logisland Documentation

Release 1.1.1

bailet.thomas

Apr 15, 2019

Contents

1 Contents: **3**

- 1.1 Introduction . . . 3
- 1.2 Core concepts . . . 3
- 1.3 Architecture . . . 4
- 1.4 User Documentation . . . 6
- 1.5 Developer Documentation . . . 133
- 1.6 Plugins . . . 152
- 1.7 Connectors . . . 154
- 1.8 Tutorials . . . 158
- 1.9 API design . . . 281
- 1.10 Logisland REST API . . . 287
- 1.11 What's new in logisland ? . . . 299
- 1.12 Frequently Asked Questions . . . 302

2 Indices and tables **307**

Chat with us on Gitter

Download the latest release build and unzip on an edge node.

CHAPTER 1

Contents:

1.1 Introduction

you can find a quick presentation below :

1.2 Core concepts

The main goal of LogIsland framework is to provide tools to automatically extract valuable knowledge from historical log data. To do so we need two different kind of processing over our technical stack :

1. Grab events from logs
2. Perform Event Pattern Mining (EPM)

What we know about `Log`/`Event` properties :

- they're naturally temporal
- they carry a global type (user request, error, operation, system failure. . .)
- they're semi-structured
- they're produced by software, so we can deduce some templates from them
- some of them are correlated
- some of them are frequent (or rare)
- some of them are monotonic
- some of them are of great interest for system operators

1.2.1 What is a pattern ?

Patterns, actually are a set of items subsequences or substructures that occur frequently together in a data set we call this strongly correlated. Patterns usually represent intrinsic and important properties of data.

1.2.2 From raw to structure

The first part of the process is to extract semantics from semi-structured data such as logs. The main objective of this phase is to introduce a canonical semantics in log data that we will call `Event` which will be easier for us to process with data mining algorithm

- log parser
- log classification/clustering
- event generation
- event summarization

1.2.3 Event pattern mining

Once we have a cannonical semantic in the form of events we can perform time window processing over our events set. All the algorithms we can run on it will help us to find some of the following properties :

- sequential patterns
- events burst
- frequent pattern
- rare event
- highly correlated events
- correlation between time series & events

1.3 Architecture

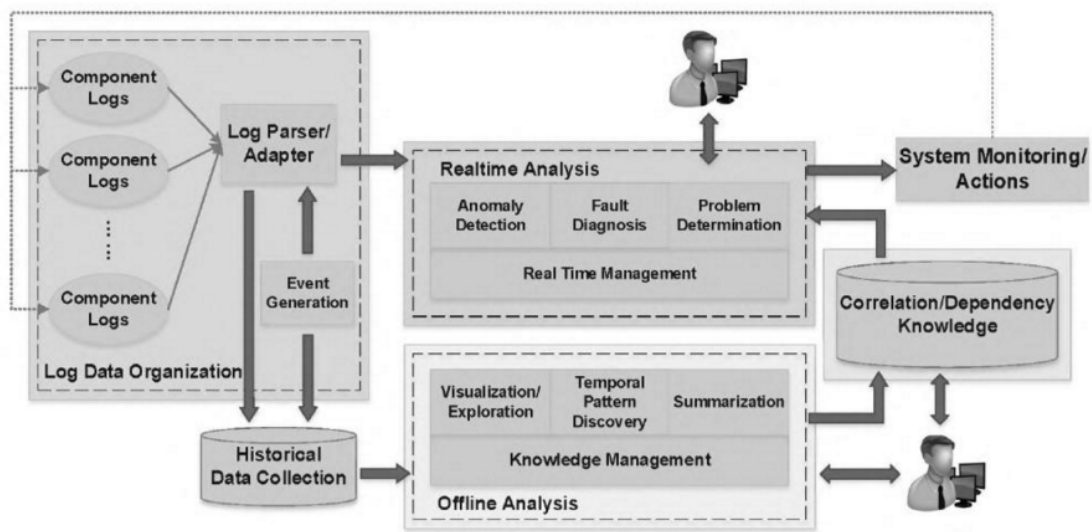
Is there something clever out there ?

Most of the systems in this data world can be observables through their **events**. You just have to look at the event sourcing pattern to get an idea of how we could define any system state as a sequence of temporal events. The main source of events are the **logs** files, application logs, transaction logs, sensor data, etc.

Large and complex systems, made of number of heterogeneous components are not easy to monitor, especially when have to deal with distributed computing. Most of the time of IT resources is spent in maintenance tasks, so there's a real need for tools to help achieving them.

Note: Basicaly LogIsland will help us to handle system events from log files.

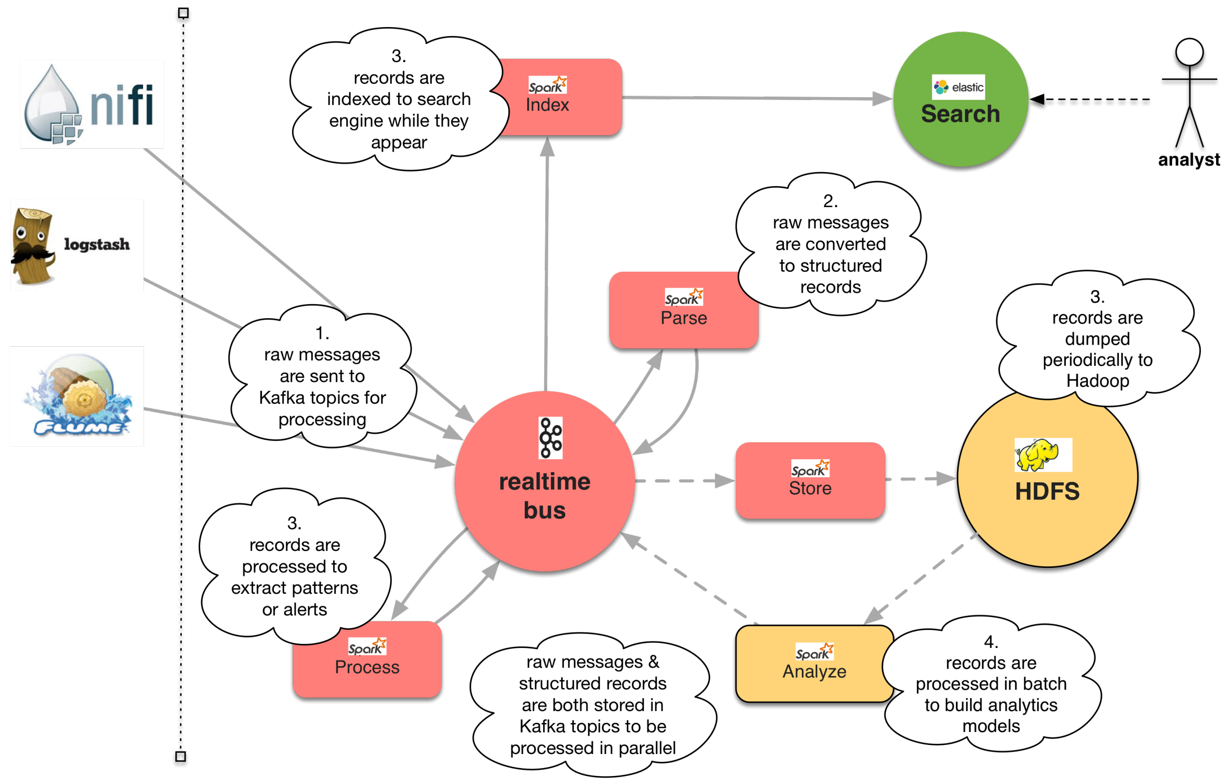
1.3.1 Data driven architecture



1.3.2 Technical design

LogIsland is an event processing framework based on Kafka and Spark. The main goal of this Open Source platform is to abstract the level of complexity of complex event processing at scale. Of course many people start with an ELK stack, which is really great but not enough to elaborate a really complete system monitoring tool. So with LogIsland, you'll move the log processing burden to a powerful distributed stack.

Kafka acts a the distributed message queue middleware while Spark is the core of the distributed processing. LogIsland glue those technologies to simplify log complex event processing at scale.



1.4 User Documentation

Contents:

1.4.1 Components

Contents:

Engines Documentation

Contents:

Engine-spark

ConsoleStructuredStreamProviderService

No description provided.

Class

com.hurence.logisland.stream.spark.structured.provider.ConsoleStructuredStreamProviderService

Tags

None.

Properties

This component has no required or optional properties.

DummyRecordStream

No description provided.

Class

com.hurence.logisland.stream.spark.DummyRecordStream

Tags

None.

Properties

This component has no required or optional properties.

KafkaConnectBaseProviderService

No description provided.

Class

com.hurence.logisland.stream.spark.provider.KafkaConnectBaseProviderService

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 1: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
kc.connector.class	The class canonical name of the kafka connector to use.		null	false	false
kc.connector.properties	The properties (key=value) for the connector.			false	false
kc.data.key.converter	Key converter class		null	false	false
kc.data.key.converter.properties	Key converter properties			false	false
kc.data.value.converter	Value converter class		null	false	false
kc.data.value.converter.properties	Value converter properties			false	false
kc.worker.tasks.max	Max number of threads for this connector		1	false	false
kc.partitions.max	Max number of partitions for this connector.		null	false	false
kc.connector.offset.backing.store	The underlying backing store to be used.	memory (Standalone in memory offset backing store. Not suitable for clustered deployments unless source is unique or stateless), file (Standalone filesystem based offset backing store. You have to specify the property offset.storage.file.filename for the file path.Not suitable for clustered deployments unless source is unique or standalone), kafka (Distributed kafka topic based offset backing store. See the javadoc of class org.apache.kafka.connect.storage.KafkaOffsetBackingStore for the configuration options.This backing store is well suited for distributed deployments.)	memory	false	false
kc.connector.offset.backing.store.properties	Properties to configure the offset backing store			false	false

KafkaConnectStructuredSinkProviderService

No description provided.

Class

com.hurence.logisland.stream.spark.provider.KafkaConnectStructuredSinkProviderService

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 2: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
kc.connector.class	The class canonical name of the kafka connector to use.		null	false	false
kc.connector.properties	The properties (key=value) for the connector.			false	false
kc.data.key.converter	Key converter class		null	false	false
kc.data.key.converter.properties	Key converter properties			false	false
kc.data.value.converter	Value converter class		null	false	false
kc.data.value.converter.properties	Value converter properties			false	false
kc.worker.tasks.max	Max number of threads for this connector		1	false	false
kc.partitions.max	Max number of partitions for this connector.		null	false	false
kc.connector.offset.backing.store	The underlying backing store to be used.	memory (Standalone in memory offset backing store. Not suitable for clustered deployments unless source is unique or stateless), file (Standalone filesystem based offset backing store. You have to specify the property offset.storage.file.filename for the file path.Not suitable for clustered deployments unless source is unique or standalone), kafka (Distributed kafka topic based offset backing store. See the javadoc of class org.apache.kafka.connect.storage.KafkaOffsetBackingStore for the configuration options.This backing store is well suited for distributed deployments.)	memory	false	false
kc.connector.offset.backing.store.properties	Properties to configure the offset backing store			false	false

---

KafkaConnectStructuredSourceProviderService

No description provided.

Class

com.hurence.logisland.stream.spark.provider.KafkaConnectStructuredSourceProviderService

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 3: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
kc.connector.class	The class canonical name of the kafka connector to use.		null	false	false
kc.connector.properties	The properties (key=value) for the connector.			false	false
kc.data.key.converter	Key converter class		null	false	false
kc.data.key.converter.properties	Key converter properties			false	false
kc.data.value.converter	Value converter class		null	false	false
kc.data.value.converter.properties	Value converter properties			false	false
kc.worker.tasks.max	Max number of threads for this connector		1	false	false
kc.partitions.max	Max number of partitions for this connector.		null	false	false
kc.connector.offset.backing.store	The underlying backing store to be used.	memory (Standalone in memory offset backing store. Not suitable for clustered deployments unless source is unique or stateless), file (Standalone filesystem based offset backing store. You have to specify the property offset.storage.file.filename for the file path.Not suitable for clustered deployments unless source is unique or standalone), kafka (Distributed kafka topic based offset backing store. See the javadoc of class org.apache.kafka.connect.storage.KafkaOffsetBackingStore for the configuration options.This backing store is well suited for distributed deployments.)	memory	false	false
kc.connector.offset.backing.store.properties	Properties to configure the offset backing store			false	false

KafkaRecordStreamDebugger

No description provided.

Class

com.hurence.logisland.stream.spark.KafkaRecordStreamDebugger

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 4: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
kafka.error.topics	Sets the error topics Kafka topic name		_errors	false	false
kafka.input.topics	Sets the input Kafka topic name		_raw	false	false
kafka.output.topics	Sets the output Kafka topic name		_records	false	false
avro.input.schema	the avro schema definition		null	false	false
avro.output.schema	the avro schema definition for the output serialization		null	false	false
kafka.input.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes)	com.hurence.logisland.serializer.KryoSerializer	false	false
kafka.output.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes)	com.hurence.logisland.serializer.KryoSerializer	false	false
kafka.error.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as	com.hurence.logisland.serializer.JsonSerializer	false	false

KafkaRecordStreamHDFSBurner

No description provided.

Class

com.hurence.logisland.stream.spark.KafkaRecordStreamHDFSBurner

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 5: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
kafka.error.topics	Sets the error topics Kafka topic name		_errors	false	false
kafka.input.topics	Sets the input Kafka topic name		_raw	false	false
kafka.output.topics	Sets the output Kafka topic name		_records	false	false
avro.input.schema	the avro schema definition		null	false	false
avro.output.schema	the avro schema definition for the output serialization		null	false	false
kafka.input.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes)	com.hurence.logisland.serializer.KryoSerializer	false	false
kafka.output.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes)	com.hurence.logisland.serializer.KryoSerializer	false	false
kafka.error.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as	com.hurence.logisland.serializer.JsonSerializer	false	false

---

KafkaRecordStreamParallelProcessing

No description provided.

Class

com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 6: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
kafka.error.topics	Sets the error topics Kafka topic name		_errors	false	false
kafka.input.topics	Sets the input Kafka topic name		_raw	false	false
kafka.output.topics	Sets the output Kafka topic name		_records	false	false
avro.input.schema	the avro schema definition		null	false	false
avro.output.schema	the avro schema definition for the output serialization		null	false	false
kafka.input.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes)	com.hurence.logisland.serializer.KryoSerializer	false	false
kafka.output.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes)	com.hurence.logisland.serializer.KryoSerializer	false	false
kafka.error.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as	com.hurence.logisland.serializer.JsonSerializer	false	false

---

KafkaRecordStreamSQLAggregator

This is a stream capable of SQL query interpretations.

Class

com.hurence.logisland.stream.spark.KafkaRecordStreamSQLAggregator

Tags

stream, SQL, query, record

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 7: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
kafka.error.topics	Sets the error topics Kafka topic name		_errors	false	false
kafka.input.topics	Sets the input Kafka topic name		_raw	false	false
kafka.output.topics	Sets the output Kafka topic name		_records	false	false
avro.input.schema	the avro schema definition		null	false	false
avro.output.schema	the avro schema definition for the output serialization		null	false	false
kafka.input.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes)	com.hurence.logisland.serializer.KryoSerializer	false	false
kafka.output.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes)	com.hurence.logisland.serializer.KryoSerializer	false	false
kafka.error.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as	com.hurence.logisland.serializer.JsonSerializer	false	false

KafkaStreamProcessingEngine

No description provided.

Class

com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 8: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
spark.app.name	Tha application name		logisland	false	false
spark.master	The url to Spark Master		local[2]	false	false
spark.monitoring.driver.port	The port for exposing monitoring metrics		null	false	false
spark.yarn.deploy-mode	The yarn deploy mode		null	false	false
spark.yarn.queue	The name of the YARN queue		default	false	false
spark.driver.memory	The memory size for Spark driver		512m	false	false
spark.executor.memory	The memory size for Spark executors		1g	false	false
spark.driver.cores	The number of cores for Spark driver		4	false	false
spark.executor.cores	The number of cores for Spark driver		1	false	false
spark.executor.instances	The number of instances for Spark app		null	false	false
spark.serializer	Class to use for serializing objects that will be sent over the network or need to be cached in serialized form		org.apache.spark.serializer.KryoSerializer	false	false
spark.streaming.blockInterval	Interval at which data received by Spark Streaming receivers is chunked into blocks of data before storing them in Spark. Minimum recommended - 50 ms		350	false	false
spark.streaming.kafka.maxRatePerPartition	Maximum rate (number of records per second) at which data will be read from each Kafka partition		5000	false	false
spark.streaming.batchDuration	No Description Provided.		2000	false	false

Continued on next page

Table 8 – continued from previous page

Name	Description	Allowable Values	Default Value	Sensitive	EL
spark.streaming.backpressure.enabled	This enables the Spark Streaming to control the receiving rate based on the current batch scheduling delays and processing times so that the system receives only as fast as the system can process.		false	false	false
spark.streaming.unpersist	Force RDDs generated and persisted by Spark Streaming to be automatically unpersisted from Spark's memory. The raw input data received by Spark Streaming is also automatically cleared. Setting this to false will allow the raw data and persisted RDDs to be accessible outside the streaming application as they will not be cleared automatically. But it comes at the cost of higher memory usage in Spark.		false	false	false
spark.ui.port	No Description Provided.		4050	false	false
spark.streaming.timeout	No Description Provided.		-1	false	false
spark.streaming.kafka.maxRetries	Maximum rate (number of records per second) at which data will be read from each Kafka partition		3	false	false
spark.streaming.ui.retainedBatches	How many batches the Spark Streaming UI and status APIs remember before garbage collecting.		200	false	false
spark.streaming.receiver.writeAheadLog.enable	Enable write ahead logs for receivers. All the input data received through receivers will be saved to write ahead logs that will allow it to be recovered after driver failures.		false	false	false
spark.yarn.maxAppAttempts	Because Spark driver and Application Master share a single JVM, any error in Spark driver stops our long-running job. Fortunately it is possible to configure maximum number of attempts that will be made to re-run the application. It is reasonable to set higher value than default 2 (derived from YARN cluster property yarn.resourcemanager.am.max-attempts). 4 works quite well, higher value may cause unnecessary restarts even if the reason of the failure is permanent.		4	false	false
spark.yarn.am.attemptFailuresValidityInterval	If the application runs for days or weeks without restart or redeployment on highly utilized cluster, 4 attempts could be exhausted in few hours. To avoid this situation, the attempt counter should be reset on every hour of so.		1h	false	false

Continued on next page

Table 8 – continued from previous page

Name	Description	Allowable Values	Default Value	Sensitive	EL
spark.yarn.max.executor.failures	The maximum number of executor failures before the application fails. By default it is max(2 * num executors, 3), well suited for batch jobs but not for long-running jobs. The property comes with corresponding validity interval which also should be set.8 * num_executors		20	false	false
spark.yarn.executor.failuresValidityInterval	If the application runs for days or weeks without restart or redeployment on highly utilized cluster, x attempts could be exhausted in few hours. To avoid this situation, the attempt counter should be reset on every hour of so.		1h	false	false
spark.task.maxFailures	For long-running jobs you could also consider to boost maximum number of task failures before giving up the job. By default tasks will be retried 4 times and then job fails.		8	false	false
spark.memory.fraction	expresses the size of M as a fraction of the (JVM heap space - 300MB) (default 0.75). The rest of the space (25%) is reserved for user data structures, internal metadata in Spark, and safeguarding against OOM errors in the case of sparse and unusually large records.		0.6	false	false
spark.memory.storageFraction	expresses the size of R as a fraction of M (default 0.5). R is the storage space within M where cached blocks immune to being evicted by execution.		0.5	false	false
spark.scheduler.mode	The scheduling mode between jobs submitted to the same SparkContext. Can be set to FAIR to use fair sharing instead of queueing jobs one after another. Useful for multi-user services.	FAIR (fair sharing), FIFO (queueing jobs one after another)	FAIR	false	false
spark.properties.file.path	for using –properties-file option while submitting spark job		null	false	false

KafkaStructuredStreamProviderService

No description provided.

Class

com.hurence.logisland.stream.spark.structured.provider.KafkaStructuredStreamProviderService

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 9: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
kafka.error.topics	Sets the error topics Kafka topic name		_errors	false	false
kafka.input.topics	Sets the input Kafka topic name		_raw	false	false
kafka.output.topics	Sets the output Kafka topic name		_records	false	false
avro.input.schema	the avro schema definition		null	false	false
avro.output.schema	the avro schema definition for the output serialization		null	false	false
kafka.input.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes)	com.hurence.logisland.serializer.KryoSerializer	false	false
kafka.output.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes)	com.hurence.logisland.serializer.KryoSerializer	false	false
kafka.error.topics.serializer	No Description Provided.	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as	com.hurence.logisland.serializer.JsonSerializer	false	false

MQTTStructuredStreamProviderService

No description provided.

Class

com.hurence.logisland.stream.spark.structured.provider.MQTTStructuredStreamProviderService

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 10: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
mqtt.broker.url	brokerUrl A url MqttClient connects to. Set this or path as the url of the Mqtt Server. e.g. tcp://localhost:1883		tcp://localhost:1883	false	false
mqtt.clean.session	cleanSession Setting it true starts a clean session, removes all checkpointed messages by a previous run of this source. This is set to false by default.		true	false	false
mqtt.client.id	clientID this client is associated. Provide the same value to recover a stopped client.		null	false	false
mqtt.connection.timeout	connectionTimeout Sets the connection timeout, a value of 0 is interpreted as wait until client connects. See MqttConnectOptions.setConnectionTimeout for more information		5000	false	false
mqtt.keep.alive	keepAlive Same as MqttConnectOptions.setKeepAliveInterval.		5000	false	false
mqtt.password	password Sets the password to use for the connection		null	false	false
mqtt.persistence	persistence By default it is used for storing incoming messages on disk. If memory is provided as value for this option, then recovery on restart is not supported.		memory	false	false
mqtt.version	mqttVersion Same as MqttConnectOptions.setMqttVersion		5000	false	false
mqtt.username	username Sets the user name to use for the connection to Mqtt Server. Do not set it, if server does not need this. Setting it empty will lead to errors.		null	false	false
mqtt.qos	QoS The maximum quality of service to subscribe each topic at.Messages published at a lower quality of service will be received at the published QoS.Messages published at a higher quality of service will be received using the QoS specified on the subscribe		0	false	false
mqtt.topic	Topic MqttClient subscribes to.		null	false	false

RemoteApiStreamProcessingEngine

No description provided.

