s

Design

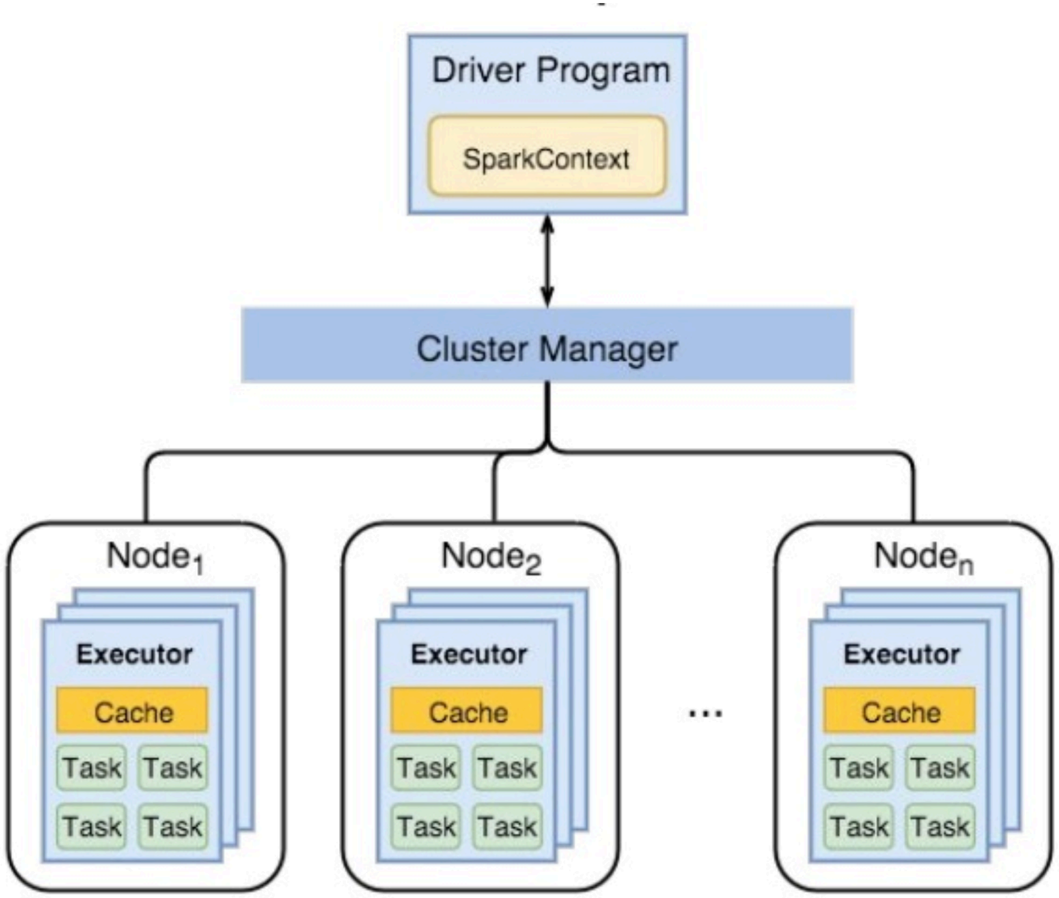

Event Producers
Collection
Stream ingestion
Transformation & Analytics
Stream ingestion
Presentation & Action
Applications
Devices
Web and Social
Sensors
Cloud gateways (web APIs)
Field gateways
OPC
Kafka
Logisland
Spark
ML components
Storage adapters
Analytics components
Transform data
Storage Adapters
Stream processing
Solr
elasticsearch
cassandra
hadoop
Documents
Graph
Search
Timeseries
Datalake
SQL
Web/thick client dashboards
Search and query
Data Analytics
Event hub
Devices to take action