Class

com.hurence.logisland.engine.spark.RemoteApiStreamProcessingEngine

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 11: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
spark.app.name	Tha application name		logisland	false	false
spark.master	The url to Spark Master		local[2]	false	false
spark.monitoring.driver.port	The port for exposing monitoring metrics		null	false	false
spark.yarn.deploy-mode	The yarn deploy mode		null	false	false
spark.yarn.queue	The name of the YARN queue		default	false	false
spark.driver.memory	The memory size for Spark driver		512m	false	false
spark.executor.memory	The memory size for Spark executors		1g	false	false
spark.driver.cores	The number of cores for Spark driver		4	false	false
spark.executor.cores	The number of cores for Spark driver		1	false	false
spark.executor.instances	The number of instances for Spark app		null	false	false
spark.serializer	Class to use for serializing objects that will be sent over the network or need to be cached in serialized form		org.apache.spark.serializer.KryoSerializer	false	false
spark.streaming.blockInterval	Interval at which data received by Spark Streaming receivers is chunked into blocks of data before storing them in Spark. Minimum recommended - 50 ms		350	false	false
spark.streaming.kafka.maxRatePerPartition	Maximum rate (number of records per second) at which data will be read from each Kafka partition		5000	false	false
spark.streaming.batchDuration	No Description Provided.		2000	false	false
spark.streaming.backpressure.enabled	This enables the Spark Streaming to control the receiving rate based on the current batch scheduling delays and processing times so that the system receives only as fast as the system can process.		false	false	false

Continued on next page

Table 11 – continued from previous page

Name	Description	Allowable Values	Default Value	Sensitive	EL
spark.streaming.unpersist	Force RDDs generated and persisted by Spark Streaming to be automatically unpersisted from Spark's memory. The raw input data received by Spark Streaming is also automatically cleared. Setting this to false will allow the raw data and persisted RDDs to be accessible outside the streaming application as they will not be cleared automatically. But it comes at the cost of higher memory usage in Spark.		false	false	false
spark.ui.port	No Description Provided.		4050	false	false
spark.streaming.timeout	No Description Provided.		-1	false	false
spark.streaming.kafka.maxRetries	Maximum rate (number of records per second) at which data will be read from each Kafka partition		3	false	false
spark.streaming.ui.retainedBatches	How many batches the Spark Streaming UI and status APIs remember before garbage collecting.		200	false	false
spark.streaming.receiver.writeAheadLog.enable	Enable write ahead logs for receivers. All the input data received through receivers will be saved to write ahead logs that will allow it to be recovered after driver failures.		false	false	false
spark.yarn.maxAppAttempts	Because Spark driver and Application Master share a single JVM, any error in Spark driver stops our long-running job. Fortunately it is possible to configure maximum number of attempts that will be made to re-run the application. It is reasonable to set higher value than default 2 (derived from YARN cluster property yarn.resourcemanager.am.max-attempts). 4 works quite well, higher value may cause unnecessary restarts even if the reason of the failure is permanent.		4	false	false
spark.yarn.am.attemptFailuresValidityInterval	If the application runs for days or weeks without restart or redeployment on highly utilized cluster, 4 attempts could be exhausted in few hours. To avoid this situation, the attempt counter should be reset on every hour of so.		1h	false	false
spark.yarn.max.executor.failures	The maximum number of executor failures before the application fails. By default it is max(2 * num executors, 3), well suited for batch jobs but not for long-running jobs. The property comes with corresponding validity interval which also should be set.8 * num_executors		20	false	false

Continued on next page

Table 11 – continued from previous page

Name	Description	Allowable Values	Default Value	Sensitive	EL
spark.yarn.executor.failuresValidityInterval	If the application runs for days or weeks without restart or redeployment on highly utilized cluster, x attempts could be exhausted in few hours. To avoid this situation, the attempt counter should be reset on every hour of so.		1h	false	false
spark.task.maxFailures	For long-running jobs you could also consider to boost maximum number of task failures before giving up the job. By default tasks will be retried 4 times and then job fails.		8	false	false
spark.memory.fraction	expresses the size of M as a fraction of the (JVM heap space - 300MB) (default 0.75). The rest of the space (25%) is reserved for user data structures, internal metadata in Spark, and safeguarding against OOM errors in the case of sparse and unusually large records.		0.6	false	false
spark.memory.storageFraction	expresses the size of R as a fraction of M (default 0.5). R is the storage space within M where cached blocks immune to being evicted by execution.		0.5	false	false
spark.scheduler.mode	The scheduling mode between jobs submitted to the same SparkContext. Can be set to FAIR to use fair sharing instead of queueing jobs one after another. Useful for multi-user services.	FAIR (fair sharing), FIFO (queueing jobs one after another)	FAIR	false	false
spark.properties.file.path	for using –properties-file option while submitting spark job		null	false	false
remote.api.baseUrl	The base URL of the remote server providing logisland configuration		null	false	false
remote.api.polling.rate	Remote api polling rate in milliseconds		null	false	false
remote.api.push.rate	Remote api configuration push rate in milliseconds		null	false	false
remote.api.timeouts.connect	Remote api connection timeout in milliseconds		10000	false	false
remote.api.auth.user	The basic authentication user for the remote api endpoint.		null	false	false
remote.api.auth.password	The basic authentication password for the remote api endpoint.		null	false	false
remote.api.timeouts.socket	Remote api default read/write socket timeout in milliseconds		10000	false	false

StructuredStream

No description provided.

Class

com.hurence.logisland.stream.spark.structured.StructuredStream

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 12: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
read.topics	the input path for any topic to be read from		null	false	false
read.topics.client.service	the controller service that gives connection information		null	false	false
read.topics.serializer	the serializer to use	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes), com.hurence.logisland.serializer.KuraProtobufSerializer (serialize events as Kura protocol buffer)	none	false	false
read.topics.key.serializer	The key serializer to use	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.KuraProtobufSerializer (serialize events as Kura protocol buffer), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes)	none	false	false
write.topics	the input path for any topic to be written to		null	false	false
write.topics.client.service	the controller service that gives connection information		null	false	false

Engine-vanilla

Find below the list.

AmqpClientPipelineStream

No description provided.

Class

com.hurence.logisland.engine.vanilla.stream.amqp.AmqpClientPipelineStream

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 13: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
connection.host	Connection host name		null	false	false
connection.port	Connection port		5672	false	false
link.credits	Flow control. How many credits for this links. Higher means higher prefetch (prebuffered number of messages		1024	false	false
connection.auth.user	Connection authenticated user name		null	false	false
connection.auth.password	Connection authenticated password		null	false	false
connection.auth.tls-cert	Connection TLS public certificate (PEM file path)		null	false	false
connection.auth.tls-key	Connection TLS private key (PEM file path)		null	false	false
connection.auth.ca-cert	Connection TLS CA cert (PEM file path)		null	false	false
read.topic	The input path for any topic to be read from			false	false
read.topic.serializer	The serializer to use	com.hurence.logisland.serializer.BsonSerializer (serialize events as bson), com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes), com.hurence.logisland.serializer.KuraProtobufSerializer (serialize events as Kura protocol buffer)	none	false	false
avro.input.schema	The avro schema definition		null	false	false
write.topic	The input path for any topic to be written to			false	false
write.topic.serializer	The serializer to use	com.hurence.logisland.serializer.BsonSerializer (serialize events as bson), com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays),	none	false	false

Extra informations

No additional information is provided

KafkaStreamsPipelineStream

No description provided.

Class

com.hurence.logisland.engine.vanilla.stream.kafka.KafkaStreamsPipelineStream

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 14: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
bootstrap.servers	List of kafka nodes to connect to		null	false	false
read.topics	The input path for any topic to be read from			false	false
avro.input.schema	The avro schema definition		null	false	false
avro.output.schema	The avro schema definition for the output serialization		null	false	false
kafka.manual.offset.reset	What to do when there is no initial offset in Kafka or if the current offset does not exist any more on the server (e.g. because that data has been deleted): earliest: automatically reset the offset to the earliest offset latest: automatically reset the offset to the latest offset none: throw exception to the consumer if no previous offset is found for the consumer's group anything else: throw exception to the consumer.	latest (the offset to the latest offset), earliest (the offset to the earliest offset), none (the latest saved offset)	earliest	false	false
read.topics.serializer	The serializer to use	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), com.hurence.logisland.serializer.BytesArraySerializer (serialize events as byte arrays), com.hurence.logisland.serializer.StringSerializer (serialize events as string), none (send events as bytes), com.hurence.logisland.serializer.KuraProtobufSerializer (serialize events as Kura protocol buffer)	none	false	false
write.topics	The input path for any topic to be written to			false	false
write.topics.serializer	The serializer to use	com.hurence.logisland.serializer.KryoSerializer (serialize events as binary blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.ExtendedJsonSerializer (serialize events as json blocs supporting nested objects/arrays), com.hurence.logisland.serializer.AvroSerializer	none	false	false

Extra informations

No additional information is provided

PlainJavaEngine

No description provided.

Class

com.hurence.logisland.engine.vanilla.PlainJavaEngine

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 15: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
jvm.heap.min	Minimum memory the JVM should allocate for its heap		null	false	false
jvm.heap.max	Maximum memory the JVM should allocate for its heap		null	false	false

Extra informations

No additional information is provided

Common-processors

Find below the list.

AddFields

Add one or more field to records

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.AddFields

Tags

record, fields, Add

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 16: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
conflict.resolution.policy	What to do when a field with the same name already exists ?	overwrite_existing (if field already exist), keep_only_old_field (keep only old field)	keep_only_old_field	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 17: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
Name of the field to add	Value of the field to add	Add a field to the record with the specified value. Expression language can be used.You can not add a field that end with '.type' as this suffix is used to specify the type of fields to add		null	**true**
Name of the field to add with the suffix '.field.type'	Type of the field to add	Add a field to the record with the specified type. These properties are only used if a correspondant property without the suffix '.field.type' is already defined. If this property is not defined, default type for adding fields is String.You can only use Logisland predefined type fields.	NULL, STRING, INT, LONG, ARRAY, FLOAT, DOUBLE, BYTES, RECORD, MAP, ENUM, BOOLEAN, UNION, DATETIME	STRING	false
Name of the field to add with the suffix '.field.name'	Name of the field to add using expression language	Add a field to the record with the specified name (which is evaluated using expression language). These properties are only used if a correspondant property without the suffix '.field.name' is already defined. If this property is not defined, the name of the field to add is the key of the first dynamic property (which is the main and only required dynamic property).		null	**true**

Extra informations

Add one or more field with constant value or dynamic value using the expression-language.Some examples of settings:

```
newStringField: bonjour
newIntField: 14
newIntField.field.type: INT
```

Would add those fields in record :

```
Field{name='newStringField', type='STRING', value='bonjour'}
Field{name='newIntField', type='INT', value=14}
```

Here a second example using expression language, once for the value, once for the key. Note that you can use for both.We suppose that our record got already those fields :

```
Field{name='field1', type='STRING', value='bonjour'}
Field{name='field2', type='INT', value=14}
```

This settings : .. code:

```
newStringField: ${field1 + "-" + field2}
fieldToCalulateKey: 555
fieldToCalulateKey.field.name: ${"_" + field1 + "-"}
```

Would add those fields in record :

```
Field{name='newStringField', type='STRING', value='bonjour-14'}
Field{name='_bonjour-', type='STRING', value='555'}
```

As you probably notice, you can not add fields with name ending by either '.field.name' either '.field.type' because they are suffix are used to sort dynamic properties. But if you really want to do this a workaround is to specify the name of the field oui expression language, for example this settings would work:

```
fieldWithReservedSuffix: bonjour
fieldWithReservedSuffix.field.type: INT
fieldWithReservedSuffix.field.type: myfield.endind.with.reserved.suffix.field.type
```

ApplyRegexp

This processor is used to create a new set of fields from one field (using regexp).

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.ApplyRegexp

Tags

parser, regex, log, record

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 18: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
conflict.resolution.policy	What to do when a field with the same name already exists ?	overwrite_existing (if field already exist), keep_only_old_field (keep only old field)	keep_only_old_field	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 19: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
alternative regex & mapping	another regex that could match	This processor is used to create a new set of fields from one field (using regexp).		null	**true**

Extra informations

This processor is used to create a new set of fields from one field (using regexp).

See Also:

com.hurence.logisland.processor.ApplyRegexp

BulkPut

Indexes the content of a Record in a Datastore using bulk processor

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.datastore.BulkPut

Tags

datastore, record, put, bulk

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values, and whether a property supports the Expression Language .

Table 20: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
datastore.client.service	The instance of the Controller Service to use for accessing datastore.		null	false	false
default.collection	The name of the collection/index/table to insert into		null	false	**true**
timebased.collection	do we add a date suffix	no (no date added to default index), today (today's date added to default index), yesterday (yesterday's date added to default index)	no	false	false
date.format	simple date format for date suffix. default : yyyy.MM.dd		yyyy.MM.dd	false	false
collection.field	the name of the event field containing es index name => will override index value if set		null	false	**true**

Extra informations

Indexes the content of a Record in a Datastore using bulk processor.

CheckAlerts

Add one or more records representing alerts. Using a datastore.

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.alerting.CheckAlerts

Tags

record, alerting, thresholds, opc, tag

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 21: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
max.cpu.time	maximum CPU time in milliseconds allowed for script execution.		100	false	false
max.memory	maximum memory in Bytes which JS executor thread can allocate		51200	false	false
allow.no.brace	Force, to check if all blocks are enclosed with curly braces “”{}”“.		false	false	false
max.prepared.statements	The size of prepared statements LRU cache. If 0, this is disabled.		30	false	false
datastore.client.service	The instance of the Controller Service to use for accessing datastore.		null	false	false
datastore.cache.collection	The collection where to find cached objects		test	false	false
js.cache.service	The cache service to be used to store already sanitized JS expressions. If not specified a in-memory unlimited hash map will be used.		null	false	false
output.record.type	the type of the output record		event	false	false
profile.activation.condition	A javascript expression that activates this alerting profile when true		0==0	false	false
alert.criticity	from 0 to …		0	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 22: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
field to add	a default value	Add a field to the record with the default value		null	false

Extra informations

Add one or more records representing alerts. Using a datastore.

CheckThresholds

Compute threshold cross from given formulas.

- each dynamic property will return a new record according to the formula definition
- the record name will be set to the property name
- the record time will be set to the current timestamp

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.alerting.CheckThresholds

Tags

record, threshold, tag, alerting

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 23: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
max.cpu.time	maximum CPU time in milliseconds allowed for script execution.		100	false	false
max.memory	maximum memory in Bytes which JS executor thread can allocate		51200	false	false
allow.no.brace	Force, to check if all blocks are enclosed with curly braces “”{}”“.		false	false	false
max.prepared.statements	The size of prepared statements LRU cache. If 0, this is disabled.		30	false	false
datastore.client.service	The instance of the Controller Service to use for accessing datastore.		null	false	false
datastore.cache.collection	The collection where to find cached objects		test	false	false
js.cache.service	The cache service to be used to store already sanitized JS expressions. If not specified a in-memory unlimited hash map will be used.		null	false	false
output.record.type	the type of the output record		event	false	false
record.ttl	How long (in ms) do the record will remain in cache		30000	false	false
min.update.time.ms	The minimum amount of time (in ms) that we expect between two consecutive update of the same threshold record		200	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 24: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
field to add	a default value	Add a field to the record with the default value		null	false

Extra informations

Compute threshold cross from given formulas.

- each dynamic property will return a new record according to the formula definition
- the record name will be set to the property name
- the record time will be set to the current timestamp

ComputeTags

Compute tag cross from given formulas.

- each dynamic property will return a new record according to the formula definition
- the record name will be set to the property name
- the record time will be set to the current timestamp

a threshold_cross has the following properties : count, sum, avg, time, duration, value

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.alerting.ComputeTags

Tags

record, fields, Add

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 25: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
max.cpu.time	maximum CPU time in milliseconds allowed for script execution.		100	false	false
max.memory	maximum memory in Bytes which JS executor thread can allocate		51200	false	false
allow.no.brace	Force, to check if all blocks are enclosed with curly braces “”{}”“.		false	false	false
max.prepared.statements	The size of prepared statements LRU cache. If 0, this is disabled.		30	false	false
datastore.client.service	The instance of the Controller Service to use for accessing datastore.		null	false	false
datastore.cache.collection	The collection where to find cached objects		test	false	false
js.cache.service	The cache service to be used to store already sanitized JS expressions. If not specified a in-memory unlimited hash map will be used.		null	false	false
output.record.type	the type of the output record		event	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 26: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
field to add	a default value	Add a field to the record with the default value		null	false

Extra informations

Compute tag cross from given formulas.

- each dynamic property will return a new record according to the formula definition
- the record name will be set to the property name
- the record time will be set to the current timestamp

a threshold_cross has the following properties : count, sum, avg, time, duration, value

ConvertFieldsType

Converts a field value into the given type. does nothing if conversion is not possible

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.ConvertFieldsType

Tags

type, fields, update, convert

Properties

This component has no required or optional properties.

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 27: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
field	the new type	convert field value into new type		null	**true**

Extra informations

Converts a field value into the given type. does nothing if conversion is not possible.

ConvertSimpleDateFormatFields

Convert one or more field representing a date into a Unix Epoch Time (time in milliseconds since &st January 1970, 00:00:00 GMT)…

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.ConvertSimpleDateFormatFields

Tags

record, fields, Add

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 28: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
conflict.resolution.policy	What to do when a field with the same name already exists ?	overwrite_existing (if field already exist), keep_only_old_field (keep only old field)	keep_only_old_field	false	false
input.date.format	Simple date format representation of the input field to convert		null	false	false
timezone	Specify the timezone (default is CET)		CET	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 29: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
field name to add	value to convert into Epoch timestamp using given input.date.format	Add a field to the record with the name, converting value using java SimpleDateFormat		null	**true**

Extra informations

Convert one or more field representing a date into a Unix Epoch Time (time in milliseconds since &st January 1970, 00:00:00 GMT)…

DebugStream

This is a processor that logs incoming records

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.DebugStream

Tags

record, debug

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 30: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
event.serializer	the way to serialize event	json (serialize events as json blocs), string (serialize events as toString() blocs)	json	false	false

Extra informations

This is a processor that logs incoming records.

EnrichRecords

Enrich input records with content indexed in datastore using multiget queries. Each incoming record must be possibly enriched with information stored in datastore. The plugin properties are :

- es.index (String) : Name of the datastore index on which the multiget query will be performed. This field is mandatory and should not be empty, otherwise an error output record is sent for this specific incoming record.
- record.key (String) : Name of the field in the input record containing the id to lookup document in elastic search. This field is mandatory.
- es.key (String) : Name of the datastore key on which the multiget query will be performed. This field is mandatory.

- includes (ArrayList<String>) : List of patterns to filter in (include) fields to retrieve. Supports wildcards. This field is not mandatory.
- excludes (ArrayList<String>) : List of patterns to filter out (exclude) fields to retrieve. Supports wildcards. This field is not mandatory.

Each outcoming record holds at least the input record plus potentially one or more fields coming from of one datastore document.

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.datastore.EnrichRecords

Tags

datastore, enricher

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values, and whether a property supports the Expression Language .

Table 31: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
datastore.client.service	The instance of the Controller Service to use for accessing datastore.		null	false	false
record.key	The name of field in the input record containing the document id to use in ES multi-get query		null	false	**true**
includes.field	The name of the ES fields to include in the record.		•	false	**true**
excludes.field	The name of the ES fields to exclude.		N/A	false	false
type.name	The typle of record to look for		null	false	**true**
collection.name	The name of the collection to look for		null	false	**true**

Extra informations

Enrich input records with content indexed in datastore using multiget queries. Each incoming record must be possibly enriched with information stored in datastore. The plugin properties are :

- es.index (String) : Name of the datastore index on which the multiget query will be performed. This field is mandatory and should not be empty, otherwise an error output record is sent for this specific incoming record.
- record.key (String) : Name of the field in the input record containing the id to lookup document in elastic search. This field is mandatory.

- es.key (String) : Name of the datastore key on which the multiget query will be performed. This field is mandatory.
- includes (ArrayList<String>) : List of patterns to filter in (include) fields to retrieve. Supports wildcards. This field is not mandatory.
- excludes (ArrayList<String>) : List of patterns to filter out (exclude) fields to retrieve. Supports wildcards. This field is not mandatory.

Each outcoming record holds at least the input record plus potentially one or more fields coming from of one datastore document.

EvaluateJsonPath

Evaluates one or more JsonPath expressions against the content of a FlowFile. The results of those expressions are assigned to Records Fields depending on configuration of the Processor. JsonPaths are entered by adding user-defined properties; the name of the property maps to the Field Name into which the result will be placed. The value of the property must be a valid JsonPath expression. A Return Type of 'auto-detect' will make a determination based off the configured destination. If the JsonPath evaluates to a JSON array or JSON object and the Return Type is set to 'scalar' the Record will be routed to error. A Return Type of JSON can return scalar values if the provided JsonPath evaluates to the specified value. If the expression matches nothing, Fields will be created with empty strings as the value

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.EvaluateJsonPath

Tags

JSON, evaluate, JsonPath

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 32: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
return.type	Indicates the desired return type of the JSON Path expressions. Selecting 'auto-detect' will set the return type to 'json' or 'scalar'	json, scalar	scalar	false	false
path.not.found.behavior	Indicates how to handle missing JSON path expressions. Selecting 'warn' will generate a warning when a JSON path expression is not found.	warn, ignore	ignore	false	false
Null Value Representation	Indicates the desired representation of JSON Path expressions resulting in a null value.	empty string, the string 'null'	empty string	false	false
json.input.field.name	the name of the field containing the json string		record_value	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 33: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
A Record field	A JsonPath expression	will be set to any JSON objects that match the JsonPath.		null	false

Extra informations

Evaluates one or more JsonPath expressions against the content of a FlowFile. The results of those expressions are assigned to Records Fields depending on configuration of the Processor. JsonPaths are entered by adding user-defined properties; the name of the property maps to the Field Name into which the result will be placed. The value of the property must be a valid JsonPath expression. A Return Type of 'auto-detect' will make a determination based off the configured destination. If the JsonPath evaluates to a JSON array or JSON object and the Return Type is set to 'scalar' the Record will be routed to error. A Return Type of JSON can return scalar values if the provided JsonPath evaluates to the specified value. If the expression matches nothing, Fields will be created with empty strings as the value.

ExpandMapFields

Expands the content of a MAP field to the root.

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.ExpandMapFields

Tags

record, fields, Expand, Map

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 34: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
fields.to.expand	Comma separated list of fields of type map that will be expanded to the root		null	false	false
conflict.resolution.policy	What to do when a field with the same name already exists ?	overwrite_existing (if field already exist), keep_only_old_field (keep only old field)	keep_only_old_field	false	false

Extra informations

Expands the content of a MAP field to the root.

FilterRecords

Keep only records based on a given field value

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.FilterRecords

Tags

record, fields, remove, delete

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 35: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
field.name	the field name		record_id	false	false
field.value	the field value to keep		null	false	false

Extra informations

Keep only records based on a given field value.

FlatMap

Converts each field records into a single flatten record...

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.FlatMap

Tags

record, fields, flatmap, flatten

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 36: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
keep.root.record	do we add the original record in		true	false	false
copy.root.record.fields	do we copy the original record fields into the flattened records		true	false	false
leaf.record.type	the new type for the flattened records if present			false	false
concat.fields	comma separated list of fields to apply concatenation ex : $rootField/$leaffield		null	false	false
concat.separator	returns $rootField/$leaf/field		/	false	false
include.position	do we add the original record position in		true	false	false

Extra informations

Converts each field records into a single flatten record…

GenerateRandomRecord

This is a processor that make random records given an Avro schema

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.GenerateRandomRecord

Tags

record, avro, generator

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 37: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
avro.output.schema	the avro schema definition for the output serialization		null	false	false
min.events.count	the minimum number of generated events each run		10	false	false
max.events.count	the maximum number of generated events each run		200	false	false

Extra informations

This is a processor that make random records given an Avro schema.

ModifyId

modify id of records or generate it following defined rules

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.ModifyId

Tags

record, id, idempotent, generate, modify

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 38: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
id.generation.strategy	the strategy to generate new Id	randomUuid (generate a randomUid using java library), hashFields (generate a hash from fields), fromFields (generate a string from java pattern and fields), typetimehash (generate a concatenation of type, time and a hash from fields (as for generate_hash strategy))	randomUuid	false	false
fields.to.hash	the comma separated list of field names (e.g. : 'policyid,date_raw'		record_value	false	false
hash.charset	the charset to use to hash id string (e.g. 'UTF-8')		UTF-8	false	false
hash.algorithm	the algorithme to use to hash id string (e.g. 'SHA-256'	SHA-384, SHA-224, SHA-256, MD2, SHA, SHA-512, MD5	SHA-256	false	false
java.formatter.string	the format to use to build id string (e.g. '%4$2s %3$2s %2$2s %1$2s' (see java Formatter)		null	false	false
language.tag	the language to use to format numbers in string	aa, ab, ae, af, ak, am, an, ar, as, av, ay, az, ba, be, bg, bh, bi, bm, bn, bo, br, bs, ca, ce, ch, co, cr, cs, cu, cv, cy, da, de, dv, dz, ee, el, en, eo, es, et, eu, fa, ff, fi, fj, fo, fr, fy, ga, gd, gl, gn, gu, gv, ha, he, hi, ho, hr, ht, hu, hy, hz, ia, id, ie, ig, ii, ik, in, io, is, it, iu, iw, ja, ji, jv, ka, kg, ki, kj, kk, kl, km, kn, ko, kr, ks, ku, kv, kw, ky, la, lb, lg, li, ln, lo, lt, lu, lv, mg, mh, mi, mk, ml, mn, mo, mr, ms, mt, my, na, nb, nd, ne, ng, nl, nn, no, nr, nv, ny, oc, oj, om, or, os, pa, pi, pl, ps, pt, qu, rm, rn, ro, ru, rw, sa, sc, sd, se, sg, si, sk, sl, sm, sn, so, sq, sr, ss, st, su, sv, sw, ta, te, tg, th, ti, tk, tl, tn, to, tr, ts, tt, tw, ty, ug, uk, ur, uz, ve, vi,	en	false	false

Extra informations

modify id of records or generate it following defined rules.

MultiGet

Retrieves a content from datastore using datastore multiget queries. Each incoming record contains information regarding the datastore multiget query that will be performed. This information is stored in record fields whose names are configured in the plugin properties (see below) :

- collection (String) : name of the datastore collection on which the multiget query will be performed. This field is mandatory and should not be empty, otherwise an error output record is sent for this specific incoming record.
- type (String) : name of the datastore type on which the multiget query will be performed. This field is not mandatory.
- ids (String) : comma separated list of document ids to fetch. This field is mandatory and should not be empty, otherwise an error output record is sent for this specific incoming record.
- includes (String) : comma separated list of patterns to filter in (include) fields to retrieve. Supports wildcards. This field is not mandatory.
- excludes (String) : comma separated list of patterns to filter out (exclude) fields to retrieve. Supports wildcards. This field is not mandatory.

Each outcoming record holds data of one datastore retrieved document. This data is stored in these fields :

- collection (same field name as the incoming record) : name of the datastore collection.
- type (same field name as the incoming record) : name of the datastore type.
- id (same field name as the incoming record) : retrieved document id.
- a list of String fields containing :
- field name : the retrieved field name
- field value : the retrieved field value

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.datastore.MultiGet

Tags

datastore, get, multiget

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 39: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
datastore.client.service	The instance of the Controller Service to use for accessing datastore.		null	false	false
collection.field	the name of the incoming records field containing es collection name to use in multiget query.		null	false	false
type.field	the name of the incoming records field containing es type name to use in multiget query		null	false	false
ids.field	the name of the incoming records field containing es document Ids to use in multiget query		null	false	false
includes.field	the name of the incoming records field containing es includes to use in multiget query		null	false	false
excludes.field	the name of the incoming records field containing es excludes to use in multiget query		null	false	false

Extra informations

Retrieves a content from datastore using datastore multiget queries. Each incoming record contains information regarding the datastore multiget query that will be performed. This information is stored in record fields whose names are configured in the plugin properties (see below) :

- collection (String) : name of the datastore collection on which the multiget query will be performed. This field is mandatory and should not be empty, otherwise an error output record is sent for this specific incoming record.
- type (String) : name of the datastore type on which the multiget query will be performed. This field is not mandatory.
- ids (String) : comma separated list of document ids to fetch. This field is mandatory and should not be empty, otherwise an error output record is sent for this specific incoming record.
- includes (String) : comma separated list of patterns to filter in (include) fields to retrieve. Supports wildcards. This field is not mandatory.
- excludes (String) : comma separated list of patterns to filter out (exclude) fields to retrieve. Supports wildcards. This field is not mandatory.

Each outcoming record holds data of one datastore retrieved document. This data is stored in these fields :

- collection (same field name as the incoming record) : name of the datastore collection.
- type (same field name as the incoming record) : name of the datastore type.
- id (same field name as the incoming record) : retrieved document id.
- a list of String fields containing :
- field name : the retrieved field name
- field value : the retrieved field value

NormalizeFields

Changes the name of a field according to a provided name mapping…

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.NormalizeFields

Tags

record, fields, normalizer

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 40: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
conflict.resolution.policy	what to do when a field with the same name already exists ?	do_nothing (leave record as it was), overwrite_existing (if field already exist), keep_only_old_field (keep only old field and delete the other), keep_both_fields (creates an alias for the new field)	do_nothing	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 41: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
alternative mapping	a comma separated list of possible field name	when a field has a name contained in the list it will be renamed with this property field name		null	**true**

Extra informations

Changes the name of a field according to a provided name mapping. . .

ParseProperties

Parse a field made of key=value fields separated by spaces a string like "a=1 b=2 c=3" will add a,b & c fields, respectively with values 1,2 & 3 to the current Record

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.ParseProperties

Tags

record, properties, parser

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 42: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
properties.field	the field containing the properties to split and treat		null	false	false

Extra informations

Parse a field made of key=value fields separated by spaces a string like "a=1 b=2 c=3" will add a,b & c fields, respectively with values 1,2 & 3 to the current Record

---

RemoveFields

Removes a list of fields defined by a comma separated list of field names or keeps only fields defined by a comma separated list of field names.

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.RemoveFields

Tags

record, fields, remove, delete, keep

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 43: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
fields.to.remove	A comma separated list of field names to remove (e.g. 'policyid,date_raw'). Usage of this property is mutually exclusive with the fields.to.keep property. In any case the technical logisland fields record_id, record_time and record_type are not removed even if specified in the list to remove.		null	false	false
fields.to.keep	A comma separated list of field names to keep (e.g. 'policyid,date_raw'. All other fields will be removed. Usage of this property is mutually exclusive with the PropertyDescriptor[fields.to.remove] property. In any case the technical logisland fields record_id, record_time and record_type are not removed even if not specified in the list to keep.		null	false	false

Extra informations

Removes a list of fields defined by a comma separated list of field names or keeps only fields defined by a comma separated list of field names.

SelectDistinctRecords

Keep only distinct records based on a given field

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.SelectDistinctRecords

Tags

record, fields, remove, delete

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 44: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
field.name	the field to distinct records		record_id	false	false

Extra informations

Keep only distinct records based on a given field.

SendMail

The SendMail processor is aimed at sending an email (like for instance an alert email) from an incoming record. There are three ways an incoming record can generate an email according to the special fields it must embed. Here is a list of the record fields that generate a mail and how they work:

- **mail_text**: this is the simplest way for generating a mail. If present, this field means to use its content (value) as the payload of the mail to send. The mail is sent in text format if there is only this special field in the record. Otherwise, used with either mail_html or mail_use_template, the content of mail_text is the aletrnative text to the HTML mail that is generated.
- **mail_html**: this field specifies that the mail should be sent as HTML and the value of the field is mail payload. If mail_text is also present, its value is used as the alternative text for the mail. mail_html cannot be used with mail_use_template: only one of those two fields should be present in the record.
- **mail_use_template**: If present, this field specifies that the mail should be sent as HTML and the HTML content is to be generated from the template in the processor configuration key **html.template**. The template can contain parameters which must also be present in the record as fields. See documentation of html.template for further explanations. mail_use_template cannot be used with mail_html: only one of those two fields should be present in the record.

  If **allow_overwrite** configuration key is true, any mail.* (dot format) configuration key may be overwritten with a matching field in the record of the form mail_* (underscore format). For instance if allow_overwrite is true and mail.to is set to config_address@domain.com, a record generating a mail with a mail_to field set to record_address@domain.com will send a mail to record_address@domain.com.

  Apart from error records (when he is unable to process the incoming record or to send the mail), this processor is not expected to produce any output records.

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.SendMail

Tags

smtp, email, e-mail, mail, mailer, sendmail, message, alert, html

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 45: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
debug	Enable debug. If enabled, debug information are written to stdout.		false	false	false
smtp.server	FQDN, hostname or IP address of the SMTP server to use.		null	false	false
smtp.port	TCP port number of the SMTP server to use.		25	false	false
smtp.security.username	SMTP username.		null	false	false
smtp.security.password	SMTP password.		null	false	false
smtp.security.ssl	Use SSL under SMTP or not (SMTPS). Default is false.		false	false	false
mail.from.address	Valid mail sender email address.		null	false	false
mail.from.name	Mail sender name.		null	false	false
mail.bounce.address	Mail bounce email address (where error mail is sent if the mail is refused by the recipient server).		null	false	false
mail.replyto.address	Reply to email address.		null	false	false
mail.subject	Mail subject.		[LOGISLAND] Automatic email	false	false
mail.to	Comma separated list of email recipients. If not set, the record must have a mail_to field and allow_overwrite configuration key should be true.		null	false	false
allow_overwrite	If true, allows to overwrite processor configuration with special record fields (mail_to, mail_from_address, mail_from_name, mail_bounce_address, mail_replyto_address, mail_subject). If false, special record fields are ignored and only processor configuration keys are used.		true	false	false
html.template	HTML template to use. It is used when the incoming record contains a mail_use_template field. The template may contain some parameters. The parameter format in the template is of the form ${xxx}. For instance ${param_user} in the template means that a field named param_user must be present in the record and its value will replace the ${param_user} string in the HTML template when the mail will be sent. If some parameters are declared in the template, everyone of them must be present in the record as fields, otherwise the record will generate an error record. If an incoming record contains a mail_use_template field, a template must be present in the configuration and the HTML mail format will be used. If the record also contains a mail_text field, its content will be used as an alternative text message to be used in the mail reader program of the recipient if it does not supports HTML.		null	false	false

Extra informations

The SendMail processor is aimed at sending an email (like for instance an alert email) from an incoming record. There are three ways an incoming record can generate an email according to the special fields it must embed. Here is a list of the record fields that generate a mail and how they work:

- **mail_text**: this is the simplest way for generating a mail. If present, this field means to use its content (value) as the payload of the mail to send. The mail is sent in text format if there is only this special field in the record. Otherwise, used with either mail_html or mail_use_template, the content of mail_text is the aletrnative text to the HTML mail that is generated.
- **mail_html**: this field specifies that the mail should be sent as HTML and the value of the field is mail payload. If mail_text is also present, its value is used as the alternative text for the mail. mail_html cannot be used with mail_use_template: only one of those two fields should be present in the record.
- **mail_use_template**: If present, this field specifies that the mail should be sent as HTML and the HTML content is to be generated from the template in the processor configuration key **html.template**. The template can contain parameters which must also be present in the record as fields. See documentation of html.template for further explanations. mail_use_template cannot be used with mail_html: only one of those two fields should be present in the record.

  If **allow_overwrite** configuration key is true, any mail.* (dot format) configuration key may be overwritten with a matching field in the record of the form mail_* (underscore format). For instance if allow_overwrite is true and mail.to is set to config_address@domain.com, a record generating a mail with a mail_to field set to record_address@domain.com will send a mail to record_address@domain.com.

  Apart from error records (when he is unable to process the incoming record or to send the mail), this processor is not expected to produce any output records.

SetJsonAsFields

The SetJsonAsFields processor reads the content of a string field containing a json string and sets each json attribute as a field of the current record. Note that this could be achieved with the EvaluateJsonPath processor, but this implies to declare each json first level attribute in the configuration and also to know by advance every one of them. Whereas for this simple case, the SetJsonAsFields processor does not require such a configuration and will work with any incoming json, regardless of the list of first level attributes.

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.SetJsonAsFields

Tags

json

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 46: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
debug	Enable debug. If enabled, debug information are written to stdout.		false	false	false
json.field	Field name of the string field that contains the json document to parse.		record_value	false	false
keep.json.field	Keep the original json field or not. Default is false so default is to remove the json field.		false	false	false
overwrite.existing.field	Overwrite an existing record field or not. Default is true so default is to remove the conflicting field.		true	false	false
omit.null.attributes	Omit json attributes with null values. Default is false so to set them as null record fields		false	false	false
omit.empty.string.attributes	Omit json attributes with empty string values. Default is false so to set them as empty string record fields		false	false	false

Extra informations

The SetJsonAsFields processor reads the content of a string field containing a json string and sets each json attribute as a field of the current record. Note that this could be achieved with the EvaluateJsonPath processor, but this implies to declare each json first level attribute in the configuration and also to know by advance every one of them. Whereas for this simple case, the SetJsonAsFields processor does not require such a configuration and will work with any incoming json, regardless of the list of first level attributes.

SplitField

This processor is used to create a new set of fields from one field (using split).

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.SplitField

Tags

parser, split, log, record

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 47: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
conflict.resolution.policy	What to do when a field with the same name already exists ?	overwrite_existing (if field already exist), keep_only_old_field (keep only old field)	keep_only_old_field	false	false
split.limit	Specify the maximum number of split to allow		10	false	false
split.counter.enable	Enable the counter of items returned by the split		false	false	false
split.counter.suffix	Enable the counter of items returned by the split		Counter	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 48: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
alternative split field	another split that could match	This processor is used to create a new set of fields from one field (using split).		null	**true**

Extra informations

This processor is used to create a new set of fields from one field (using split).

See Also:

com.hurence.logisland.processor.SplitField

SplitText

This is a processor that is used to split a String into fields according to a given Record mapping

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.SplitText

Tags

parser, regex, log, record

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 49: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
value.regex	the regex to match for the message value		null	false	false
value.fields	a comma separated list of fields corresponding to matching groups for the message value		null	false	false
key.regex	the regex to match for the message key		.*	false	false
key.fields	a comma separated list of fields corresponding to matching groups for the message key		record_key	false	false
record.type	default type of record		record	false	false
keep.raw.content	do we add the initial raw content ?		true	false	false
timezone.record.time	what is the time zone of the string formatted date for 'record_time' field.		UTC	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 50: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
alternative regex & mapping	another regex that could match	this regex will be tried if the main one has not matched. It must be in the form alt.value.regex.1 and alt.value.fields.1		null	**true**

Extra informations

This is a processor that is used to split a String into fields according to a given Record mapping.

See Also:

com.hurence.logisland.processor.SplitTextMultiline

SplitTextMultiline

No description provided.

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.SplitTextMultiline

Tags

None.

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 51: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
regex	the regex to match		null	false	false
fields	a comma separated list of fields corresponding to matching groups		null	false	false
event.type	the type of event		null	false	false

Extra informations

No description provided.

SplitTextWithProperties

This is a processor that is used to split a String into fields according to a given Record mapping

Module

com.hurence.logisland:logisland-processor-common:1.1.1

Class

com.hurence.logisland.processor.SplitTextWithProperties

Tags

parser, regex, log, record

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 52: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
value.regex	the regex to match for the message value		null	false	false
value.fields	a comma separated list of fields corresponding to matching groups for the message value		null	false	false
key.regex	the regex to match for the message key		.*	false	false
key.fields	a comma separated list of fields corresponding to matching groups for the message key		record_key	false	false
record.type	default type of record		record	false	false
keep.raw.content	do we add the initial raw content ?		true	false	false
properties.field	the field containing the properties to split and treat		properties	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 53: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
alternative regex & mapping	another regex that could match	this regex will be tried if the main one has not matched. It must be in the form alt.value.regex.1 and alt.value.fields.1		null	**true**

Extra informations

This is a processor that is used to split a String into fields according to a given Record mapping.

See Also:

com.hurence.logisland.processor.SplitTextMultiline

Other-processors

Find below the list.

ParseUserAgent

The user-agent processor allows to decompose User-Agent value from an HTTP header into several attributes of interest. There is no standard format for User-Agent strings, hence it is not easily possible to use regexp to handle them. This processor rely on the YAUAA library to do the heavy work.

Module

com.hurence.logisland:logisland-processor-useragent:1.1.1

Class

com.hurence.logisland.processor.useragent.ParseUserAgent

Tags

User-Agent, clickstream, DMP

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 54: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
debug	Enable debug.		false	false	false
cache.enabled	Enable caching. Caching to avoid to redo the same computation for many identical User-Agent strings.		true	false	false
cache.size	Set the size of the cache.		1000	false	false
useragent.field	Must contain the name of the field that contains the User-Agent value in the incoming record.		null	false	false
useragent.keep	Defines if the field that contained the User-Agent must be kept or not in the resulting records.		true	false	false
confidence.enabled	Enable confidence reporting. Each field will report a confidence attribute with a value comprised between 0 and 10000.		false	false	false
ambiguity.enabled	Enable ambiguity reporting. Reports a count of ambiguities.		false	false	false
fields	Defines the fields to be returned.		DeviceClass, DeviceName, DeviceBrand, DeviceCpu, DeviceFirmwareVersion, DeviceVersion, OperatingSystemClass, OperatingSystemName, OperatingSystemVersion, OperatingSystemNameVersion, OperatingSystemVersionBuild, LayoutEngineClass, LayoutEngineName, LayoutEngineVersion, LayoutEngineVersionMajor, LayoutEngineNameVer-	false	false

Extra informations

The user-agent processor allows to decompose User-Agent value from an HTTP header into several attributes of interest. There is no standard format for User-Agent strings, hence it is not easily possible to use regexp to handle them. This processor rely on the YAUAA library to do the heavy work.

BulkAddElasticsearch

Indexes the content of a Record in Elasticsearch using elasticsearch's bulk processor

Module

com.hurence.logisland:logisland-processor-elasticsearch:1.1.1

Class

com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch

Tags

elasticsearch

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values, and whether a property supports the Expression Language .

Table 55: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
elasticsearch.client.service	The instance of the Controller Service to use for accessing Elasticsearch.		null	false	false
default.index	The name of the index to insert into		null	false	**true**
default.type	The type of this document (used by Elasticsearch for indexing and searching)		null	false	**true**
timebased.index	do we add a date suffix	no (no date added to default index), today (today's date added to default index), yesterday (yesterday's date added to default index)	no	false	false
es.index.field	the name of the event field containing es index name => will override index value if set		null	false	false
es.type.field	the name of the event field containing es doc type => will override type value if set		null	false	false

Extra informations

Indexes the content of a Record in Elasticsearch using elasticsearch's bulk processor.