Log-centric architecture



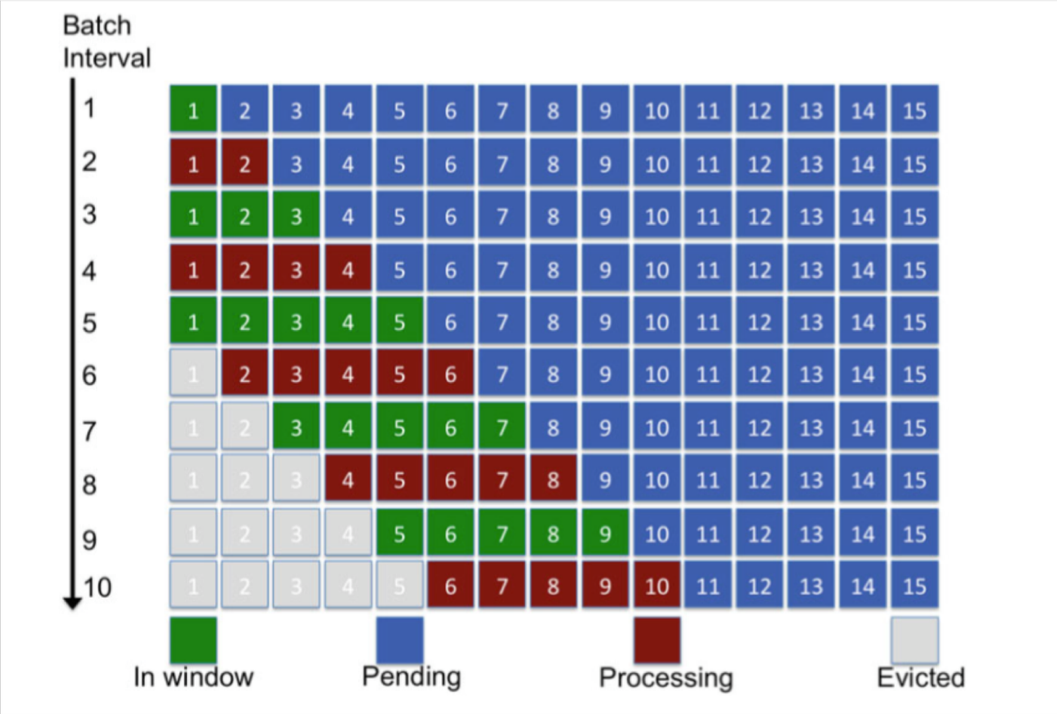


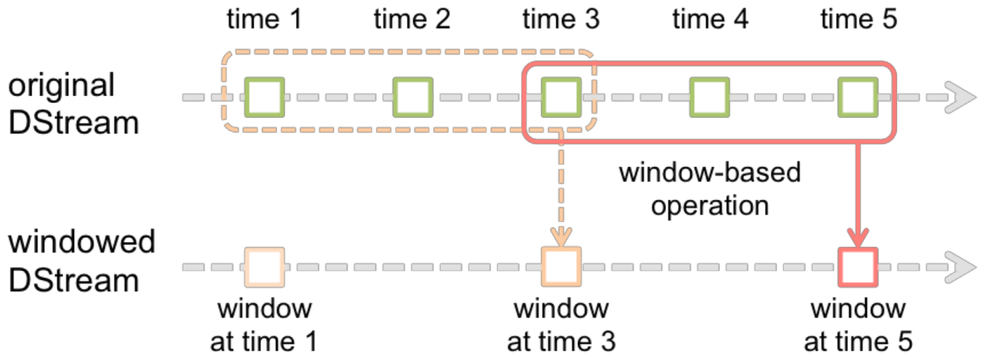
Grid computing



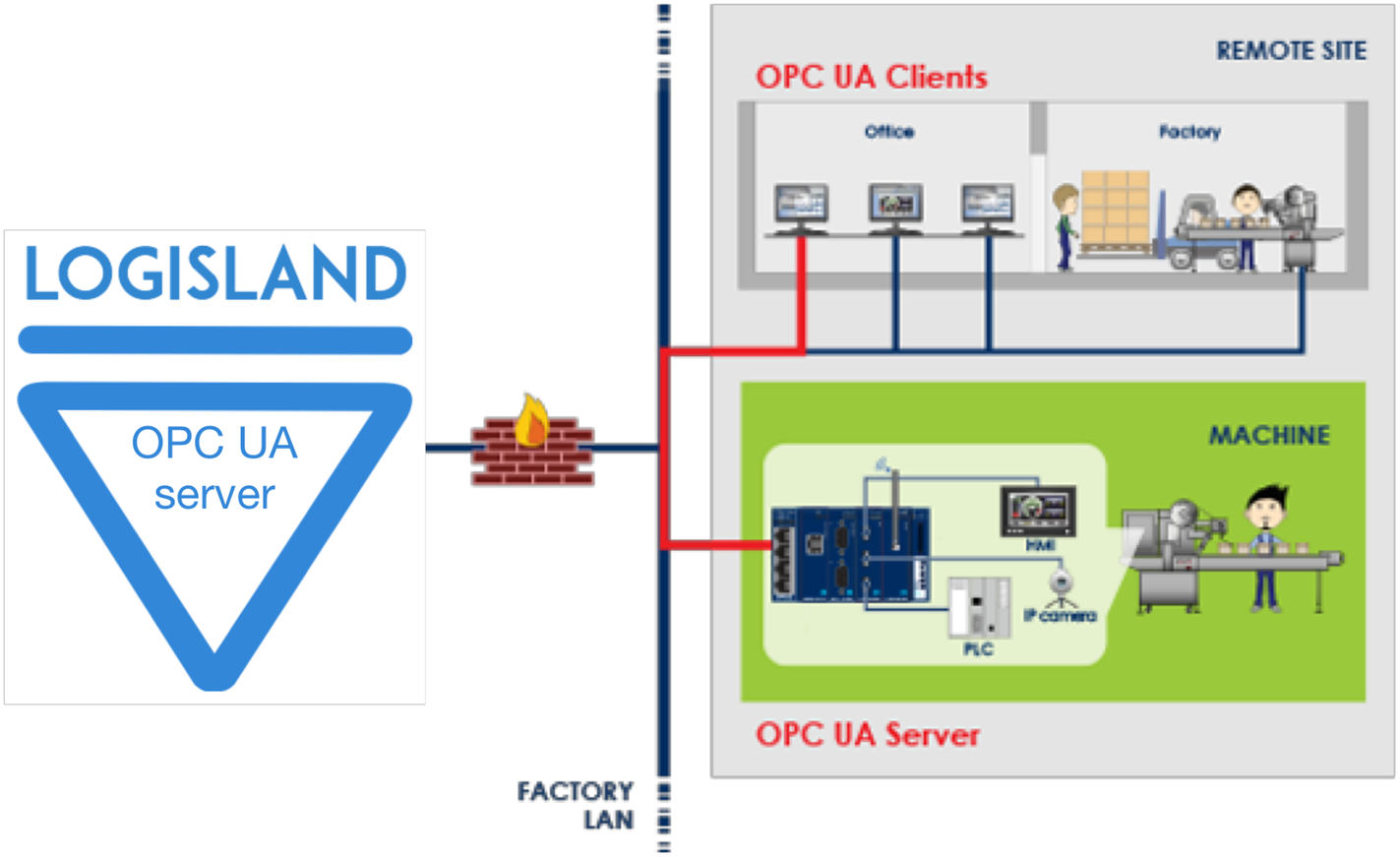
- Spark Driver
  - separate process to execute user applications
  - creates *SparkContext* to schedule jobs execution and negotiate with cluster manager
- Executors
  - run tasks scheduled by driver
  - store computation results in memory, on disk or off-heap
  - interact with storage systems
- Cluster Manager
  - Mesos
  - YARN
  - Spark Standalone

Micro-batch processing





OPC UA compliance (HDA)



API

Record

a basic unit of processing

Record

The basic unit of processing is the Record.
A `Record` is a collection of `Field`, while a `Field` has a `name`, a `type` and a `value`.

```
String id = "firewall_record1";
String type = "cisco";
Record record = new Record(type).setId(id);

assertTrue(record.isEmpty());
assertEquals(record.size(), 0);
```