ConsolidateSession

The ConsolidateSession processor is the Logisland entry point to get and process events from the Web Analytics.As an example here is an incoming event from the Web Analytics:

"fields": [{ "name": "timestamp", "type": "long" },{ "name": "remoteHost", "type": "string"},{ "name": "record_type", "type": ["null", "string"], "default": null },{ "name": "record_id", "type": ["null", "string"], "default": null },{ "name": "location", "type": ["null", "string"], "default": null },{ "name": "hitType", "type": ["null", "string"], "default": null },{ "name": "eventCategory", "type": ["null", "string"], "default": null },{ "name": "eventAction", "type": ["null", "string"], "default": null },{ "name": "eventLabel", "type": ["null", "string"], "default": null },{ "name": "localPath", "type": ["null", "string"], "default": null },{ "name": "q", "type": ["null", "string"], "default": null },{ "name": "n", "type": ["null", "int"], "default": null },{ "name": "referer", "type": ["null", "string"], "default": null },{ "name": "viewportPixelWidth", "type": ["null", "int"], "default": null },{ "name": "viewportPixelHeight", "type": ["null", "int"], "default": null },{ "name": "screenPixelWidth", "type": ["null", "int"], "default": null },{ "name": "screenPixelHeight", "type": ["null", "int"], "default": null },{ "name": "partyId", "type": ["null", "string"], "default": null },{ "name": "sessionId", "type": ["null", "string"], "default": null },{ "name": "pageViewId", "type": ["null", "string"], "default": null },{ "name": "is_newSession", "type": ["null", "boolean"],"default": null },{ "name": "userAgentString", "type": ["null", "string"], "default": null },{ "name": "pageType", "type": ["null", "string"], "default": null },{ "name": "UserId", "type": ["null", "string"], "default": null },{ "name": "B2Bunit", "type": ["null", "string"], "default": null },{ "name": "pointOfService", "type": ["null", "string"], "default": null },{ "name": "companyID", "type": ["null", "string"], "default": null },{ "name": "GroupCode", "type": ["null", "string"], "default": null },{ "name": "userRoles", "type": ["null", "string"], "default": null },{ "name": "is_PunchOut", "type": ["null", "string"], "default": null }]The ConsolidateSession processor groups the records by sessions and compute the duration between now and the last received event. If the distance from the last event is beyond a given threshold (by default 30mn), then the session is considered closed.The ConsolidateSession is building an aggregated session object for each active session.This aggregated object includes: - The actual session duration. - A boolean representing wether the session is considered active or closed. Note: it is possible to ressurect a session if for instance an event arrives after a session has been marked closed. - User related infos: userId, B2Bunit code, groupCode, userRoles, companyId - First visited page: URL - Last visited page: URL The properties to configure the processor are: - sessionid.field: Property name containing the session identifier (default: sessionId). - timestamp.field: Property name containing the timestamp of the event (default: timestamp). - session.timeout: Timeframe of inactivity (in seconds) after which a session is considered closed (default: 30mn). - visitedpage.field: Property name containing the page visited by the customer (default: location). - fields.to.return: List of fields to return in the aggregated object. (default: N/A)

Module

com.hurence.logisland:logisland-processor-web-analytics:1.1.1

Class

com.hurence.logisland.processor.webAnalytics.ConsolidateSession

Tags

analytics, web, session

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 56: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
debug	Enable debug. If enabled, the original JSON string is embedded in the record_value field of the record.		null	false	false
session.timeout	session timeout in sec		1800	false	false
sessionid.field	the name of the field containing the session id => will override default value if set		sessionId	false	false
timestamp.field	the name of the field containing the timestamp => will override default value if set		h2kTimestamp	false	false
visitedpage.field	the name of the field containing the visited page => will override default value if set		location	false	false
userid.field	the name of the field containing the userId => will override default value if set		userId	false	false
fields.to.return	the list of fields to return		null	false	false
firstVisitedPage.out.field	the name of the field containing the first visited page => will override default value if set		firstVisitedPage	false	false
lastVisitedPage.out.field	the name of the field containing the last visited page => will override default value if set		lastVisitedPage	false	false
isSessionActive.out.field	the name of the field stating whether the session is active or not => will override default value if set		is_sessionActive	false	false
sessionDuration.out.field	the name of the field containing the session duration => will override default value if set		sessionDuration	false	false
eventsCounter.out.field	the name of the field containing the session duration => will override default value if set		eventsCounter	false	false
firstEventDateTime.out.field	the name of the field containing the date of the first event => will override default value if set		firstEventDateTime	false	false
lastEventDateTime.out.field	the name of the field containing the date of the last event => will override default value if set		lastEventDateTime	false	false
sessionInactivityDuration.out.field	the name of the field containing the session inactivity duration => will override default value if set		sessionInactivityDuration	false	false

Extra informations

The ConsolidateSession processor is the Logisland entry point to get and process events from the Web Analytics.As an example here is an incoming event from the Web Analytics:

“fields”: [{ “name”: “timestamp”, “type”: “long” },{ “name”: “remoteHost”, “type”: “string”},{ “name”: “record_type”, “type”: [“null”, “string”], “default”: null },{ “name”: “record_id”, “type”: [“null”, “string”], “default”: null },{ “name”: “location”, “type”: [“null”, “string”], “default”: null },{ “name”: “hitType”, “type”: [“null”, “string”], “default”: null },{ “name”: “eventCategory”, “type”: [“null”, “string”], “default”: null },{ “name”: “eventAction”, “type”: [“null”, “string”], “default”: null },{ “name”: “eventLabel”, “type”: [“null”, “string”], “default”: null },{ “name”: “localPath”, “type”: [“null”, “string”], “default”: null },{ “name”: “q”, “type”: [“null”, “string”], “default”: null },{ “name”: “n”, “type”: [“null”, “int”], “default”: null },{ “name”: “referer”, “type”: [“null”, “string”], “default”: null },{ “name”: “viewportPixelWidth”, “type”: [“null”, “int”], “default”: null },{ “name”: “viewportPixelHeight”, “type”: [“null”, “int”], “default”: null },{ “name”: “screenPixelWidth”, “type”: [“null”, “int”], “default”: null },{ “name”: “screenPixelHeight”, “type”: [“null”, “int”], “default”: null },{ “name”: “partyId”, “type”: [“null”, “string”], “default”: null },{ “name”: “sessionId”, “type”: [“null”, “string”], “default”: null },{ “name”: “pageViewId”, “type”: [“null”, “string”], “default”: null },{ “name”: “is_newSession”, “type”: [“null”, “boolean”],”default”: null },{ “name”: “userAgentString”, “type”: [“null”, “string”], “default”: null },{ “name”: “pageType”, “type”: [“null”, “string”], “default”: null },{ “name”: “UserId”, “type”: [“null”, “string”], “default”: null },{ “name”: “B2Bunit”, “type”: [“null”, “string”], “default”: null },{ “name”: “pointOfService”, “type”: [“null”, “string”], “default”: null },{ “name”: “companyID”, “type”: [“null”, “string”], “default”: null },{ “name”: “GroupCode”, “type”: [“null”, “string”], “default”: null },{ “name”: “userRoles”, “type”: [“null”, “string”], “default”: null },{ “name”: “is_PunchOut”, “type”: [“null”, “string”], “default”: null }]The ConsolidateSession processor groups the records by sessions and compute the duration between now and the last received event. If the distance from the last event is beyond a given threshold (by default 30mn), then the session is considered closed.The ConsolidateSession is building an aggregated session object for each active session.This aggregated object includes: - The actual session duration. - A boolean representing wether the session is considered active or closed. Note: it is possible to ressurect a session if for instance an event arrives after a session has been marked closed. - User related infos: userId, B2Bunit code, groupCode, userRoles, companyId - First visited page: URL - Last visited page: URL The properties to configure the processor are: - sessionid.field: Property name containing the session identifier (default: sessionId). - timestamp.field: Property name containing the timestamp of the event (default: timestamp). - session.timeout: Timeframe of inactivity (in seconds) after which a session is considered closed (default: 30mn). - visitedpage.field: Property name containing the page visited by the customer (default: location). - fields.to.return: List of fields to return in the aggregated object. (default: N/A)

DetectOutliers

Outlier Analysis: A Hybrid Approach

In order to function at scale, a two-phase approach is taken

For every data point

- Detect outlier candidates using a robust estimator of variability (e.g. median absolute deviation) that uses distributional sketching (e.g. Q-trees)
- Gather a biased sample (biased by recency)
- Extremely deterministic in space and cheap in computation

For every outlier candidate

- Use traditional, more computationally complex approaches to outlier analysis (e.g. Robust PCA) on the biased sample

- Expensive computationally, but run infrequently

This becomes a data filter which can be attached to a timeseries data stream within a distributed computational framework (i.e. Storm, Spark, Flink, NiFi) to detect outliers.

Module

com.hurence.logisland:logisland-processor-outlier-detection:1.1.1

Class

com.hurence.logisland.processor.DetectOutliers

Tags

analytic, outlier, record, iot, timeseries

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 57: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
value.field	the numeric field to get the value		record_value	false	false
time.field	the numeric field to get the value		record_time	false	false
output.record.type	the output type of the record		alert_match	false	false
rotation.policy.type		by_amount, by_time, never	by_amount	false	false
rotation.policy.amount			100	false	false
rotation.policy.unit		milliseconds, seconds, hours, days, months, years, points	points	false	false
chunking.policy.type		by_amount, by_time, never	by_amount	false	false
chunking.policy.amount			100	false	false
chunking.policy.unit		milliseconds, seconds, hours, days, months, years, points	points	false	false
sketchy.outlier.algorithm		SKETCHY_MOVING_MAD	SKETCHY_MOVING_MAD	false	false
batch.outlier.algorithm		RAD	RAD	false	false
global.statistics.min	minimum value		null	false	false
global.statistics.max	maximum value		null	false	false
global.statistics.mean	mean value		null	false	false
global.statistics.stddev	standard deviation value		null	false	false
zscore.cutoffs.normal	zscoreCutoffs level for normal outlier		0.000000000000001	false	false
zscore.cutoffs.moderate	zscoreCutoffs level for moderate outlier		1.5	false	false
zscore.cutoffs.severe	zscoreCutoffs level for severe outlier		10.0	false	false
zscore.cutoffs.notEnoughData	zscoreCutoffs level for notEnoughData outlier		100	false	false
smooth	do smoothing ?		false	false	false
decay	the decay		0.1	false	false
min.amount.to.predict	minAmountToPredict		100	false	false
min_zscore_percentile	minZscorePercentile		50.0	false	false
reservoir_size	the size of points reservoir		100	false	false
rpca.force.diff	No Description Provided.		null	false	false
rpca.lpenalty	No Description Provided.		null	false	false
rpca.min.records	No Description Provided.		null	false	false
rpca.spenalty	No Description Provided.		null	false	false
rpca.threshold	No Description Provided.		null	false	false

Extra informations

Outlier Analysis: A Hybrid Approach

In order to function at scale, a two-phase approach is taken

For every data point

- Detect outlier candidates using a robust estimator of variability (e.g. median absolute deviation) that uses distributional sketching (e.g. Q-trees)

- Gather a biased sample (biased by recency)
- Extremely deterministic in space and cheap in computation

For every outlier candidate

- Use traditional, more computationally complex approaches to outlier analysis (e.g. Robust PCA) on the biased sample
- Expensive computationally, but run infrequently

This becomes a data filter which can be attached to a timeseries data stream within a distributed computational framework (i.e. Storm, Spark, Flink, NiFi) to detect outliers.

EnrichRecordsElasticsearch

Enrich input records with content indexed in elasticsearch using multiget queries. Each incoming record must be possibly enriched with information stored in elasticsearch. Each outcoming record holds at least the input record plus potentially one or more fields coming from of one elasticsearch document.

Module

com.hurence.logisland:logisland-processor-elasticsearch:1.1.1

Class

com.hurence.logisland.processor.elasticsearch.EnrichRecordsElasticsearch

Tags

elasticsearch

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values, and whether a property supports the Expression Language .

Table 58: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
elasticsearch.client.service	The instance of the Controller Service to use for accessing Elasticsearch.		null	false	false
record.key	The name of field in the input record containing the document id to use in ES multiget query		null	false	**true**
es.index	The name of the ES index to use in multiget query.		null	false	**true**
es.type	The name of the ES type to use in multiget query.		default	false	**true**
es.includes.field	The name of the ES fields to include in the record.		•	false	**true**
es.excludes.field	The name of the ES fields to exclude.		N/A	false	false

Extra informations

Enrich input records with content indexed in elasticsearch using multiget queries. Each incoming record must be possibly enriched with information stored in elasticsearch. Each outcoming record holds at least the input record plus potentially one or more fields coming from of one elasticsearch document.

EvaluateXPath

Evaluates one or more XPaths against the content of a record. The results of those XPaths are assigned to new attributes in the records, depending on configuration of the Processor. XPaths are entered by adding user-defined properties; the name of the property maps to the Attribute Name into which the result will be placed. The value of the property must be a valid XPath expression. If the expression matches nothing, no attributes is added.

Module

com.hurence.logisland:logisland-processor-xml:1.1.1

Class

com.hurence.logisland.processor.xml.EvaluateXPath

Tags

XML, evaluate, XPath

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 59: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
source	Indicates the attribute containing the xml data to evaluate xpath against.		null	false	false
validate_dtd	Specifies whether or not the XML content should be validated against the DTD.	true, false	true	false	false
conflict.resolution.policy	What to do when a field with the same name already exists ?	overwrite_existing (if field already exist), keep_only_old_field (keep only old field)	keep_only_old_field	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 60: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
An attribute	An XPath expression	the attribute is set to the result of the XPath Expression.		null	false

Extra informations

Evaluates one or more XPaths against the content of a record. The results of those XPaths are assigned to new attributes in the records, depending on configuration of the Processor. XPaths are entered by adding user-defined properties; the name of the property maps to the Attribute Name into which the result will be placed. The value of the property must be a valid XPath expression. If the expression matches nothing, no attributes is added.

ExcelExtract

Consumes a Microsoft Excel document and converts each worksheet's line to a structured record. The processor is assuming to receive raw excel file as input record.

Module

com.hurence.logisland:logisland-processor-excel:1.1.1

Class

com.hurence.logisland.processor.excel.ExcelExtract

Tags

excel, processor, poi

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 61: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
sheets	Comma separated list of Excel document sheet names that should be extracted from the excel document. If this property is left blank then all of the sheets will be extracted from the Excel document. You can specify regular expressions. Any sheets not specified in this value will be ignored.			false	false
skip.columns	Comma delimited list of column numbers to skip. Use the columns number and not the letter designation. Use this to skip over columns anywhere in your worksheet that you don't want extracted as part of the record.			false	false
field.names	The comma separated list representing the names of columns of extracted cells. Order matters! You should use either field.names either field.row.header but not both together.		null	false	false
skip.rows	The row number of the first row to start processing.Use this to skip over rows of data at the top of your worksheet that are not part of the dataset.Empty rows of data anywhere in the spreadsheet will always be skipped, no matter what this value is set to.		0	false	false
record.type	Default type of record		excel_record	false	false
field.row.header	If set, field names mapping will be extracted from the specified row number. You should use either field.names either field.row.header but not both together.		null	false	false

Extra informations

Consumes a Microsoft Excel document and converts each worksheet's line to a structured record. The processor is assuming to receive raw excel file as input record.

FetchHBaseRow

Fetches a row from an HBase table. The Destination property controls whether the cells are added as flow file attributes, or the row is written to the flow file content as JSON. This processor may be used to fetch a fixed row on a interval by specifying the table and row id directly in the processor, or it may be used to dynamically fetch rows by referencing the table and row id from incoming flow files.

Module

com.hurence.logisland:logisland-processor-hbase:1.1.1

Class

com.hurence.logisland.processor.hbase.FetchHBaseRow

Tags

hbase, scan, fetch, get, enrich

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values, and whether a property supports the Expression Language .

Table 62: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
hbase.client.service	The instance of the Controller Service to use for accessing HBase.		null	false	false
table.name.field	The field containing the name of the HBase Table to fetch from.		null	false	**true**
row.identifier.field	The field containing the identifier of the row to fetch.		null	false	**true**
columns.field	The field containing an optional comma-separated list of “”<colFamily>:<colQualifier>”” pairs to fetch. To return all columns for a given family, leave off the qualifier such as “”<colFamily1>,<colFamily2>”“.		null	false	**true**
record.serializer	the serializer needed to i/o the record in the HBase row	com.hurence.logisland.serializer.KryoSerializer (serialize events as json blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), none (send events as bytes)	com.hurence.logisland.serializer.KryoSerializer	false	false
record.schema	the avro schema definition for the Avro serialization		null	false	false
table.name.default	The table to use if table name field is not set		null	false	false

Extra informations

Fetches a row from an HBase table. The Destination property controls whether the cells are added as flow file attributes, or the row is written to the flow file content as JSON. This processor may be used to fetch a fixed row on a interval by specifying the table and row id directly in the processor, or it may be used to dynamically fetch rows by referencing the table and row id from incoming flow files.

IncrementalWebSession

This processor creates and updates web-sessions based on incoming web-events. Note that both web-sessions and web-events ar

Firstly, web-events are grouped by their session identifier and processed in chronological order. Then each web-session associated to each group is retrieved from elasticsearch. In case none exists yet then a new web session is created based on the first web event. The following fields of the newly created web session are set based on the associated web event: session identifier, first timestamp, first visited page. Secondly, once created, or retrieved, the web session is updated by the remaining web-events. Updates have impacts on fields of the web session such as event counter, last visited page, session duration, … Before updates are actually applied, checks are performed to detect rules that would trigger the creation of a new session:

the duration between the web session and the web event must not exceed the specified time-out, the web session and the web event must have timestamps within the same day (at midnight a new web session is created), source of traffic (campaign, …) must be the same on the web session and the web event.

When a breaking rule is detected, a new web session is created with a new session identifier where as remaining web-events still have the original session identifier. The new session identifier is the original session suffixed with the character '#' followed with an incremented counter. This new session identifier is also set on the remaining web-events. Finally when all web events were applied, all web events -potentially modified with a new session identifier- are save in elasticsearch. And web sessions are passed to the next processor.

WebSession information are: - first and last visited page - first and last timestamp of processed event - total number of processed events - the userId - a boolean denoting if the web-session is still active or not - an integer denoting the duration of the web-sessions - optional fields that may be retrieved from the processed events

Module

com.hurence.logisland:logisland-processor-web-analytics:1.1.1

Class

com.hurence.logisland.processor.webAnalytics.IncrementalWebSession

Tags

analytics, web, session

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 63: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
debug	Enable debug. If enabled, debug information are logged.		false	false	false
es.session.index.field	Name of the field in the record defining the ES index containing the web session documents.		null	false	false
es.session.type.name	Name of the ES type of web session documents.		null	false	false
es.event.index.prefix	Prefix of the index containing the web event documents.		null	false	false
es.event.type.name	Name of the ES type of web event documents.		null	false	false
es.mapping.event.to.session.index.name	Name of the ES index containing the mapping of web session documents.		null	false	false
sessionid.field	the name of the field containing the session id => will override default value if set		sessionId	false	false
timestamp.field	the name of the field containing the timestamp => will override default value if set		h2kTimestamp	false	false
visitedpage.field	the name of the field containing the visited page => will override default value if set		location	false	false
userid.field	the name of the field containing the userId => will override default value if set		userId	false	false
fields.to.return	the list of fields to return		null	false	false
firstVisitedPage.out.field	the name of the field containing the first visited page => will override default value if set		firstVisitedPage	false	false
lastVisitedPage.out.field	the name of the field containing the last visited page => will override default value if set		lastVisitedPage	false	false
isSessionActive.out.field	the name of the field stating whether the session is active or not => will override default value if set		is_sessionActive	false	false
sessionDuration.out.field	the name of the field containing the session duration => will override default value if set		sessionDuration	false	false
sessionInactivityDuration.out.field	the name of the field containing the session inactivity duration => will override default value if set		sessionInactivityDuration	false	false
session.timeout	session timeout in sec		1800	false	false
eventsCounter.out.field	the name of the field containing the session duration => will override default value if set		eventsCounter	false	false
firstEventDateTime.out.field	the name of the field containing the date of the first event => will override default value if set		firstEventDateTime	false	false
lastEventDateTime.out.field	the name of the field containing the date of the last event => will override default value if set		lastEventDateTime	false	false
newSessionReason.out.field	the name of the field containing the reason why a new session was created => will override default value if set		reasonForNewSession	false	false
transactionIds.out.field	the name of the field containing all transactionIds => will override default value if set		transactionIds	false	false
source_of_traffic.suffix	Prefix for the source of the traffic related fields		source_of_traffic	false	false
elasticsearch.client.service	The instance of the Controller Service to use for accessing Elasticsearch.		null	false	false

Extra informations

This processor creates and updates web-sessions based on incoming web-events. Note that both web-sessions and web-events are

> Firstly, web-events are grouped by their session identifier and processed in chronological order. Then each web-session associated to each group is retrieved from elasticsearch. In case none exists yet then a new web session is created based on the first web event. The following fields of the newly created web session are set based on the associated web event: session identifier, first timestamp, first visited page. Secondly, once created, or retrieved, the web session is updated by the remaining web-events. Updates have impacts on fields of the web session such as event counter, last visited page, session duration, … Before updates are actually applied, checks are performed to detect rules that would trigger the creation of a new session:
>
> > the duration between the web session and the web event must not exceed the specified time-out, the web session and the web event must have timestamps within the same day (at midnight a new web session is created), source of traffic (campaign, …) must be the same on the web session and the web event.
>
> When a breaking rule is detected, a new web session is created with a new session identifier where as remaining web-events still have the original session identifier. The new session identifier is the original session suffixed with the character '#' followed with an incremented counter. This new session identifier is also set on the remaining web-events. Finally when all web events were applied, all web events -potentially modified with a new session identifier- are save in elasticsearch. And web sessions are passed to the next processor.

WebSession information are: - first and last visited page - first and last timestamp of processed event - total number of processed events - the userId - a boolean denoting if the web-session is still active or not - an integer denoting the duration of the web-sessions - optional fields that may be retrieved from the processed events

---

IpToFqdn

Translates an IP address into a FQDN (Fully Qualified Domain Name). An input field from the record has the IP as value. An new field is created and its value is the FQDN matching the IP address. The resolution mechanism is based on the underlying operating system. The resolution request may take some time, specially if the IP address cannot be translated into a FQDN. For these reasons this processor relies on the logisland cache service so that once a resolution occurs or not, the result is put into the cache. That way, the real request for the same IP is not re-triggered during a certain period of time, until the cache entry expires. This timeout is configurable but by default a request for the same IP is not triggered before 24 hours to let the time to the underlying DNS system to be potentially updated.

Module

com.hurence.logisland:logisland-processor-enrichment:1.1.1

Class

com.hurence.logisland.processor.enrichment.IpToFqdn

Tags

dns, ip, fqdn, domain, address, fqhn, reverse, resolution, enrich

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 64: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
ip.address.field	The name of the field containing the ip address to use.		null	false	false
fqdn.field	The field that will contain the full qualified domain name corresponding to the ip address.		null	false	false
overwrite.fqdn.field	If the field should be overwritten when it already exists.		false	false	false
cache.service	The name of the cache service to use.		null	false	false
cache.max.time	The amount of time, in seconds, for which a cached FQDN value is valid in the cache service. After this delay, the next new request to translate the same IP into FQDN will trigger a new reverse DNS request and the result will overwrite the entry in the cache. This allows two things: if the IP was not resolved into a FQDN, this will get a chance to obtain a FQDN if the DNS system has been updated, if the IP is resolved into a FQDN, this will allow to be more accurate if the DNS system has been updated. A value of 0 seconds disables this expiration mechanism. The default value is 84600 seconds, which corresponds to new requests triggered every day if a record with the same IP passes every day in the processor.		84600	false	false
resolution.timeout	The amount of time, in milliseconds, to wait at most for the resolution to occur. This avoids to block the stream for too much time. Default value is 1000ms. If the delay expires and no resolution could occur before, the FQDN field is not created. A special value of 0 disables the logisland timeout and the resolution request may last for many seconds if the IP cannot be translated into a FQDN by the underlying operating system. In any case, whether the timeout occurs in logisland of in the operating system, the fact that a timeout occurs is kept in the cache system so that a resolution request for the same IP will not occur before the cache entry expires.		1000	false	false
debug	If true, some additional debug fields are added. If the FQDN field is named X, a debug field named X_os_resolution_time_ms contains the resolution time in ms (using the operating system, not the cache). This field is added whether the resolution occurs or time is out. A debug field named X_os_resolution_timeout contains a boolean value to indicate if the timeout occurred. Finally, a debug field named X_from_cache contains a boolean value to indicate the origin of the FQDN field. The default value for this property is false (debug is disabled.		false	false	false

Extra informations

Translates an IP address into a FQDN (Fully Qualified Domain Name). An input field from the record has the IP as value. An new field is created and its value is the FQDN matching the IP address. The resolution mechanism is based on the underlying operating system. The resolution request may take some time, specially if the IP address cannot be translated into a FQDN. For these reasons this processor relies on the logisland cache service so that once a resolution occurs or not, the result is put into the cache. That way, the real request for the same IP is not re-triggered during a certain period of time, until the cache entry expires. This timeout is configurable but by default a request for the same IP is not triggered before 24 hours to let the time to the underlying DNS system to be potentially updated.

IpToGeo

Looks up geolocation information for an IP address. The attribute that contains the IP address to lookup must be provided in the **ip.address.field** property. By default, the geo information are put in a hierarchical structure. That is, if the name of the IP field is 'X', then the the geo attributes added by enrichment are added under a father field named X_geo. "_geo" is the default hierarchical suffix that may be changed with the **geo.hierarchical.suffix** property. If one wants to put the geo fields at the same level as the IP field, then the **geo.hierarchical** property should be set to false and then the geo attributes are created at the same level as him with the naming pattern X_geo_<geo_field>. "_geo_" is the default flat suffix but this may be changed with the **geo.flat.suffix** property. The IpToGeo processor requires a reference to an Ip to Geo service. This must be defined in the **iptogeo.service** property. The added geo fields are dependant on the underlying Ip to Geo service. The **geo.fields** property must contain the list of geo fields that should be created if data is available for the IP to resolve. This property defaults to "*" which means to add every available fields. If one only wants a subset of the fields, one must define a comma separated list of fields as a value for the **geo.fields** property. The list of the available geo fields is in the description of the **geo.fields** property.