A record is defined by its type and a collection of fields.
There are three special fields:

```
// shortcut for id
assertEquals(record.getId(), id);
assertEquals(record.getField(FieldDictionary.RECORD_ID).asString(), id);

// shortcut for time
assertEquals(record.getTime().getTime(),
    record.getField(FieldDictionary.RECORD_TIME).asLong().longValue());

// shortcut for type
assertEquals(record.getType(), type);
```

And the *standard* fields have generic setters, getters and removers

```
record.setStringField("url_host", "origin-www.20minutes.fr")
      .setField("method", FieldType.STRING, "GET")
      .setField("response_size", FieldType.INT, 452)
      .setField("is_outside_office_hours", FieldType.BOOLEAN, false)
      .setField("tags",
                FieldType.ARRAY,
                Arrays.asList("spam", "filter", "mail"));

assertEquals(record.getField("method").asString(), "GET");
assertTrue(record.getField("response_size").asInteger() - 452 == 0);
record.removeField("is_outside_office_hours");
assertFalse(record.hasField("is_outside_office_hours"));
```

Fields are strongly typed, you can validate them

```
Record record = new StandardRecord();
record.setField("request_size", FieldType.INT, 1399);
assertTrue(record.isValid());

record.setField("request_size", FieldType.INT, "zer");
assertFalse(record.isValid());

record.setField("request_size", FieldType.DOUBLE, 45.5d);
assertTrue(record.isValid());

record.setField("request_size", FieldType.STRING, 45L);
assertFalse(record.isValid());
```

Processor

a reusable processing component

Processor

Logisland is a component centric framework,

It's built over an abstraction layer to build configurable components.

A component can be `Configurable` and `Configured`.

The most common component you'll use is the `Processor` which takes a collection of `Record` and publish another collection of records

```
public interface Processor extends ConfigurableComponent {

    /**
     * Setup stateful parameters
     */
    void init(final ProcessContext context);

    /**
     * Process the incoming collection of records to
     * generate a new collection of records
     */
    Collection<Record> process(ProcessContext context,
                               Collection<Record> records);
}
```

Sample Processor config

```
- processor: apache_parser
  component: com.hurence.logisland.processor.SplitText
  type: parser
  documentation: a parser for apache log REGEX
  configuration:
    record.type: apache_log
    value.regex: (\S+)\s+(\S+)\s+(\S+)\s+\[([\w:\/] ...
    value.fields: src_ip,identd,user,record_time,http_method, ...
```

ControllerService

a shared executor-bounded service

ControllerService injection

share access to external Services across the Processors,
for example bulk buffers or client connections to external data sources.

For example a cache service that could cache K/V tuple across the worker node.

ControllerService definition

```
public interface CacheService<K,V> extends ControllerService {

    PropertyDescriptor CACHE_SIZE = new PropertyDescriptor.Builder()
            .name("cache.size")
            .description("The maximum number of element in the cache.")
            .required(false)
            .defaultValue("16384")
            .addValidator(StandardValidators.POSITIVE_INTEGER_VALIDATOR)
            .build();

    public V get(K k);

    public void set(K k, V v);
}
```

ControllerService config

The injection is done through yaml config files by injecting the instance of `lru_cache` Service.

```
controllerServiceConfigurations:
  - controllerService: lru_cache
    component: com.hurence.logisland.service.elasticsearch.LRUKeyValueCacheService
    configuration:
      cache.size: 5000
streamConfigurations:
  - stream: parsing_stream
    component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
    processorConfigurations:
      - processor: mock_processor
        component: com.hurence.logisland.processor.TestProcessor
        configuration:
           cache.service: lru_cache
```

Stream

a distributed processing pipeline

Stream

a record `Stream` basically :