Module

com.hurence.logisland:logisland-processor-enrichment:1.1.1

Class

com.hurence.logisland.processor.enrichment.IpToGeo

Tags

geo, enrich, ip

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 65: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
ip.address.field	The name of the field containing the ip address to use.		null	false	false
iptogeo.service	The reference to the IP to Geo service to use.		null	false	false
geo.fields	Comma separated list of geo information fields to add to the record. Defaults to '*', which means to include all available fields. If a list of fields is specified and the data is not available, the geo field is not created. The geo fields are dependant on the underlying defined Ip to Geo service. The currently only supported type of Ip to Geo service is the Maxmind Ip to Geo service. This means that the currently supported list of geo fields is the following:**continent**: the identified continent for this IP address. **continent_code**: the identified continent code for this IP address. **city**: the identified city for this IP address. **latitude**: the identified latitude for this IP address. **longitude**: the identified longitude for this IP address. **location**: the identified location for this IP address, defined as Geo-point expressed as a string with the format: 'latitude,longitude'. **accuracy_radius**: the approximate accuracy radius, in kilometers, around the latitude and longitude for the location. **time_zone**: the identified time zone for this IP address. **subdivision_N**: the identified subdivision for this IP address. N is a one-up number at the end of the attribute name, starting with 0. **subdivision_isocode_N**: the iso code matching the identified subdivision_N. **country**: the identified country for this IP address. **country_isocode**: the iso code for the identified country for this IP address. **postalcode**: the identified postal code for this IP address. **lookup_micros**: the number of microseconds that the geo lookup took. The Ip to Geo service must have the lookup_micros property enabled in order to have this field available.		•	false	false
geo.hierarchical	Should the additional geo information fields be added under a hierarchical father field or not.		true	false	false
geo.hierarchical.suffix	Suffix to use for the field holding geo information. If geo.hierarchical is true, then use this suffix appended to the IP field name to define the father field name. This may be used for instance to distinguish between geo fields with various locales using many Ip to Geo service instances.		_geo	false	false
geo.flat.suffix	Suffix to use for geo information fields when they are flat. If geo.hierarchical is false, then use this suffix appended to the IP field name but before the geo field name. This may be		_geo_	false	false

Extra informations

Looks up geolocation information for an IP address. The attribute that contains the IP address to lookup must be provided in the **ip.address.field** property. By default, the geo information are put in a hierarchical structure. That is, if the name of the IP field is 'X', then the the geo attributes added by enrichment are added under a father field named X_geo. "_geo" is the default hierarchical suffix that may be changed with the **geo.hierarchical.suffix** property. If one wants to put the geo fields at the same level as the IP field, then the **geo.hierarchical** property should be set to false and then the geo attributes are created at the same level as him with the naming pattern X_geo_<geo_field>. "_geo_" is the default flat suffix but this may be changed with the **geo.flat.suffix** property. The IpToGeo processor requires a reference to an Ip to Geo service. This must be defined in the **iptogeo.service** property. The added geo fields are dependant on the underlying Ip to Geo service. The **geo.fields** property must contain the list of geo fields that should be created if data is available for the IP to resolve. This property defaults to "*" which means to add every available fields. If one only wants a subset of the fields, one must define a comma separated list of fields as a value for the **geo.fields** property. The list of the available geo fields is in the description of the **geo.fields** property.

MatchIP

IP address Query matching (using **'Luwak <http://www.confluent.io/blog/real-time-full-text-search-with-luwak-and-samza/>)'_**

You can use this processor to handle custom events matching IP address (CIDR) The record sent from a matching an IP address record is tagged appropriately.

A query is expressed as a lucene query against a field like for example:

```
message:'bad exception'
error_count:[10 TO *]
bytes_out:5000
user_name:tom*
```

Please read the Lucene syntax guide for supported operations

> **Warning:** don't forget to set numeric fields property to handle correctly numeric ranges queries

Module

com.hurence.logisland:logisland-processor-querymatcher:1.1.1

Class

com.hurence.logisland.processor.MatchIP

Tags

analytic, percolator, record, record, query, lucene

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 66: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
numeric.fields	a comma separated string of numeric field to be matched		null	false	false
output.record.type	the output type of the record		alert_match	false	false
record.type.updatePolicy	Record type update policy		overwrite	false	false
policy.onmatch	the policy applied to match events: 'first' (default value) match events are tagged with the name and value of the first query that matched;'all' match events are tagged with all names and values of the queries that matched.		first	false	false
policy.onmiss	the policy applied to miss events: 'discard' (default value) drop events that did not match any query;'forward' include also events that did not match any query.		discard	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 67: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
query	some Lucene query	generate a new record when this query is matched		null	**true**

Extra informations

IP address Query matching (using **'Luwak <http://www.confluent.io/blog/real-time-full-text-search-with-luwak-and-samza/>)'_**

You can use this processor to handle custom events matching IP address (CIDR) The record sent from a matching an IP address record is tagged appropriately.

A query is expressed as a lucene query against a field like for example:

```
message:'bad exception'
error_count:[10 TO *]
bytes_out:5000
user_name:tom*
```

Please read the Lucene syntax guide for supported operations

Warning: don't forget to set numeric fields property to handle correctly numeric ranges queries

---

MatchQuery

Query matching based on Luwak

you can use this processor to handle custom events defined by lucene queries a new record is added to output each time a registered query is matched

A query is expressed as a lucene query against a field like for example:

```
message:'bad exception'
error_count:[10 TO *]
bytes_out:5000
user_name:tom*
```

Please read the Lucene syntax guide for supported operations

Warning: don't forget to set numeric fields property to handle correctly numeric ranges queries

Module

com.hurence.logisland:logisland-processor-querymatcher:1.1.1

Class

com.hurence.logisland.processor.MatchQuery

Tags

analytic, percolator, record, record, query, lucene

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 68: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
numeric.fields	a comma separated string of numeric field to be matched		null	false	false
output.record.type	the output type of the record		alert_match	false	false
record.type.updatePolicy	Record type update policy		overwrite	false	false
policy.onmatch	the policy applied to match events: 'first' (default value) match events are tagged with the name and value of the first query that matched;'all' match events are tagged with all names and values of the queries that matched.		first	false	false
policy.onmiss	the policy applied to miss events: 'discard' (default value) drop events that did not match any query;'forward' include also events that did not match any query.		discard	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 69: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
query	some Lucene query	generate a new record when this query is matched		null	**true**

Extra informations

Query matching based on Luwak

you can use this processor to handle custom events defined by lucene queries a new record is added to output each time a registered query is matched

A query is expressed as a lucene query against a field like for example:

```
message:'bad exception'
error_count:[10 TO *]
bytes_out:5000
user_name:tom*
```

Please read the Lucene syntax guide for supported operations

> **Warning:** don't forget to set numeric fields property to handle correctly numeric ranges queries.

MultiGetElasticsearch

Retrieves a content indexed in elasticsearch using elasticsearch multiget queries. Each incoming record contains information regarding the elasticsearch multiget query that will be performed. This information is stored in record fields whose names are configured in the plugin properties (see below) :

- index (String) : name of the elasticsearch index on which the multiget query will be performed. This field is mandatory and should not be empty, otherwise an error output record is sent for this specific incoming record.
- type (String) : name of the elasticsearch type on which the multiget query will be performed. This field is not mandatory.
- ids (String) : comma separated list of document ids to fetch. This field is mandatory and should not be empty, otherwise an error output record is sent for this specific incoming record.
- includes (String) : comma separated list of patterns to filter in (include) fields to retrieve. Supports wildcards. This field is not mandatory.
- excludes (String) : comma separated list of patterns to filter out (exclude) fields to retrieve. Supports wildcards. This field is not mandatory.

Each outcoming record holds data of one elasticsearch retrieved document. This data is stored in these fields :

- index (same field name as the incoming record) : name of the elasticsearch index.
- type (same field name as the incoming record) : name of the elasticsearch type.
- id (same field name as the incoming record) : retrieved document id.
- a list of String fields containing :
  - field name : the retrieved field name
  - field value : the retrieved field value

Module

com.hurence.logisland:logisland-processor-elasticsearch:1.1.1

Class

com.hurence.logisland.processor.elasticsearch.MultiGetElasticsearch

Tags

elasticsearch

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 70: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
elasticsearch.client.service	The instance of the Controller Service to use for accessing Elasticsearch.		null	false	false
es.index.field	the name of the incoming records field containing es index name to use in multiget query.		null	false	false
es.type.field	the name of the incoming records field containing es type name to use in multiget query		null	false	false
es.ids.field	the name of the incoming records field containing es document Ids to use in multiget query		null	false	false
es.includes.field	the name of the incoming records field containing es includes to use in multiget query		null	false	false
es.excludes.field	the name of the incoming records field containing es excludes to use in multiget query		null	false	false

Extra informations

Retrieves a content indexed in elasticsearch using elasticsearch multiget queries. Each incoming record contains information regarding the elasticsearch multiget query that will be performed. This information is stored in record fields whose names are configured in the plugin properties (see below) :

- index (String) : name of the elasticsearch index on which the multiget query will be performed. This field is mandatory and should not be empty, otherwise an error output record is sent for this specific incoming record.
- type (String) : name of the elasticsearch type on which the multiget query will be performed. This field is not mandatory.
- ids (String) : comma separated list of document ids to fetch. This field is mandatory and should not be empty, otherwise an error output record is sent for this specific incoming record.
- includes (String) : comma separated list of patterns to filter in (include) fields to retrieve. Supports wildcards. This field is not mandatory.
- excludes (String) : comma separated list of patterns to filter out (exclude) fields to retrieve. Supports wildcards. This field is not mandatory.

Each outcoming record holds data of one elasticsearch retrieved document. This data is stored in these fields :

- index (same field name as the incoming record) : name of the elasticsearch index.
- type (same field name as the incoming record) : name of the elasticsearch type.
- id (same field name as the incoming record) : retrieved document id.
- a list of String fields containing :
  - field name : the retrieved field name
  - field value : the retrieved field value

ParseBroEvent

The ParseBroEvent processor is the Logisland entry point to get and process Bro events. The Bro-Kafka plugin should be used and configured in order to have Bro events sent to Kafka. See the Bro/Logisland tutorial for an example of usage for this processor. The ParseBroEvent processor does some minor pre-processing on incoming Bro events from the Bro-Kafka plugin to adapt them to Logisland.

Basically the events coming from the Bro-Kafka plugin are JSON documents with a first level field indicating the type of the event. The ParseBroEvent processor takes the incoming JSON document, sets the event type in a record_type field and sets the original sub-fields of the JSON event as first level fields in the record. Also any dot in a field name is transformed into an underscore. Thus, for instance, the field id.orig_h becomes id_orig_h. The next processors in the stream can then process the Bro events generated by this ParseBroEvent processor.

As an example here is an incoming event from Bro:

{

“conn”: {

“id.resp_p”: 9092,

“resp_pkts”: 0,

“resp_ip_bytes”: 0,

“local_orig”: true,

“orig_ip_bytes”: 0,

“orig_pkts”: 0,

“missed_bytes”: 0,

“history”: “Cc”,

“tunnel_parents”: [],

“id.orig_p”: 56762,

“local_resp”: true,

“uid”: “Ct3Ms01I3Yc6pmMZx7”,

“conn_state”: “OTH”,

“id.orig_h”: “172.17.0.2”,

“proto”: “tcp”,

“id.resp_h”: “172.17.0.3”,

“ts”: 1487596886.953917

}

}

It gets processed and transformed into the following Logisland record by the ParseBroEvent processor:

“@timestamp”: “2017-02-20T13:36:32Z”

“record_id”: “6361f80a-c5c9-4a16-9045-4bb51736333d”

“record_time”: 1487597792782

“record_type”: “conn”

“id_resp_p”: 9092

"resp_pkts": 0

"resp_ip_bytes": 0

"local_orig": true

"orig_ip_bytes": 0

"orig_pkts": 0

"missed_bytes": 0

"history": "Cc"

"tunnel_parents": []

"id_orig_p": 56762

"local_resp": true

"uid": "Ct3Ms01I3Yc6pmMZx7"

"conn_state": "OTH"

"id_orig_h": "172.17.0.2"

"proto": "tcp"

"id_resp_h": "172.17.0.3"

"ts": 1487596886.953917

Module

com.hurence.logisland:logisland-processor-cyber-security:1.1.1

Class

com.hurence.logisland.processor.bro.ParseBroEvent

Tags

bro, security, IDS, NIDS

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 71: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
debug	Enable debug. If enabled, the original JSON string is embedded in the record_value field of the record.		false	false	false

Extra informations

The ParseBroEvent processor is the Logisland entry point to get and process Bro events. The Bro-Kafka plugin should be used and configured in order to have Bro events sent to Kafka. See the Bro/Logisland tutorial for an example of usage for this processor. The ParseBroEvent processor does some minor pre-processing on incoming Bro events from the Bro-Kafka plugin to adapt them to Logisland.

Basically the events coming from the Bro-Kafka plugin are JSON documents with a first level field indicating the type of the event. The ParseBroEvent processor takes the incoming JSON document, sets the event type in a record_type field and sets the original sub-fields of the JSON event as first level fields in the record. Also any dot in a field name is transformed into an underscore. Thus, for instance, the field id.orig_h becomes id_orig_h. The next processors in the stream can then process the Bro events generated by this ParseBroEvent processor.

As an example here is an incoming event from Bro:

{

“conn”: {

“id.resp_p”: 9092,

“resp_pkts”: 0,

“resp_ip_bytes”: 0,

“local_orig”: true,

“orig_ip_bytes”: 0,

“orig_pkts”: 0,

“missed_bytes”: 0,

“history”: “Cc”,

“tunnel_parents”: [],

“id.orig_p”: 56762,

“local_resp”: true,

“uid”: “Ct3Ms01I3Yc6pmMZx7”,

“conn_state”: “OTH”,

“id.orig_h”: “172.17.0.2”,

“proto”: “tcp”,

“id.resp_h”: “172.17.0.3”,

“ts”: 1487596886.953917

}

}

It gets processed and transformed into the following Logisland record by the ParseBroEvent processor:

“@timestamp”: “2017-02-20T13:36:32Z”

“record_id”: “6361f80a-c5c9-4a16-9045-4bb51736333d”

“record_time”: 1487597792782

“record_type”: “conn”

“id_resp_p”: 9092

"resp_pkts": 0

"resp_ip_bytes": 0

"local_orig": true

"orig_ip_bytes": 0

"orig_pkts": 0

"missed_bytes": 0

"history": "Cc"

"tunnel_parents": []

"id_orig_p": 56762

"local_resp": true

"uid": "Ct3Ms01I3Yc6pmMZx7"

"conn_state": "OTH"

"id_orig_h": "172.17.0.2"

"proto": "tcp"

"id_resp_h": "172.17.0.3"

"ts": 1487596886.953917

ParseGitlabLog

The Gitlab logs processor is the Logisland entry point to get and process Gitlab logs. This allows for instance to monitor activities in your Gitlab server. The expected input of this processor are records from the production_json.log log file of Gitlab which contains JSON records. You can for instance use the kafkacat command to inject those logs into kafka and thus Logisland.

Module

com.hurence.logisland:logisland-processor-common-logs:1.1.1

Class

com.hurence.logisland.processor.commonlogs.gitlab.ParseGitlabLog

Tags

logs, gitlab

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 72: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
debug	Enable debug. If enabled, the original JSON string is embedded in the record_value field of the record.		false	false	false

Extra informations

The Gitlab logs processor is the Logisland entry point to get and process Gitlab logs. This allows for instance to monitor activities in your Gitlab server. The expected input of this processor are records from the production_json.log log file of Gitlab which contains JSON records. You can for instance use the kafkacat command to inject those logs into kafka and thus Logisland.

ParseNetflowEvent

The Netflow V5 processor is the Logisland entry point to process Netflow (V5) events. NetFlow is a feature introduced on Cisco routers that provides the ability to collect IP network traffic.We can distinguish 2 components:

- Flow exporter: aggregates packets into flows and exports flow records (binary format) towards one or more flow collectors
- Flow collector: responsible for reception, storage and pre-processing of flow data received from a flow exporter

The collected data are then available for analysis purpose (intrusion detection, traffic analysis. . .) Netflow are sent to kafka in order to be processed by logisland. In the tutorial we will simulate Netflow traffic using nfgen. this traffic will be sent to port 2055. The we rely on nifi to listen of that port for incoming netflow (V5) traffic and send them to a kafka topic. The Netflow processor could thus treat these events and generate corresponding logisland records. The following processors in the stream can then process the Netflow records generated by this processor.

Module

com.hurence.logisland:logisland-processor-cyber-security:1.1.1

Class

com.hurence.logisland.processor.netflow.ParseNetflowEvent

Tags

netflow, security

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 73: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
debug	Enable debug. If enabled, the original JSON string is embedded in the record_value field of the record.		false	false	false
output.record.type	the output type of the record		netflowevent	false	false
enrich.record	Enrich data. If enabledthe netflow record is enriched with inferred data		false	false	false

Extra informations

The Netflow V5 processor is the Logisland entry point to process Netflow (V5) events. NetFlow is a feature introduced on Cisco routers that provides the ability to collect IP network traffic.We can distinguish 2 components:

- Flow exporter: aggregates packets into flows and exports flow records (binary format) towards one or more flow collectors
- Flow collector: responsible for reception, storage and pre-processing of flow data received from a flow exporter

The collected data are then available for analysis purpose (intrusion detection, traffic analysis…) Netflow are sent to kafka in order to be processed by logisland. In the tutorial we will simulate Netflow traffic using nfgen. this traffic will be sent to port 2055. The we rely on nifi to listen of that port for incoming netflow (V5) traffic and send them to a kafka topic. The Netflow processor could thus treat these events and generate corresponding logisland records. The following processors in the stream can then process the Netflow records generated by this processor.

ParseNetworkPacket

The ParseNetworkPacket processor is the LogIsland entry point to parse network packets captured either off-the-wire (stream mode) or in pcap format (batch mode). In batch mode, the processor decodes the bytes of the incoming pcap record, where a Global header followed by a sequence of [packet header, packet data] pairs are stored. Then, each incoming pcap event is parsed into n packet records. The fields of packet headers are then extracted and made available in dedicated record fields. See the Capturing Network packets tutorial for an example of usage of this processor.

Module

com.hurence.logisland:logisland-processor-cyber-security:1.1.1

Class

com.hurence.logisland.processor.networkpacket.ParseNetworkPacket

Tags

PCap, security, IDS, NIDS

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 74: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
debug	Enable debug.		false	false	false
flow.mode	Flow Mode. Indicate whether packets are provided in batch mode (via pcap files) or in stream mode (without headers). Allowed values are batch and stream.	batch, stream	null	false	false

Extra informations

No additional information is provided

PutHBaseCell

Adds the Contents of a Record to HBase as the value of a single cell

Module

com.hurence.logisland:logisland-processor-hbase:1.1.1

Class

com.hurence.logisland.processor.hbase.PutHBaseCell

Tags

hadoop, hbase

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values, and whether a property supports the Expression Language .

Table 75: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
hbase.client.service	The instance of the Controller Service to use for accessing HBase.		null	false	false
table.name.field	The field containing the name of the HBase Table to put data into		null	false	**true**
row.identifier.field	Specifies field containing the Row ID to use when inserting data into HBase		null	false	**true**
row.identifier.encoding.strategy	Specifies the data type of Row ID used when inserting data into HBase. The default behavior is to convert the row id to a UTF-8 byte array. Choosing Binary will convert a binary formatted string to the correct byte[] representation. The Binary option should be used if you are using Binary row keys in HBase	String (Stores the value of row id as a UTF-8 String.), Binary (Stores the value of the rows id as a binary byte array. It expects that the row id is a binary formatted string.)	String	false	false
column.family.field	The field containing the Column Family to use when inserting data into HBase		null	false	**true**
column.qualifier.field	The field containing the Column Qualifier to use when inserting data into HBase		null	false	**true**
batch.size	The maximum number of Records to process in a single execution. The Records will be grouped by table, and a single Put per table will be performed.		25	false	false
record.schema	the avro schema definition for the Avro serialization		null	false	false
record.serializer	the serializer needed to i/o the record in the HBase row	com.hurence.logisland.serializer.KryoSerializer (serialize events as json blocs), com.hurence.logisland.serializer.JsonSerializer (serialize events as json blocs), com.hurence.logisland.serializer.AvroSerializer (serialize events as avro blocs), none (send events as bytes)	com.hurence.logisland.serializer.KryoSerializer	false	false
table.name.default	The table table to use if table name field is not set		null	false	false
column.family.default	The column family to use if column family field is not set		null	false	false
column.qualifier.default	The column qualifier to use if column qualifier field is not set		null	false	false

Extra informations

Adds the Contents of a Record to HBase as the value of a single cell.

RunPython

!!!! WARNING !!!!

The RunPython processor is currently an experimental feature : it is delivered as is, with the current set of features and is subject to modifications in API or anything else in further logisland releases without warnings. There is no tutorial yet. If you want to play with this processor, use the python-processing.yml example and send the apache logs of the index apache logs tutorial. The debug stream processor at the end of the stream should output events in stderr file of the executors from the spark console.

This processor allows to implement and run a processor written in python. This can be done in 2 ways. Either directly defining the process method code in the **script.code.process** configuration property or poiting to an external python module script file in the **script.path** configuration property. Directly defining methods is called the inline mode whereas using a script file is called the file mode. Both ways are mutually exclusive. Whether using the inline of file mode, your python code may depend on some python dependencies. If the set of python dependencies already delivered with the Logisland framework is not sufficient, you can use the **dependencies.path** configuration property to give their location. Currently only the nltk python library is delivered with Logisland.

Module

com.hurence.logisland:logisland-processor-scripting:1.1.1

Class

com.hurence.logisland.processor.scripting.python.RunPython

Tags

scripting, python

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 76: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
script.code.imports	For inline mode only. This is the python code that should hold the import statements if required.		null	false	false
script.code.init	The python code to be called when the processor is initialized. This is the python equivalent of the init method code for a java processor. This is not mandatory but can only be used if **script.code.process** is defined (inline mode).		null	false	false
script.code.process	The python code to be called to process the records. This is the pyhton equivalent of the process method code for a java processor. For inline mode, this is the only minimum required configuration property. Using this property, you may also optionally define the **script.code.init** and **script.code.imports** properties.		null	false	false
script.path	The path to the user's python processor script. Use this property for file mode. Your python code must be in a python file with the following constraints: let's say your pyhton script is named MyProcessor.py. Then MyProcessor.py is a module file that must contain a class named MyProcessor which must inherits from the Logisland delivered class named AbstractProcessor. You can then define your code in the process method and in the other traditional methods (init. . .) as you would do in java in a class inheriting from the AbstractProcessor java class.		null	false	false
dependencies.path	The path to the additional dependencies for the user's python code, whether using inline or file mode. This is optional as your code may not have additional dependencies. If you defined **script.path** (so using file mode) and if **dependencies.path** is not defined, Logisland will scan a potential directory named **dependencies** in the same directory where the script file resides and if it exists, any python code located there will be loaded as dependency as needed.		null	false	false
logisland.dependencies.path	The path to the directory containing the python dependencies shipped with logisland. You should not have to tune this parameter.		null	false	false

Extra informations

!!!! WARNING !!!!

The RunPython processor is currently an experimental feature : it is delivered as is, with the current set of features and is subject to modifications in API or anything else in further logisland releases without warnings. There is no tutorial yet. If you want to play with this processor, use the python-processing.yml example and send the apache logs of the index apache logs tutorial. The debug stream processor at the end of the stream should output events in stderr file of the executors from the spark console.

This processor allows to implement and run a processor written in python. This can be done in 2 ways. Either directly defining the process method code in the **script.code.process** configuration property or poiting to an external python module script file in the **script.path** configuration property. Directly defining methods is called the inline mode whereas using a script file is called the file mode. Both ways are mutually exclusive. Whether using the inline of file mode, your python code may depend on some python dependencies. If the set of python dependencies already delivered with the Logisland framework is not sufficient, you can use the **dependencies.path** configuration property to give their location. Currently only the nltk python library is delivered with Logisland.

SampleRecords

Query matching based on Luwak

you can use this processor to handle custom events defined by lucene queries a new record is added to output each time a registered query is matched

A query is expressed as a lucene query against a field like for example:

```
message:'bad exception'
error_count:[10 TO *]
bytes_out:5000
user_name:tom*
```

Please read the Lucene syntax guide for supported operations

> **Warning:** don't forget to set numeric fields property to handle correctly numeric ranges queries

Module

com.hurence.logisland:logisland-processor-sampling:1.1.1

Class

com.hurence.logisland.processor.SampleRecords

Tags

analytic, sampler, record, iot, timeseries

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 77: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
record.value.field	the name of the numeric field to sample		record_value	false	false
record.time.field	the name of the time field to sample		record_time	false	false
sampling.algorithm	the implementation of the algorithm	none, lttb, average, first_item, min_max, mode_median	null	false	false
sampling.parameter	the parmater of the algorithm		null	false	false

Extra informations

Query matching based on Luwak

you can use this processor to handle custom events defined by lucene queries a new record is added to output each time a registered query is matched

A query is expressed as a lucene query against a field like for example:

```
message:'bad exception'
error_count:[10 TO *]
bytes_out:5000
user_name:tom*
```

Please read the Lucene syntax guide for supported operations

> **Warning:** don't forget to set numeric fields property to handle correctly numeric ranges queries

---

URLDecoder

Decode one or more field containing an URL with possibly special chars encoded ...

Module

com.hurence.logisland:logisland-processor-web-analytics:1.1.1

Class

com.hurence.logisland.processor.webAnalytics.URLDecoder

Tags

record, fields, Decode

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 78: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
decode.fields	List of fields (URL) to decode		null	false	false
charset	Charset to use to decode the URL		UTF-8	false	false

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 79: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
fields to decode	a default value	Decode one or more fields from the record		null	false

Extra informations

Decode one or more field containing an URL with possibly special chars encoded.

---

setSourceOfTraffic

Compute the source of traffic of a web session. Users arrive at a website or application through a variety of sources, including advertising/paying campaigns, search engines, social networks, referring sites or direct access. When analysing user experience on a webshop, it is crucial to collect, process, and report the campaign and traffic-source data. To compute the source of traffic of a web session, the user has to provide the utm_* related properties if available i-e: **utm_source.field**, **utm_medium.field**, **utm_campaign.field**, **utm_content.field**, **utm_term.field**) , the referer (**referer.field** property) and the first visited page of the session (**first.visited.page.field** property). By default the source of traffic information are placed in a flat structure (specified by the **source_of_traffic.suffix** property with a default value of source_of_traffic). To work properly the setSourceOfTraffic processor needs to have access to an Elasticsearch index containing a list of the most popular search engines and social networks. The ES index (specified by the **es.index** property) should be structured such that the _id of an ES document MUST be the name of the domain. If the domain is a search engine, the related ES doc MUST have a boolean field (default being search_engine) specified by the property **es.search_engine.field** with a value set to true. If the domain is a social network , the related ES doc MUST have a boolean field (default being social_network) specified by the property **es.social_network.field** with a value set to true.

Module

com.hurence.logisland:logisland-processor-web-analytics:1.1.1

Class

com.hurence.logisland.processor.webAnalytics.setSourceOfTraffic

Tags

session, traffic, source, web, analytics

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 80: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
referer.field	Name of the field containing the referer value in the session		referer	false	false
first.visited.page.field	Name of the field containing the first visited page in the session		firstVisitedPage	false	false
utm_source.field	Name of the field containing the utm_source value in the session		utm_source	false	false
utm_medium.field	Name of the field containing the utm_medium value in the session		utm_medium	false	false
utm_campaign.field	Name of the field containing the utm_campaign value in the session		utm_campaign	false	false
utm_content.field	Name of the field containing the utm_content value in the session		utm_content	false	false
utm_term.field	Name of the field containing the utm_term value in the session		utm_term	false	false
source_of_traffic.suffix	Suffix for the source of the traffic related fields		source_of_traffic	false	false
source_of_traffic.hierarchical	Should the additional source of trafic information fields be added under a hierarchical father field or not.		false	false	false
elasticsearch.client.service	The instance of the Controller Service to use for accessing Elasticsearch.		null	false	false
cache.service	Name of the cache service to use.		null	false	false
cache.validity.timeout	Timeout validity (in seconds) of an entry in the cache.		0	false	false
debug	If true, an additional debug field is added. If the source info fields prefix is X, a debug field named X_from_cache contains a boolean value to indicate the origin of the source fields. The default value for this property is false (debug is disabled).		false	false	false
es.index	Name of the ES index containing the list of search engines and social network.		null	false	false
es.type	Name of the ES type to use.		default	false	false
es.search_engine.field	Name of the ES field used to specify that the domain is a search engine.		search_engine	false	false
es.social_network.field	Name of the ES field used to specify that the domain is a social network.		social_network	false	false

Extra informations

Compute the source of traffic of a web session. Users arrive at a website or application through a variety of sources, including advertising/paying campaigns, search engines, social networks, referring sites or direct access. When analysing user experience on a webshop, it is crucial to collect, process, and report the campaign and traffic-source data. To compute the source of traffic of a web session, the user has to provide the utm_* related properties if available i-e: **utm_source.field**, **utm_medium.field**, **utm_campaign.field**, **utm_content.field**, **utm_term.field**) , the referer (**referer.field** property) and the first visited page of the session (**first.visited.page.field** property). By default the source of traffic information are placed in a flat structure (specified by the **source_of_traffic.suffix** property with a default value of source_of_traffic). To work properly the setSourceOfTraffic processor needs to have access to an

Elasticsearch index containing a list of the most popular search engines and social networks. The ES index (specified by the **es.index** property) should be structured such that the _id of an ES document MUST be the name of the domain. If the domain is a search engine, the related ES doc MUST have a boolean field (default being search_engine) specified by the property **es.search_engine.field** with a value set to true. If the domain is a social network , the related ES doc MUST have a boolean field (default being social_network) specified by the property **es.social_network.field** with a value set to true.

Services

Find below the list.

CSVKeyValueCacheService

A cache that store csv lines as records loaded from a file

Module

com.hurence.logisland:logisland-service-inmemory-cache:1.1.1

Class

com.hurence.logisland.service.cache.CSVKeyValueCacheService

Tags

csv, service, cache

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 81: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
csv.format	a configuration for loading csv	default (Standard comma separated format, as for RFC4180 but allowing empty lines. Settings are: withDelimiter(',') withQuote('"') withRecord-Separator("rn") withIgnoreEmpty-Lines(true)), excel (Excel file format (using a comma as the value delimiter). Note that the actual value delimiter used by Excel is locale dependent, it might be necessary to customize this format to accom-modate to your regional settings. withDelimiter(',') withQuote('"') withRecord-Separator("rn") withIgnoreEmpty-Lines(false) withAllowMiss-ingColumn-Names(true)), excel_fr (Excel file format (using a comma as the value delimiter). Note that the actual value delimiter used by Excel is locale dependent, it might be necessary to customize this format to accom-modate to your regional settings. withDelimiter(';') withQuote('"') withRecord-Separator("rn") withIgnoreEmpty-Lines(false) withAllowMiss-ingColumn-Names(true)), mysql (Default	default	false	false

Extra informations

No additional information is provided

CassandraControllerService

Provides a controller service that for the moment only allows to bulkput records into cassandra.

Module

com.hurence.logisland:logisland-service-cassandra-client:1.1.1

Class

com.hurence.logisland.service.cassandra.CassandraControllerService

Tags

cassandra, service

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 82: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
cassandra.hosts	Cassandra cluster hosts as a comma separated value list		null	false	false
cassandra.port	Cassandra cluster port		null	false	false
cassandra.with-ssl	If this property is true, use SSL. Default is no SSL (false).		false	false	false
cassandra.with-credentials	If this property is true, use credentials. Default is no credentials (false).		false	false	false
cassandra.credentials.user	The user name to use for authentication. cassandra.with-credentials must be true for that property to be used.		null	false	false
cassandra.credentials.password	The user password to use for authentication. cassandra.with-credentials must be true for that property to be used.		null	false	false
batch.size	The preferred number of Records to setField to the database in a single transaction		1000	false	false
bulk.size	bulk size in MB		5	false	false
flush.interval	flush interval in ms		500	false	false

Extra informations

No additional information is provided

Elasticsearch_2_4_0_ClientService

Implementation of ElasticsearchClientService for Elasticsearch 2.4.0.

Module

com.hurence.logisland:logisland-service-elasticsearch_2_4_0-client:1.1.1

Class

com.hurence.logisland.service.elasticsearch.Elasticsearch_2_4_0_ClientService

Tags

elasticsearch, client

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values, and whether a property is considered "sensitive"..

Table 83: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
backoff.policy	strategy for retrying to execute requests in bulkRequest	noBackoff (when a request fail there won't be any retry.), constantBackoff (wait a fixed amount of time between retries, using user put retry number and throttling delay), exponentialBackoff (time waited between retries grow exponentially, using user put retry number and throttling delay), defaultExponentialBackoff (time waited between retries grow exponentially, using es default parameters)	defaultExponentialBackoff	false	false
throttling.delay	number of time we should wait between each retry (in milliseconds)		500	false	false
num.retry	number of time we should try to inject a bulk into es		3	false	false
batch.size	The preferred number of Records to setField to the database in a single transaction		1000	false	false
bulk.size	bulk size in MB		5	false	false
flush.interval	flush interval in sec		5	false	false
concurrent.requests	setConcurrentRequests		2	false	false
cluster.name	Name of the ES cluster (for example, elasticsearch_brew). Defaults to 'elasticsearch'		elasticsearch	false	false
ping.timeout	The ping timeout used to determine when a node is unreachable. For example, 5s (5 seconds). If non-local recommended is 30s		5s	false	false
sampler.interval	How often to sample / ping the nodes listed and connected. For example, 5s (5 seconds). If non-local recommended is 30s.		5s	false	false
username	Username to access the Elasticsearch cluster		null	false	false
password	Password to access the Elasticsearch cluster		null	**true**	false
shield.location	Specifies the path to the JAR for the Elasticsearch Shield plugin. If the Elasticsearch cluster has been secured with the Shield plugin, then the Shield plugin JAR must also be available to this processor. Note: Do NOT place the Shield JAR into NiFi's lib/ directory, doing so will prevent the Shield plugin from being loaded.		null	false	false
hosts	ElasticSearch Hosts, which should be comma separated and colon for hostname/port host1:port,host2:port,.... For example testcluster:9300.		null	false	false
ssl.context.service	The SSL Context Service used to provide client certificate information for TLS/SSL connections. This service only applies if the		null	false	false

Extra informations

No additional information is provided

Elasticsearch_5_4_0_ClientService

Implementation of ElasticsearchClientService for Elasticsearch 5.4.0.

Module

com.hurence.logisland:logisland-service-elasticsearch_5_4_0-client:1.1.1

Class

com.hurence.logisland.service.elasticsearch.Elasticsearch_5_4_0_ClientService

Tags

elasticsearch, client

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values, and whether a property is considered "sensitive"..

Table 84: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
backoff.policy	strategy for retrying to execute requests in bulkRequest	noBackoff (when a request fail there won't be any retry.), constantBackoff (wait a fixed amount of time between retries, using user put retry number and throttling delay), exponentialBackoff (time waited between retries grow exponentially, using user put retry number and throttling delay), defaultExponentialBackoff (time waited between retries grow exponentially, using es default parameters)	defaultExponentialBackoff	false	false
throttling.delay	number of time we should wait between each retry (in milliseconds)		500	false	false
num.retry	number of time we should try to inject a bulk into es		3	false	false
batch.size	The preferred number of Records to setField to the database in a single transaction		1000	false	false
bulk.size	bulk size in MB		5	false	false
flush.interval	flush interval in sec		5	false	false
concurrent.requests	setConcurrentRequests		2	false	false
cluster.name	Name of the ES cluster (for example, elasticsearch_brew). Defaults to 'elasticsearch'		elasticsearch	false	false
ping.timeout	The ping timeout used to determine when a node is unreachable. For example, 5s (5 seconds). If non-local recommended is 30s		5s	false	false
sampler.interval	How often to sample / ping the nodes listed and connected. For example, 5s (5 seconds). If non-local recommended is 30s.		5s	false	false
username	Username to access the Elasticsearch cluster		null	false	false
password	Password to access the Elasticsearch cluster		null	**true**	false
shield.location	Specifies the path to the JAR for the Elasticsearch Shield plugin. If the Elasticsearch cluster has been secured with the Shield plugin, then the Shield plugin JAR must also be available to this processor. Note: Do NOT place the Shield JAR into NiFi's lib/ directory, doing so will prevent the Shield plugin from being loaded.		null	false	false
hosts	ElasticSearch Hosts, which should be comma separated and colon for hostname/port host1:port,host2:port,.... For example testcluster:9300.		null	false	false
ssl.context.service	The SSL Context Service used to provide client certificate information for TLS/SSL connections. This service only applies if the		null	false	false

Extra informations

No additional information is provided

HBase_1_1_2_ClientService

Implementation of HBaseClientService for HBase 1.1.2. This service can be configured by providing a comma-separated list of configuration files, or by specifying values for the other properties. If configuration files are provided, they will be loaded first, and the values of the additional properties will override the values from the configuration files. In addition, any user defined properties on the processor will also be passed to the HBase configuration.

Module

com.hurence.logisland:logisland-service-hbase_1_1_2-client:1.1.1

Class

com.hurence.logisland.service.hbase.HBase_1_1_2_ClientService

Tags

hbase, client

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values, and whether a property supports the Expression Language .

Table 85: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
hadoop.configuration.files	Comma-separated list of Hadoop Configuration files, such as hbase-site.xml and core-site.xml for kerberos, including full paths to the files.		null	false	false
zookeeper.quorum	Comma-separated list of ZooKeeper hosts for HBase. Required if Hadoop Configuration Files are not provided.		null	false	false
zookeeper.client.port	The port on which ZooKeeper is accepting client connections. Required if Hadoop Configuration Files are not provided.		null	false	false
zookeeper.znode.parent	The ZooKeeper ZNode Parent value for HBase (example: /hbase). Required if Hadoop Configuration Files are not provided.		null	false	false
hbase.client.retries	The number of times the HBase client will retry connecting. Required if Hadoop Configuration Files are not provided.		3	false	false
phoenix.client.jar.location	The full path to the Phoenix client JAR. Required if Phoenix is installed on top of HBase.		null	false	**true**

Dynamic Properties

Dynamic Properties allow the user to specify both the name and value of a property.

Table 86: dynamic-properties

Name	Value	Description	Allowable Values	Default Value	EL
The name of an HBase configuration property.	The value of the given HBase configuration property.	These properties will be set on the HBase configuration after loading any provided configuration files.		null	false

Extra informations

No additional information is provided

LRUKeyValueCacheService

A controller service for caching data by key value pair with LRU (last recently used) strategy. using LinkedHashMap

Module

com.hurence.logisland:logisland-service-inmemory-cache:1.1.1

Class

com.hurence.logisland.service.cache.LRUKeyValueCacheService

Tags

cache, service, key, value, pair, LRU

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 87: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
cache.size	The maximum number of element in the cache.		16384	false	false

Extra informations

No additional information is provided

MaxmindIpToGeoService

Implementation of the IP 2 GEO Service using maxmind lite db file

Module

com.hurence.logisland:logisland-service-ip-to-geo-maxmind:1.1.1

Class

com.hurence.logisland.service.iptogeo.maxmind.MaxmindIpToGeoService

Tags

ip, service, geo, maxmind

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 88: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
maxmind.database.uri	Path to the Maxmind Geo Enrichment Database File.		null	false	false
maxmind.database.path	Local Path to the Maxmind Geo Enrichment Database File.		null	false	false
locale	Locale to use for geo information. Defaults to 'en'.		en	false	false
lookup.time	Should the additional lookup_micros field be returned or not.		false	false	false

Extra informations

No additional information is provided

MongoDBControllerService

Provides a controller service that wraps most of the functionality of the MongoDB driver.

Module

com.hurence.logisland:logisland-service-mongodb-client:1.1.1

Class

com.hurence.logisland.service.mongodb.MongoDBControllerService

Tags

mongo, mongodb, service

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values, and whether a property supports the Expression Language .

Table 89: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
mongo.uri	MongoURI, typically of the form: mongodb://host1[:port1][,host2[:port2],. . .]		null	false	**true**
mongo.db.name	The name of the database to use		null	false	**true**
mongo.collection.name	The name of the collection to use		null	false	**true**
batch.size	The preferred number of Records to setField to the database in a single transaction		1000	false	false
bulk.size	bulk size in MB		5	false	false
mongo.bulk.mode	Bulk mode (insert or upsert)	insert (Insert records whose key must be unique), upsert (Insert records if not already existing or update the record if already existing)	insert	false	false
flush.interval	flush interval in ms		500	false	false
mongo.write.concern	The write concern to use	ACKNOWLEDGED, UNACKNOWLEDGED, FSYNCED, JOURNALED, REPLICA_ACKNOWLEDGED, MAJORITY	ACKNOWLEDGED	false	false
mongo.bulk.upsert.condition	A custom condition for the bulk upsert (Filter for the bulkwrite). If not specified the standard condition is to match same id ('_id': data._id)		${'{ "_id" :"' + record_id + '"}'}	false	**true**

Extra informations

No additional information is provided

RedisKeyValueCacheService

A controller service for caching records by key value pair with LRU (last recently used) strategy. using LinkedHashMap

Module

com.hurence.logisland:logisland-service-redis:1.1.1

Class

com.hurence.logisland.redis.service.RedisKeyValueCacheService

Tags

cache, service, key, value, pair, redis

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values, and whether a property is considered "sensitive"..

Table 90: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
redis.mode	The type of Redis being communicated with - standalone, sentinel, or clustered.	standalone (A single standalone Redis instance.), sentinel (Redis Sentinel which provides high-availability. Described further at https://redis.io/topics/sentinel), cluster (Clustered Redis which provides sharding and replication. Described further at https://redis.io/topics/cluster-spec)	standalone	false	false
connection.string	The connection string for Redis. In a standalone instance this value will be of the form hostname:port. In a sentinel instance this value will be the comma-separated list of sentinels, such as host1:port1,host2:port2,host3:port3. In a clustered instance this value will be the comma-separated list of cluster masters, such as host1:port,host2:port,host3:port.		null	false	false
database.index	The database index to be used by connections created from this connection pool. See the databases property in redis.conf, by default databases 0-15 will be available.		0	false	false
communication.timeout	The timeout to use when attempting to communicate with Redis.		10 seconds	false	false
cluster.max.redirects	The maximum number of redirects that can be performed when clustered.		5	false	false
sentinel.master	The name of the sentinel master, require when Mode is set to Sentinel		null	false	false
password	The password used to authenticate to the Redis server. See the requirepass property in redis.conf.		null	**true**	false
pool.max.total	The maximum number of connections that can be allocated by the pool (checked out to clients, or idle awaiting checkout). A negative value indicates that there is no limit.		8	false	false
pool.max.idle	The maximum number of idle connections that can be held in the pool, or a negative value if there is no limit.		8	false	false
pool.min.idle	The target for the minimum number of idle connections to maintain in the pool. If the configured value of Min Idle is greater than the configured value for Max Idle, then the value of Max Idle will be used instead.		0	false	false
pool.block.when.exhausted	Whether or not clients should block and wait when trying to obtain a connection from the pool when the pool has no available connections. Setting this to false means an error will occur immediately when a client requests a connection and none are avail-	true, false	true	false	false

Extra informations

No additional information is provided

Solr_5_5_5_ClientService

Implementation of ElasticsearchClientService for Solr 5.5.5.

Module

com.hurence.logisland:logisland-service-solr_5_5_5-client:1.1.1

Class

com.hurence.logisland.service.solr.Solr_5_5_5_ClientService

Tags

solr, client

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 91: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
batch.size	The preferred number of Records to setField to the database in a single transaction		1000	false	false
bulk.size	bulk size in MB		5	false	false
solr.cloud	is slor cloud enabled		false	false	false
solr.collection	name of the collection to use		null	false	false
solr.connection.string	zookeeper quorum host1:2181,host2:2181 for solr cloud or http address of a solr core		localhost:8983/solr	false	false
solr.concurrent.requests	setConcurrentRequests		2	false	false
flush.interval	flush interval in ms		500	false	false
schema.update_timeout	Schema update timeout interval in s		15	false	false

Extra informations

No additional information is provided

Solr_6_4_2_ChronixClientService

Implementation of ChronixClientService for Solr 6 4 2

Module

com.hurence.logisland:logisland-service-solr_chronix_6.4.2-client:1.1.1

Class

com.hurence.logisland.service.solr.Solr_6_4_2_ChronixClientService

Tags

solr, client

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 92: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
batch.size	The preferred number of Records to setField to the database in a single transaction		1000	false	false
solr.cloud	is slor cloud enabled		false	false	false
solr.collection	name of the collection to use		null	false	false
solr.connection.string	zookeeper quorum host1:2181,host2:2181 for solr cloud or http address of a solr core		localhost:8983/solr	false	false
flush.interval	flush interval in ms		500	false	false
group.by	The field the chunk should be grouped by			false	false

Extra informations

No additional information is provided

Solr_6_6_2_ClientService

Implementation of ElasticsearchClientService for Solr 5.5.5.