- reads a distributed collection of `Record` from Kafka input topics
- transmits them to a chain of `Processor`
- write the processed `Records` to some Kafka output topics

```
public interface RecordStream extends ConfigurableComponent {

    void start();
    void stop();
}
```

Streaming paradigm

You can handle partionned data in 3 ways :

- **fully in parrallel**, eg. a thread by RDD/kafka partition, like with `KafkaRecordStreamParallelProcessing`, when records have no link with each other
- by **joining partitions** like with `KafkaRecordStreamSQLAggregator` or `KafkaRecordStreamHDFSBurner` when you need to join related records (costly join and shuffling operations)
- with **Structured Streaming**

Sample Stream configuration

```
- stream: parsing_stream
    component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
    configuration:
        kafka.input.topics: logisland_raw
        kafka.output.topics: logisland_events
        kafka.error.topics: logisland_errors
        kafka.input.topics.serializer: none
        kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
        kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    ...
    processorConfigurations:
```

HDFS burner stream

```
- stream: events_burner
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamHDFSBurner
  type: stream
  documentation: average bytes sent by host_name
  configuration:
    kafka.input.topics: logisland_events
    kafka.output.topics: none
    kafka.metadata.broker.list: ${KAFKA_BROKERS}
    kafka.zookeeper.quorum: ${ZK_QUORUM}
    output.format: parquet
    output.folder.path: data/syslog_events
```

Aggregation stream

```
- stream: metrics_by_host
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamSQLAggregator
  configuration:
    kafka.input.topics: logisland_events
    kafka.output.topics: logisland_aggregations
    window.duration: 10
    avro.input.schema: >
      {  "version":1,
         "type": "record",
         "name": "com.hurence.logisland.record.apache_log",
         "fields": [
         ...
           { "name": "bytes_out",   "type": ["long","null"] },
           { "name": "http_query",  "type": ["string","null"] },
           ...
            ]}
    sql.query: >
      SELECT count(*) AS connections_count, avg(bytes_out) AS avg_bytes_out, src_ip, first(record_time) as record_time
      FROM logisland_events
      GROUP BY src_ip
      ORDER BY connections_count DESC
      LIMIT 20
    max.results.count: 1000
    output.record.type: top_client_metrics
```

Structured streaming in logisland

Structured Streaming is a scalable and fault-tolerant stream processing engine built on the Spark SQL engine.

```
stream: parsing_stream_source
    component: com.hurence.logisland.stream.spark.structured.StructuredStream
    configuration:
        read.topics.client.service: kc_source_service
        write.topics.client.service: kafka_out_service
        processorConfigurations:
        - processor: flatten
            component: com.hurence.logisland.processor.FlatMap
            configuration:
                keep.root.record: false
                copy.root.record.fields: true
```

Spark structured stream vs Flink

- Spark Structured Streaming has still microbatches used in background
- it supports event-time processing with quite low latency
- supports SQL and type-safe queries on the streams in one API
- It has end-to-end exactly-one semantics (at least they says it ;))
- The throughput is better than in Flink
- Spark Continous Processing Mode is in progress and it will give Spark ~1ms latency, comparable to those from Flink.

Plugins management

classloading proxyfication

```
Using Logisland home: /usr/local/bin/logisland-1.0.0-RC1

usage: components.sh [-h] -i <artifact> | -l | -r <artifact>
 -h,--help                  Print this help.
 -i,--install <artifact>    Install a component. It can be either a logisland plugin or a kafka connect module.
 -l,--list                  List installed components.
 -r,--remove <artifact>     Removes a component. It can be either a logisland plugin or a kafka connect module.
```