Module

com.hurence.logisland:logisland-service-solr_6_6_2-client:1.1.1

Class

com.hurence.logisland.service.solr.Solr_6_6_2_ClientService

Tags

solr, client

Properties

In the list below, the names of required properties appear in **bold**. Any other properties (not in bold) are considered optional. The table also indicates any default values.

Table 93: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
batch.size	The preferred number of Records to setField to the database in a single transaction		1000	false	false
bulk.size	bulk size in MB		5	false	false
solr.cloud	is slor cloud enabled		false	false	false
solr.collection	name of the collection to use		null	false	false
solr.connection.string	zookeeper quorum host1:2181,host2:2181 for solr cloud or http address of a solr core		localhost:8983/solr	false	false
solr.concurrent.requests	setConcurrentRequests		2	false	false
flush.interval	flush interval in ms		500	false	false
schema.update_timeout	Schema update timeout interval in s		15	false	false

Extra informations

No additional information is provided

1.4.2 Dynamic properties

Overview

You use components to run jobs in logisland that manipulate records. Those components use properties that you specify in the job configuration file. Some of them are defined in advance by the component's developer. They got a name and you have to use it to define these properties. We call those properties *static properties*.

Some components support dynamic *properties*. When this is the case, any properties specified in job conf for this component that is not a static property will be used as a dynamic property instead of throwing an error for a bad configuration.

In this section we will talk about those properties and how you can use them.

Structure of a dynamic properties

Dynamic properties are really just like static properties but build on the fly. It allow to use both the name and the value of the property by the developer. For example instead of specifying :

```
record.name: myName
record.value: myValue
```

You could specify :

```
myName: myValue
```

The advantage is that you can have any number of dynamic property whereas you have to specify in advance all static properties…

Usage of a dynamic properties

You can check the documentation of *AddFields* processor that we will use in those example.

Adding a field which is concatenation of two others using '_' as joining string

set those dynamic properties in AddFields processor :

- concat2fields : value1
- my_countries : 3
- my_countries.type : INT

Then records processed by this processor would have 2 more fields out of this processors:

- field 'concat2fields' of type String with value 'value1'
- field 'my_countries' of type Int with value '3'

By default if no type is specified by a dynamic property it use a type of String or the same type as old value if field already existed and you choose an overwrite policy.

See *AddFields* processor doc fore more information.

Conclusion

As you can see dynamic properties are very flexible but it's usage is very dependent of the implementation of the component's developer.

1.4.3 Expression Language

Overview

All data in Logisland is represented by an abstraction called a Record. Those records contains fields of different types.

You use components to run jobs in logisland that manipulate those records. Those components use properties that you specify in the job configuration file. Some of them support the expression language (EL). In this section we will talk about those properties and how you can use them.

Structure of a Logisland Expression

The Logisland expression Language always begins with the start delimiter *${* and ends with the end delimiter *}*. Between the start and end delimiters is the text of the expression itself. In its most basic form, the expression can consist of just a record field name. For example, *${name}* will return the value of the field *name* of the record used.

The use of the property depends on the implementation of the components ! Indeed it is the component that decide to evaluate your Logisland expression with which Record.

For example the AddField processor use Logisland expression in its dynamic properties.

- The key representing the name of the field to add.
- The value can be a Logisland expression that will be used to calculate the value of the new field. In this expression you can use fields value of the current Record because it is passed as context of the Logisland expression by this processor.

So be sure to carefully read description of the properties to understand how it will be evaluated and for what purpose.

We are currently using the **mvel** language which you can check documentation here.

Note: If you want to be able to use another ScriptEngine than mvel (javascript for example). You can open an issue to ask this feature. Feel free to make a Pull request as well to implement this new feature.

We have implemented some example as unit test as well if you want to check in the code source, the class is **com.hurence.logisland.component.TestInterpretedPropertyValueWithMvelEngine** in the module **com.hurence.logisland:logisland-api**.

Otherwise we will show you some simple examples using the AddField processor in next Section.

Usage of a Logisland Expression

You can check the documentation of *AddFields* processor that we will use in those example.

Adding a field which is concatenation of two others using '_' as joining string

set those dynamic properties in AddFields processor :

- concat2fields : ${field1 + "_" + field2}
- my_countries : ${["france", "allemagne"]}
- my_countries.type : array
- my_employees_by_countries : ${["france" : 100, "allemagne" : 50]}
- my_employees_by_countries.type : map

Then if in input of this processor there is records with fields : field1=value1 and field2=value2, it would have 3 more fields once out of this processor:

- field 'concat2fields' of type String with value 'value1_value2'
- field 'my_countries' of type Array containing values 'france' and 'allemagne'
- field 'my_employees_by_countries' of type Map with key value pairs "france" : 100 and "allemagne" : 50

By default if no type is specified by a dynamic property it use a type of String or the same type as old value if field already existed and you choose an overwrite policy.

See *AddFields* processor doc for more information.

Conclusion

As you can see the language expression is very flexible but it's usage is very dependent of the implementation of the component's developer.

1.5 Developer Documentation

Contents:

1.5.1 Developer Guide

This document summarizes information relevant to logisland committers and contributors. It includes information about the development processes and policies as well as the tools we use to facilitate those.

Workflows

This section explains how to perform common activities such as reporting a bug or merging a pull request.

Internal dev (aka logisland team)

We're using GitFlow for github so read carefully the docs : https://datasift.github.io/gitflow/GitFlowForGitHub.html

Coding Guidelines

Basic

1. Avoid cryptic abbreviations. Single letter variable names are fine in very short methods with few variables, otherwise make them informative.
2. Clear code is preferable to comments. When possible make your naming so good you don't need comments. When that isn't possible comments should be thought of as mandatory, write them to be read.
3. Logging, configuration, and public APIs are our "UI". Make them pretty, consistent, and usable.
4. Maximum line length is 130.
5. Don't leave TODOs in the code or FIXMEs if you can help it. Don't leave println statements in the code. TODOs should be filed as github issues.
6. User documentation should be considered a part of any user-facing the feature, just like unit tests. Example REST apis should've accompanying documentation.
7. Tests should never rely on timing in order to pass.
8. Every unit test should leave no side effects, i.e., any test dependencies should be set during setup and clean during tear down.

Java

1. Apache license headers. Make sure you have Apache License headers in your files.
2. Tabs vs. spaces. We are using 4 spaces for indentation, not tabs.
3. Blocks. All statements after if, for, while, do, … must always be encapsulated in a block with curly braces (even if the block contains one statement):

```
for (...) {
    ...
}
```

4. No wildcard imports.
5. No unused imports. Remove all unused imports.
6. No raw types. Do not use raw generic types, unless strictly necessary (sometime necessary for signature matches, arrays).
7. Suppress warnings. Add annotations to suppress warnings, if they cannot be avoided (such as "unchecked", or "serial").
8. Comments. Add JavaDocs to public methods or inherit them by not adding any comments to the methods.
9. logger instance should be upper case LOG.
10. When in doubt refer to existing code or Java Coding Style except line breaking, which is described above.

Logging

1. Please take the time to assess the logs when making a change to ensure that the important things are getting logged and there is no junk there.
2. There are six levels of logging TRACE, DEBUG, INFO, WARN, ERROR, and FATAL, they should be used as follows.

   2.1. INFO is the level you should assume the software will be run in. INFO messages are things which are not bad but which the user will definitely want to know about every time they occur.

   2.2 TRACE and DEBUG are both things you turn on when something is wrong and you want to figure out what is going on. DEBUG should not be so fine grained that it will seriously effect the performance of the server. TRACE can be anything. Both DEBUG and TRACE statements should be wrapped in an if(logger.isDebugEnabled) if an expensive computation in the argument list of log method call.

   2.3. WARN and ERROR indicate something that is bad. Use WARN if you aren't totally sure it is bad, and ERROR if you are.

   2.4. Use FATAL only right before calling System.exit().

3. Logging statements should be complete sentences with proper capitalization that are written to be read by a person not necessarily familiar with the source code.
4. **String appending using StringBuilders should not be used for building log messages.** Formatting should be used. For ex: LOG.debug("Loaded class [{}] from jar [{}]", className, jarFile);
5. In Logisland class implementing ConfigurableComponent use **getLogger** method to log. Most of components in Logisland are ConfigurableComponent.

TimeZone in Tests

Your environment jdk can be different than travis ones. Be aware that there is changes on TimeZone objects between different version of jdk… Even between 8.x.x versions. For example TimeZone "America/Cancun" may not give the same date in your environment than in travis one…

Contribute code

Create a pull request

Pull requests should be done against the read-only git repository at https://github.com/hurence/logisland.

Take a look at Creating a pull request. In a nutshell you need to:

1. Fork the Logisland GitHub repository at https://github.com/hurence/logisland to your personal GitHub account. See Fork a repo for detailed instructions.
2. Commit any changes to your fork.
3. Send a pull request to the Logisland GitHub repository that you forked in step 1. If your pull request is related to an existing IoTaS github issue ticket – for instance, because you reported a bug report via github issue earlier – then prefix the title of your pull request with the corresponding github issue ticket number (e.g. *IOT-123: …*).

You may want to read Syncing a fork for instructions on how to keep your fork up to date with the latest changes of the upstream *Streams* repository.

We are using gitflow to have standard way of starting features, hotfixes and releases. You can check documentation about gitflow here.

Git Commit Messages Format

The Git commit messages must be standardized as follows:

LOGISLAND-XXX: Title matching exactly the github issue Summary (title)

- An optional, bulleted (+, -, ., *), summary of the contents of
- the patch. The goal is not to describe the contents of every file,
- but rather give a quick overview of the main functional areas
- addressed by the patch.

The text immediately following the github issue number (LOGISLAND-XXX:) must be an exact transcription of the github issue summary (title), not the a summary of the contents of the patch.

If the github issue summary does not accurately describe what the patch is addressing, the github issue summary must be modified, and then copied to the Git commit message.

A summary with the contents of the patch is optional but strongly encouraged if the patch is large and/or the github issue title is not expressive enough to describe what the patch is doing. This text must be bulleted using one of the following bullet points (+, -, .,). There must be at last a 1 space indent before the bullet char, and exactly one space between bullet char and the first letter of the text. Bullets are not optional, but *required**.

Develop components

You can find help on these topics here :

- *Processors*
- *Services*
- *Connectors*
- *Streams*
- *Engines*

Build the code and run the tests

Prerequisites

First of all you need to make sure you are using maven 3.2.5 or higher and JDK 1.8 or higher.

Building

The following commands must be run from the top-level directory.

```
mvn install
```

Would build a light version of logisland with only common processors installed.

```
mvn install -Pfull
```

Would build a heavy version of logisland with all logisland plugins installed.

If you wish to skip the unit tests you can do this by adding *-DskipTests* to the command line.

If you wish to add all the plugins to the build you can do this by adding *-Pfull* to the command line.

Release to maven repositories

to release artifacts (if you're allowed to), follow this guide release to OSS Sonatype with maven

```
./update-version.sh -o 1.1.1 -n 14.4
 mvn license:format
 mvn test
 mvn -DperformRelease=true clean deploy -Pfull
 mvn versions:commit
```

follow the staging procedure in oss.sonatype.org or read Sonatype book

go to oss.sonatype.org to release manually the artifact

Publish release assets to github

please refer to https://developer.github.com/v3/repos/releases

curl -XPOST https://uploads.github.com/repos/Hurence/logisland/releases/v1.1.1/assets?name=logisland-1.1.1-bin.tar.gz -v –data-binary @logisland-assembly/target/logisland-1.1.1-bin.tar.gz –user oalam -H 'Content-Type: application/gzip'

Publish Docker image

Building the image

```
# build logisland
mvn install -DskipTests -Pdocker -Pfull

# verify image build
docker images
```

then login and push the latest image

```
docker login
docker push hurence/logisland
```

Publish artifact to github

Tag the release + upload latest tgz

1.5.2 Components

Contents:

Processors

This document summarizes information relevant to develop a logisland Processor.

Interfaces

A Logisland processor **must** implements the *com.hurence.logisland.processor.Processor* Interface.

Base of processors

For making easier the processor implementation we advise you to extends *com.hurence.logisland.processor.AbstractProcessor*. This way most of the work is already done for you and you will benefit from future improvements.

Note: If you do not extend *com.hurence.logisland.processor.AbstractProcessor*, there is several point to be carefull with. Read following section

---

Not using AbstractProcessor

The documentation for this part is not available yet. If you want to borrow this path, feel free to open an issue and/or talk with us on gitter about it so we can advise you on the important point to be carefull with.

Important Object Notions

Here we will present you the objects that you will probably have to use.

PropertyDescriptor

To implement a Processor you will have to add *PropertyDescriptors* to your processor. The standard way to do this is to add them as static variables of your Processor Classes. Then they will be used in the processor's methods.

```
private static final AllowableValue OVERWRITE_EXISTING =
        new AllowableValue("overwrite_existing", "overwrite existing field", "if␣
↪field already exist");

private static final AllowableValue KEEP_OLD_FIELD =
        new AllowableValue("keep_only_old_field", "keep only old field value", "keep␣
↪only old field");

private static final PropertyDescriptor CONFLICT_RESOLUTION_POLICY = new␣
↪PropertyDescriptor.Builder()
        .name("conflict.resolution.policy")
        .description("What to do when a field with the same name already exists ?")
        .required(false)
        .defaultValue(KEEP_OLD_FIELD.getValue())
        .allowableValues(OVERWRITE_EXISTING, KEEP_OLD_FIELD)
        .build();
```

ProcessContext

See *ProcessContext* for more information.

Record

See *Record* for more information.

Important methods

Here we will present you the methods that you will probably have to implement or override.

getSupportedPropertyDescriptors

This method is required by *AbstractProcessor*, it is used to verify that user configuration for your processor is correct. This method should return the list of *PropertyDescriptor* that your processor supports. Be sure to add any Descriptor provided by parents if any using super.getSupportedPropertyDescriptors() methods.

Here an example with only one supported property

```
@Override
public List<PropertyDescriptor> getSupportedPropertyDescriptors() {
    return Collections.singletonList(CONFLICT_RESOLUTION_POLICY);
}
```

getSupportedDynamicPropertyDescriptor

This method is required by *AbstractProcessor* and is not required if you do not support dynamic properties. Otherwise create here yours dynamic properties descriptions.

This property descriptor will be used to validate any user key configuration that is not in the list of supported properties. If you return null, it is considered that the property name is not a valid dynamic property.

You can have several type of supported dynamic properties if you want as in the example below. Go there to learn more about *Dynamic properties*.

```
@Override
protected PropertyDescriptor getSupportedDynamicPropertyDescriptor(final String␣
↪propertyDescriptorName) {
    if (propertyDescriptorName.endsWith(DYNAMIC_PROPS_TYPE_SUFFIX)) {
        return new PropertyDescriptor.Builder()
                .name(propertyDescriptorName)
                .expressionLanguageSupported(false)
                .addValidator(new StandardValidators.EnumValidator(FieldType.class))
                .allowableValues(FieldType.values())
                .defaultValue(FieldType.STRING.getName().toUpperCase())
                .required(false)
                .dynamic(true)
                .build();
    }
    if (propertyDescriptorName.endsWith(DYNAMIC_PROPS_NAME_SUFFIX)) {
        return new PropertyDescriptor.Builder()
                .name(propertyDescriptorName)
                .expressionLanguageSupported(true)
                .addValidator(StandardValidators.NON_EMPTY_VALIDATOR)
                .required(false)
                .dynamic(true)
                .build();
    }
    return new PropertyDescriptor.Builder()
            .name(propertyDescriptorName)
            .expressionLanguageSupported(true)
            .addValidator(StandardValidators.NON_EMPTY_VALIDATOR)
            .required(false)
            .dynamic(true)
            .build();
}
```

init

This method should contain all initialization variables of your processor. It is called at least once before processing records. So you can do quite heavy initialization here. But you can also use controller services as property for sharing heavy components between different processors. You should always use a controller service for interacting with extern sources. LINK TODO services as property

Note: It is required to use at the start of the method the super.init method ! (It does some core initializing).

Example :

```
@Override
public void init(ProcessContext context) {
    super.init(context);
    initDynamicProperties(context);
    this.conflictPolicy = context.getPropertyValue(CONFLICT_RESOLUTION_POLICY).
↪asString();
}
```

process

This method is the core of the processor. This is this method that interact with Logisland Record. It either modify them, use them, filter them or whatever you want. Below an example that is just adding a new field to each record (this is obviously not a real processor).

```
@Override
public Collection<Record> process(ProcessContext context, Collection<Record> records)
↪{
    for (Record record : records) {
        record.setStringField("my_first_processor_impl", "Hello world !");
    }
    return records;
}
```

Add documentation about the processor

The logisland-documentation module contains logisland documentation. See *Documentation Guide* for more information. Some part of the documentation is automatically generated at build time. It uses annotation in logisland code.

In our case of a processors you have to add those *Annotation of ConfigurableComponent*.

Also you need to add your module dependency in documentation module like explained here *Add a ConfigurableComponent in the auto generate documentation*.

Add your processor as a logisland plugin

Unless the new processor you implemented is already in an existing logisland module you will have to do those two steps below.

Make your module a logisland plugin container

You will have to build your module as a plugin in two steps : * Using **spring-boot-maven-plugin** that will build a fat jar of your module. * Using our custom plugin **logisland-maven-plugin** that will modify the manifest of the jar so that logisland get some meta information.

You just have to add this code in the *pom.xml* of your module.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
```

(continues on next page)

(continued from previous page)

```
            <artifactId>spring-boot-maven-plugin</artifactId>
            <executions>
                <execution>
                    <phase>package</phase>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>com.hurence.logisland</groupId>
            <artifactId>logisland-maven-plugin</artifactId>
            <executions>
                <execution>
                    <phase>package</phase>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Add your module in tar gz assembly

You will have to add your module as a dependency in the **logisland-assembly** module. Add it in **full** maven profile so that it is automatically Added to logisland jar when building with -Pfull option.

```
<profile>
    <id>full</id>
    <activation>
        <activeByDefault>false</activeByDefault>
    </activation>
    <dependencies>
        ...
        <dependency>
            <groupId>com.hurence.logisland</groupId>
            <artifactId>YOUR_MODULE_NAME</artifactId>
            <version>${project.version}</version>
        </dependency>
     </dependencies>
</profile>
```

Services

This document summarizes information relevant to develop a logisland controller service.

Interfaces

A Logisland controller service **must** implements the *com.hurence.logisland.controller.ControllerService* Interface.

Base of controller services

For making easier the controller service implementation we advise you to extends *com.hurence.logisland.controller.AbstractControllerService*. This way most of the work is already done for you and you will benefit from future improvements.

Note: If you do not extend *com.hurence.logisland.controller.AbstractControllerService*, there is several point to be carefull with. Read following section

Not using AbstractControllerService

The documentation for this part is not available yet. If you want to borrow this path, feel free to open an issue and/or talk with us on gitter about it so we can advise you on the important point to be carefull with.

Important Object Notions

Here we will present you the objects that you will probably have to use.

PropertyDescriptor

To implement a Processor you will have to add *PropertyDescriptors* to your processor. The standard way to do this is to add them as static variables of your Processor Classes. Then they will be used in the processor's methods.

```
private static final AllowableValue OVERWRITE_EXISTING =
        new AllowableValue("overwrite_existing", "overwrite existing field", "if␣
↪field already exist");

private static final AllowableValue KEEP_OLD_FIELD =
        new AllowableValue("keep_only_old_field", "keep only old field value", "keep␣
↪only old field");

private static final PropertyDescriptor CONFLICT_RESOLUTION_POLICY = new␣
↪PropertyDescriptor.Builder()
        .name("conflict.resolution.policy")
        .description("What to do when a field with the same name already exists ?")
        .required(false)
        .defaultValue(KEEP_OLD_FIELD.getValue())
        .allowableValues(OVERWRITE_EXISTING, KEEP_OLD_FIELD)
        .build();
```

ControllerServiceInitializationContext

See *ControllerServiceInitializationContext* for more information.

Record

See *Record* for more information.

Important methods

Here we will present you the methods that you will probably have to implement or override.

getSupportedPropertyDescriptors

This method is required by *AbstractProcessor*, it is used to verify that user configuration for your processor is correct. This method should return the list of *PropertyDescriptor* that your processor supports. Be sure to add any Descriptor provided by parents if any using super.getSupportedPropertyDescriptors() methods.

Here an example with only one supported property

```
@Override
public List<PropertyDescriptor> getSupportedPropertyDescriptors() {
    return Collections.singletonList(CONFLICT_RESOLUTION_POLICY);
}
```

getSupportedDynamicPropertyDescriptor

This method is required by *AbstractProcessor* and is not required if you do not support dynamic properties. Otherwise create here yours dynamic properties descriptions.

This property descriptor will be used to validate any user key configuration that is not in the list of supported properties. If you return null, it is considered that the property name is not a valid dynamic property.

You can have several type of supported dynamic properties if you want as in the example below.

```
@Override
protected PropertyDescriptor getSupportedDynamicPropertyDescriptor(final String␣
↪propertyDescriptorName) {
    if (propertyDescriptorName.endsWith(DYNAMIC_PROPS_TYPE_SUFFIX)) {
        return new PropertyDescriptor.Builder()
                .name(propertyDescriptorName)
                .expressionLanguageSupported(false)
                .addValidator(new StandardValidators.EnumValidator(FieldType.class))
                .allowableValues(FieldType.values())
                .defaultValue(FieldType.STRING.getName().toUpperCase())
                .required(false)
                .dynamic(true)
                .build();
    }
    if (propertyDescriptorName.endsWith(DYNAMIC_PROPS_NAME_SUFFIX)) {
        return new PropertyDescriptor.Builder()
                .name(propertyDescriptorName)
                .expressionLanguageSupported(true)
                .addValidator(StandardValidators.NON_EMPTY_VALIDATOR)
                .required(false)
                .dynamic(true)
                .build();
    }
    return new PropertyDescriptor.Builder()
            .name(propertyDescriptorName)
            .expressionLanguageSupported(true)
            .addValidator(StandardValidators.NON_EMPTY_VALIDATOR)
            .required(false)
            .dynamic(true)
            .build();
}
```

init

This method should contain all initialization variables of your controller service. It is called at least once before you can use it. So you can do quite heavy initialization here. You should instantiate connection with your service you want to controll so that user of this controller can request the service without having to etablish the contact first. Note that you should handle case where service session time out or is closed for any reason. In this case, your service should be able to establish a connection again automatically when needed, the framework will not handle this for you.

Note: It is required to use at the start of the method the super.init method ! (It does some core initializing).

Example :

```
@Override
public void init(ProcessContext context) {
    super.init(context);
    this.serviceClient = buildServiceClient();
}
```

Other methods defined in an API

Services should implement an interface defining an API. For exemple *com.hurence.logisland.service.datastore.DatastoreClientService* represents a generic api for any datastore. The advantage of using this is that a processor can work with all services implementing this interface if it is declared as a *DatastoreClientService* instance.

For example the BulkPut processor use a *DatastoreClientService* as input so it can inject in using any service implementing *DatastoreClientService*. So it can inject potentially in any database.