Plugin example

```
Artifact: com.hurence.logisland:logisland-processor-common:1.0.0-RC1
Name: Common processors bundle
Version: 1.0.0-RC1
Location: /usr/local/bin/logisland-1.0.0-RC1/lib/logisland-processor-common-1.0.0-RC1.jar
Components provided:
    com.hurence.logisland.processor.AddFields
    com.hurence.logisland.processor.ApplyRegexp
    com.hurence.logisland.processor.ConvertFieldsType
    com.hurence.logisland.processor.DebugStream
    com.hurence.logisland.processor.EvaluateJsonPath
    com.hurence.logisland.processor.FilterRecords
    com.hurence.logisland.processor.FlatMap
    com.hurence.logisland.processor.GenerateRandomRecord
    com.hurence.logisland.processor.ModifyId
    com.hurence.logisland.processor.NormalizeFields
    com.hurence.logisland.processor.ParseProperties
    com.hurence.logisland.processor.RemoveFields
...
```

Engine

an execution model abstraction

Engine

The `Engine` manage a collection of `Stream`

this is the abstraction of the execution model, mainly in Spark actually but plans are to integrate Beam to move on Storm and Kafka Streams

you configure here your Spark job parameters

```
public interface ProcessingEngine extends ConfigurableComponent {

    void start(EngineContext engineContext);
    void shutdown(EngineContext engineContext);
}
```

Kafka connect integration

Kafka Connect, an open source component of Apache Kafka, is a framework for connecting Kafka with external systems such as databases, key-value stores, search indexes, and file systems.

Using Kafka Connect you can use existing connector implementations for common data sources and sinks to move data into and out of Kafka.

Kafka connect to the blochain

```
- controllerService: kc_source_service
    component: com.hurence.logisland.stream.spark.provider.KafkaConnectStructuredSourceProviderService
    configuration:
        kc.data.value.converter: com.hurence.logisland.connect.converter.LogIslandRecordConverter
        kc.data.value.converter.properties: |
            record.serializer=com.hurence.logisland.serializer.KryoSerializer
        kc.data.key.converter.properties: |
            schemas.enable=false
        kc.data.key.converter: org.apache.kafka.connect.storage.StringConverter
        kc.worker.tasks.max: 1
        kc.connector.class: |
            com.datamountaineer.streamreactor.connect.blockchain.source.BlockchainSourceConnector
        kc.connector.offset.backing.store: memory
        kc.connector.properties: |
            connect.blockchain.source.url=wss://ws.blockchain.info/inv
            connect.blockchain.source.kafka.topic=blockchain
```

Sample engine configuration

```
engine:
  component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine
  type: engine
  documentation: Index some apache logs with logisland
  configuration:
    spark.app.name: IndexApacheLogsDemo
    spark.master: yarn-cluster
    spark.driver.memory: 1G
    spark.driver.cores: 1
    spark.executor.memory: 2G
    spark.executor.instances: 4
    spark.executor.cores: 2
    spark.yarn.queue: default
    ...
  streamConfigurations:
```

quick start

Getting started (Hadoop cluster)

Download the latest release from github

```
tar -xzf logisland-1.0.0-RC1-bin.tar.gz
```

Create a job configuration

```
vim conf/index-apache-logs.yml
```

Run the job

```
export SPARK_HOME=/usr/hdp/current/spark-client
bin/logisland.sh --conf conf/index-apache-logs.yml
```

Getting started (lightweight container)

Pull & run the image from Docker Repository

```
docker pull hurence/logisland
docker run  -it --name logisland \
    -p 8080:8080 -p 5601:5601 -p 9200:9200 \
    -h sandbox  hurence/logisland bash
```

Run the job

```
bin/logisland.sh --conf conf/index-apache-logs.yml
```

Play with your data

Next ?

Roadmap

- visual Stream configuration
- Auto-scaling to optimize cluster resources
- Pattern discovery through Deep Learning
- App store, per use-case knowledge bundles (cybersecurity, fraud, ...)
- GPU native integration

Resources

- **source** : https://github.com/Hurence/logisland/releases
- **Docker** : https://hub.docker.com/r/hurence/logisland/tags/
- **Maven** : https://search.maven.org/#search%7Cga%7C1%7Clogisland
- **Documentation** : http://logisland.readthedocs.io/en/latest/concepts.html
- **support** : https://gitter.im/logisland/logisland
- **contact** : thomas.bailet@hurence.com

Questions ?

hurence