You can create a special module to create a desired interface that you want your service to implement. This way other services would be able to use it as well.

Here a method for example defined in *DatastoreClientService*.

```
/**
 * Drop the specified collection/index/table/bucket.
 * Specify namespace as dotted notation like in `global.users`
 */
void dropCollection(String name)throws DatastoreClientServiceException;
```

Add documentation about the service

The logisland-documentation module contains logisland documentation. See *Documentation Guide* for more information. Some part of the documentation is automatically generated at build time. It uses annotation in logisland code.

In our case of a service you have to add those *Annotation of ConfigurableComponent*.

Also you need to add your module dependency in documentation module like explained here *Add a ConfigurableComponent in the auto generate documentation*.

Add your service as a logisland plugin

Unless the new service you implemented is already in an existing logisland module you will have to do those two steps below.

Make your module a logisland plugin container

You will have to build your module as a plugin in two steps : * Using **spring-boot-maven-plugin** that will build a fat jar of your module. * Using our custom plugin **logisland-maven-plugin** that will modify the manifest of the jar so that logisland get some meta information.

You just have to add this code in the *pom.xml* of your module.

```
<build>
    <plugins>
        <plugin>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-maven-plugin</artifactId>
            <executions>
                <execution>
                    <phase>package</phase>
                </execution>
            </executions>
        </plugin>
        <plugin>
            <groupId>com.hurence.logisland</groupId>
            <artifactId>logisland-maven-plugin</artifactId>
            <executions>
                <execution>
                    <phase>package</phase>
                </execution>
            </executions>
        </plugin>
    </plugins>
</build>
```

Add your module in tar gz assembly

You will have to add your module as a dependency in the **logisland-assembly** module. Add it in **full** maven profile so that it is automatically Added to logisland jar when building with -Pfull option.

```
<profile>
    <id>full</id>
    <activation>
        <activeByDefault>false</activeByDefault>
    </activation>
    <dependencies>
        ...
        <dependency>
            <groupId>com.hurence.logisland</groupId>
            <artifactId>YOUR_MODULE_NAME</artifactId>
            <version>${project.version}</version>
        </dependency>
     </dependencies>
</profile>
```

Connectors

This documentation is not available yet but you can check on existing examples in **logisland-connectors** module. All connectors should be implemented in this module.

Streams

This documentation is not available yet.

Engines

This documentation is not available yet but you can check on existing examples in **logisland-engines** module. All engines should be implemented in this module.

Add your engine in the assembly

You'll have to add your engine in the assembly in module **logisland-assembly**. Add it in profile *full* of pom.

Add your engine in the documentation

To add docs about your engine you can check *Add a ConfigurableComponent in the auto generate documentation*.

1.5.3 Object Model

Contents:

Record

This documentation is not available yet.

PropertyDescriptors

This document summarizes information relevant for using *com.hurence.logisland.component.PropertyDescriptor* which is part of Logisland api and is used throughout Logisland.

Purpose

This object is used to describe a property that users can used in job configuration when using a component. In a component, you will describe those properties using *com.hurence.logisland.component.PropertyDescriptor*.

Builder

You create a PropertyDescriptor using the builder this way :

```
private static final PropertyDescriptor CONFLICT_RESOLUTION_POLICY = new␣
↪PropertyDescriptor.Builder()
        .name("conflict.resolution.policy")
        .description("What to do when a field with the same name already exists ?")
        .required(false)
        .defaultValue(KEEP_OLD_FIELD.getValue())
        .allowableValues("value1", "value2")
        .expressionLanguageSupported(false)
```

(continues on next page)

(continued from previous page)

```
        .addValidator(StandardValidators.NON_EMPTY_VALIDATOR)
        .sensitive(true)
        .build();
```

You can use

```
.identifiesControllerService(ElasticsearchClientService.class)
```

When you want a property to be used to reference a *Services*

properties

Here we will describe each element you can set to a PropertyDescriptor.

name

This is the string that will be used by the client in the yaml conf file.

description

This is used in the auto generated documentation of components to describe properties.

required

If this property is mandatory or not

defaultValue

Default value if any

allowableValues

To specify a specific set of authorized values (Add a constraint on the expected value of the property).

expressionLanguageSupported

Specify if *Expression Language* is supported for this property or not.

addValidator

Add given validator to the property (Add a constraint on the expected value of the property).

sensitive

Specifies if the property contain sensitive information or not.

ProcessContext

This documentation is not available yet.

ControllerServiceInitializationContext

You can use it as a ProcessContext. See *ProcessContext* for more information.

1.5.4 Documentation

Contents:

Documentation Guide

Here we will describe you how the doc in logisland is build and how to modify it.

Introduction

The documentation in logisland is handled by **logisland-documentation** module which build the automated part of the doc. That is why you should correctly annotate your components when developing.

All *.rst* files in this module are used to build the doc. We use *Sphinx* and *https://readthedocs.org/* for that.

So in order to change the documentation you must change these files. But do not modify files that are automatically generated ! The auto generated files are in the **components** directory. (Except for the index files)

Modify the hard coded documentation

We use ReStructuredText format for writing the doc. Then we generate html pages with Sphinx. So you should be familiarized with this if you wants to do some advanced docs. Otherwise you can just modify files for minor changes.

Modify auto generated documentation

To generate generated documentation, just install the module

```
cd logisland-documentation
mvn install -DskipTests
```

By default, it will build all components doc. At the moment you must commit any modification to those files in order for it to appear on online documentation.

Annotation of ConfigurableComponent

The auto generated documentation use annotation in code. So be sure to add below anotations in every Component you develop.

Tags

It should be a list of words. So a user can rapidly filter out components. This is not currently a feature implemented but you should still mention those tags for future use.

CapabilityDescription

This tag is used to describe the components. It should be in *.rst* format.

DynamicProperty

This is used when your components support *Dynamic properties*. You specify each property to explain how it will be used.

For example :

```
@DynamicProperty(name = "field to add",
    supportsExpressionLanguage = false,
    value = "default value",
    description = "Add a field to the record with the default value")
```

Means :

- that the name of the property will be the name of a new field created in record.
- that the value specified can support or not expression language.
- that the value will be the used as value for the new property.
- you can add a general description as well.

DynamicProperties

This is used when your components support *Dynamic properties*. You use thi annotation instead of **DynamicProperty** if your components support different type of *Dynamic properties*.

You specify a list of annotation @DynamicProperty, one by type you support.

For example :

```
@DynamicProperties(value = {
    @DynamicProperty(name = "Name of the field to add",
            supportsExpressionLanguage = true,
            value = "Value of the field to add",
            description = "Add a field to the record with the specified value.␣
↪Expression language can be used." +
                    "You can not add a field that end with '.type' as this suffix is␣
↪used to specify the type of fields to add",
            nameForDoc = "fakeField"),
    @DynamicProperty(name = "Name of the field to add with the suffix '"+ AddFields.
↪DYNAMIC_PROPS_TYPE_SUFFIX +"'",
            supportsExpressionLanguage = false,
            value = "Type of the field to add",
            description = "Add a field to the record with the specified type. These␣
↪properties are only used if a correspondant property without" +
```

(continues on next page)

(continued from previous page)

```
                    " the suffix '"+ AddFields.DYNAMIC_PROPS_TYPE_SUFFIX +"' is␣
↪already defined. If this property is not defined, default type for adding fields is␣
↪String." +
                    "You can only use Logisland predefined type fields.",
            nameForDoc = "fakeField" + AddFields.DYNAMIC_PROPS_TYPE_SUFFIX),
    @DynamicProperty(name = "Name of the field to add with the suffix '" + AddFields.
↪DYNAMIC_PROPS_NAME_SUFFIX + "'",
            supportsExpressionLanguage = true,
            value = "Name of the field to add using expression language",
            description = "Add a field to the record with the specified name (which␣
↪is evaluated using expression language). " +
                    "These properties are only used if a correspondant property␣
↪without" +
                    " the suffix '" + AddFields.DYNAMIC_PROPS_NAME_SUFFIX + "' is␣
↪already defined. If this property is not defined, " +
                    "the name of the field to add is the key of the first dynamic␣
↪property (which is the main and only required dynamic property).",
            nameForDoc = "fakeField" + AddFields.DYNAMIC_PROPS_NAME_SUFFIX)
})
```

ConfigurableComponent Method used

Each components is instantiated as a ConfigurableComponent, then we use the method :

```
List<PropertyDescriptor> getPropertyDescriptors();
```

To add information about evey supported property by the component.

Add a ConfigurableComponent in the auto generate documentation

We have a java job **DocGenerator** which generate documentation about ConfigurableComponent in the classpath of the JVM. Here the usage of the job :

```
usage: com.hurence.logisland.documentation.DocGenerator [-a] [-d <arg>] [-f <arg>] [-
↪h]
 -a,--append              Whether to append or replace file
 -d,--doc-dir <arg>       dir to generate documentation
 -f,--file-name <arg>     file name to generate documentation about components in␣
↪classpath
 -h,--help                Print this help.
```

In the pom of the module we use this job several time with different parameters using the *exec-maven-plugin*. We launch it several time with different classpath to avoid conflict issue with different version of libraries. If you want your components documentation to be generated you have to add it in one of those executions. If you are dealing with dependencies problem you can create a completely new execution.

For processors and services this should not be too hard as they are packaged as plugin.

For example :

```
<execution>
    <id>generate doc services</id>
    <phase>install</phase>
    <configuration>
```

(continues on next page)

(continued from previous page)

```
        <executable>java</executable>
        <arguments>
            <argument>-classpath</argument>
            <classpath>
                <dependency>commons-cli:commons-cli</dependency>
                <dependency>commons-io:commons-io</dependency>
                <dependency>org.apache.commons:commons-lang3</dependency>
                <dependency>org.slf4j:slf4j-simple</dependency>
                <dependency>org.slf4j:slf4j-api</dependency>
                <dependency>com.hurence.logisland:logisland-api</dependency>
                <!--<dependency>com.fasterxml.jackson.core:jackson-core</dependency>--
↪>
                <!--<dependency>com.fasterxml.jackson.core:jackson-databind</
↪dependency>-->
                <dependency>com.hurence.logisland:logisland-utils</dependency>
                <dependency>com.hurence.logisland:logisland-api</dependency>
                <dependency>com.hurence.logisland:logisland-plugin-support</
↪dependency>
                <!--Needed dependencies by logisland-plugin-support-->
                <dependency>cglib:cglib-nodep</dependency>
                <dependency>org.springframework.boot:spring-boot-loader</dependency>
                <!--SERVICE-->
                <dependency>com.hurence.logisland:logisland-service-hbase_1_1_2-client
↪</dependency>
                <dependency>com.hurence.logisland:logisland-service-elasticsearch_2_4_
↪0-client</dependency>
                <dependency>com.hurence.logisland:logisland-service-elasticsearch_5_4_
↪0-client</dependency>
                <dependency>com.hurence.logisland:logisland-service-redis</dependency>
                <dependency>com.hurence.logisland:logisland-service-mongodb-client</
↪dependency>
                <dependency>com.hurence.logisland:logisland-service-cassandra-client</
↪dependency>
                <dependency>com.hurence.logisland:logisland-service-solr_5_5_5-client
↪</dependency>
                <dependency>com.hurence.logisland:logisland-service-solr_6_6_2-client
↪</dependency>
                <dependency>com.hurence.logisland:logisland-service-solr_chronix_6.4.
↪2-client</dependency>
            </classpath>
            <argument>com.hurence.logisland.documentation.DocGenerator</argument>
            <argument>-d</argument>
            <argument>${generate-components-dir}</argument>
            <argument>-f</argument>
            <argument>services</argument>
        </arguments>
    </configuration>
    <goals>
        <goal>exec</goal>
    </goals>
</execution>
```

Will generate documentation for all service specified. You can just add your module in there. Then generate docs with

```
mvn install -DskipTests
```

1.6 Plugins

In this chapter we will present you how the logisland plugins architecture and how to manage them

Table of Contents

- *Plugins*
  - *What's a plugin?*
  - *How a plugin is packaged*
  - *How about naming?*
  - *Getting started*
    * *List all components*
    * *Install a component*
    * *Remove a component*
  - *Which module contains my component?*
  - *How about the distribution?*

1.6.1 What's a plugin?

A logisland plugin is anything can bring a functionality to logisland.

It can be:

- A processor
- A controller service
- A connector

1.6.2 How a plugin is packaged

A plugin is a jar in the logisland lib folder containing a special manifest giving some information about:

- Exported components
- Versions
- Classloading rules

As well a plugin jar contains every additional dependency is required to make it work with logisland, that ensures the portability with a single file.

1.6.3 How about naming?

When talking about a plugin we talk about an *artifact*.

Logisland uses the same maven naming convention (groupId, artifactId, version) to locate a plugin. This ensure a component to be unique and versioned.

1.6.4 Getting started

Everything about plugins is managed through the *components.sh* client utility (in the bin folder along with logisland.sh command).

Let's see the main actions you can do with

List all components

Simply use the *-l* option.

```
bin/components.sh -l

Listing details for 1 installed modules.
Artifact: com.hurence.logisland:logisland-processor-common:1.0.0
Name: Common processors bundle
Version: 1.0.0
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
      com.hurence.logisland.processor.SelectDistinctRecords
      com.hurence.logisland.processor.SendMail
      com.hurence.logisland.processor.SplitField
      com.hurence.logisland.processor.SplitText
      com.hurence.logisland.processor.SplitTextMultiline
      com.hurence.logisland.processor.SplitTextWithProperties
      com.hurence.logisland.processor.alerting.CheckAlerts
      com.hurence.logisland.processor.alerting.CheckThresholds
      com.hurence.logisland.processor.alerting.ComputeTags
      com.hurence.logisland.processor.datastore.BulkPut
      com.hurence.logisland.processor.datastore.EnrichRecords
      com.hurence.logisland.processor.datastore.MultiGet
```

This above is the logisland common processor modules bundled by default in the distribution.

As we can see the command line tell us some nice information:

- The file name
- The version
- The components it provides

Install a component

You can install two things of components:

- A logisland plugin

- A kafka connect source or sink (more information on connectors section)

The generic syntax for both is:

```
bin/components.sh -i <plugin_artifact>
```

For instance, if we want to install elasticsearch 5.4 controller service we are going to install the related module called *com.hurence.logisland:logisland-service-elasticsearch_5_4_0-client:<logisland_version>*

```
bin/components.sh -i com.hurence.logisland:logisland-service-elasticsearch_5_4_0-
↪client:1.0.0

Downloading dependencies. Please hold on...

Found logisland plugin Elasticsearch 5.4.0 Service Plugin version 1.1.1

It will provide:
      com.hurence.logisland.service.elasticsearch.Elasticsearch_5_4_0_ClientService

Install done!
```

Remove a component

Just delete the jar on the lib folder or use the components.sh with the -r option.

Example

```
bin/components.sh -i com.hurence.logisland:logisland-service-elasticsearch_5_4_0-
↪client:1.0.0
```

1.6.5 Which module contains my component?

You can easily know with module you require to install in case you need a specific component.

The component documentation contains a *Module* section for each component. It will tell you the artifact you should install.

1.6.6 How about the distribution?

Logisland uses apache ivy to download the plugins. This allows you to choose the right repository (e.g. a common nexus or an enterprise artifactory) in order to manage and control the dependencies.

You can fine tune this by editing (at your own risks) the ivy.xml file on the conf directory.

1.7 Connectors

In this chapter we will present you how to integrate kafka connect connectors into logisland.

Table of Contents

- *Connectors*

- *Introduction*
- *Prerequisites*
- *Getting started*
- *Configuring*
- *Choosing the right converter*
- *Putting all together*
- *Going further*

1.7.1 Introduction

Logisland features the integration between kafka connect world and the spark structured streaming engine.

In order to seamlessy integrate both world, we just wrapped out the kafka connectors interfaces (unplugging them from kafka) and let the run in a logisland spark managed container. Hence the name *"Logisland Connect"* :-)

This allows you to leverage the existing kafka connectors library to import data into a logisland pipeline without having the need to make use of any another middleware or ETL system.

1.7.2 Prerequisites

You can use this functionality only with a spark engine >= 2.1.x

1.7.3 Getting started

In order to use a kafka connect source or sink you have to package and install the required libraries to the logisland lib folder.

Hopefully it can be easily done by using the *components.sh* tool.

```
bin/components.sh -i <plugin_artifact>
```

The plugin artifact should be provided according this format: *groupId:artifactId:version* where groupId, artifactId and version refer to the maven artifact you're going to install.

Some examples, with the suggested artifacts to use, in the following table:

Connector	URL	Artifact
Simulator	https://github.com/jcustenborder/kafka-connect-simulator	com.github.jcustenborder.kafka.connect:kafka-connect-simulator:0.1.118
OPC-DA / OPC-UA (IIoT)	https://github.com/Hurence/logisland	com.hurence.logisland:logisland-connector-opc:<logisland_version>
FTP	https://github.com/Eneco/kafka-connect-ftp	com.eneco:kafka-connect-ftp:0.1.4
Blockchain	https://github.com/Landoop/stream-reactor/tree/master/kafka-connect-blockchain	com.datamountaineer:kafka-connect-blockchain:1.1.1
JMS	https://github.com/Landoop/stream-reactor/tree/master/kafka-connect-jms	com.datamountaineer:kafka-connect-jms:1.1.1
JDBC	https://docs.confluent.io/current/connect/connect-jdbc/docs/index.html	io.confluent:kafka-connect-jdbc:5.0.0

1.7.4 Configuring

Once you have bundled the connectors you need, you are now ready to use them.

Let's do it step by step.

First of all we need to declare a *KafkaConnectStructuredSourceProviderService* or a *KafkaConnectStructuredSinkProviderService* that will manage our connector in Logisland. Along with this we need to put some configuration (In general you can always refer to kafka connect documentation to better understand the underlying architecture and how to configure a connector):

Property	Description
kc.connector.class	The class of the connector (Fully qualified name)
kc.data.key.converter	The class of the converter to be used for the key. Please refer to *Choosing the right converter* section
kc.data.key.converter.properties	The properties to be provided to the key converter
kc.data.value.converter	The class of the converter to be used for the value. Please refer to *Choosing the right converter* section
kc.data.value.converter.properties	The properties to be provided to the value converter
kc.connector.properties	The properties to be provided to the connector and specific to the connector itself.
kc.worker.tasks.max	How many concurrent threads to spawn for a connector
kc.connector.offset.backing.store	The offset backing store to use. Choose among: • **memory** : standalone in memory • **file** : standalone file based. • **kafka** : distributed kafka topic based
kc.connector.offset.backing.store.properties	**Specific properties to configure the chosen backing** store.

Note: Please refer to Kafka connect guide for further information about offset backing store and how to configure them.

1.7.5 Choosing the right converter

Choosing the right converter is perhaps one of the most important part. In fact we're going to adapt what is coming from kafka connect to what is flowing into our logisland pipeline. This means that we have to know how the source is managing its data.

In order to simplify your choice, we recommend you to follow this simple approach (the same applies for both keys and values):

Source data	Kafka Converter	Logisland Encoder
String	StringConverter	StringEncoder
Raw Bytes	ByteArrayConverter	BytesArraySerialiser
Structured	LogIslandRecordConverter	The serializer used by the record converter (*)

Note: (*)In case you deal with structured data, the LogIslandRecordConverter will embed the structured object in a logisland record. In order to do this you have to specify the serializer to be used to convert your data (the serializer property **record.serializer**). Generally the *KryoSerialiser* is a good choice to start with.

1.7.6 Putting all together

In the previous two sections we explained how to configure a connector and how to choose the right serializer for it.

The recap we can examine the following configuration example:

```
 # Our source service
- controllerService: kc_source_service
  component: com.hurence.logisland.stream.spark.provider.
↪KafkaConnectStructuredSourceProviderService
  documentation: A kafka source connector provider reading from its own source and␣
↪providing structured streaming to the underlying layer
  configuration:
    # We will use the logisland record converter for both key and value
    kc.data.value.converter: com.hurence.logisland.connect.converter.
↪LogIslandRecordConverter
    # Use kryo to serialize the inner data
    kc.data.value.converter.properties: |
      record.serializer=com.hurence.logisland.serializer.KryoSerializer
    kc.data.key.converter: com.hurence.logisland.connect.converter.
↪LogIslandRecordConverter
    # Use kryo to serialize the inner data
    kc.data.key.converter.properties: |
      record.serializer=com.hurence.logisland.serializer.KryoSerializer
    # Only one task to handle source input (unique)
    kc.worker.tasks.max: 1
    # The kafka source connector to wrap (here we're using a simulator source)
    kc.connector.class: com.github.jcustenborder.kafka.connect.simulator.
↪SimulatorSourceConnector
    # The properties for the connector (as per connector documentation)
    kc.connector.properties: |
      key.schema.fields=email
      topic=simulator
      value.schema.fields=email,firstName,middleName,lastName,telephoneNumber,
↪dateOfBirth
    # We are using a standalone source for testing. We can store processed offsets in␣
↪memory
    kc.connector.offset.backing.store: memory
```

In the example both key and value provided by the connector are structured objects.

For this reason we use for that the converter *LogIslandRecordConverter*. We provide the serializer to be used for both key and value converter specifying

```
record.serializer=com.hurence.logisland.serializer.KryoSerializer
```

among the related converter properties.

1.7.7 Going further

Please do not hesitate to take a look to our kafka connect tutorials for more details and practical use cases.

1.8 Tutorials

Chat with us on Gitter

Download the latest release build and unzip on an edge node.

Contents:

1.8.1 Prerequisites

There are two main ways to launch a logisland job :

- within Docker containers
- within an Hadoop distribution (Cloudera, Hortonworks, . . .)

1. Trough a Docker container (testing way)

Logisland is packaged as a Docker container that you can build yourself or pull from Docker Hub.

To facilitate integration testing and to easily run tutorials, you can use *docker-compose* with the following docker-compose.yml.

Once you have this file you can run a *docker-compose* command to launch all the needed services (zookeeper, kafka, es, kibana and logisland)

Elasticsearch on docker needs a special tweak as described here

```
# set vm.max_map_count kernel setting for elasticsearch
sudo sysctl -w vm.max_map_count=262144

#
cd /tmp
wget https://raw.githubusercontent.com/Hurence/logisland/master/logisland-framework/
↪logisland-resources/src/main/resources/conf/docker-compose.yml
docker-compose up
```

Note: you should add an entry for **sandbox** and **kafka** (with the container ip) in your `/etc/hosts` as it will be easier to access to all web services in logisland running container.

Any logisland script can now be launched by running a *logisland.sh* script within the logisland docker container like in the example below where we launch *index-apache-logs.yml* job :

```
docker exec -i -t logisland bin/logisland.sh --conf conf/index-apache-logs.yml
```

2. Through an Hadoop cluster (production way)

Now you have played with the tool, you're ready to deploy your jobs into a real distributed cluster. From an edge node of your cluster :

- download and extract the latest release of logisland
- export *SPARK_HOME* and *HADOOP_CONF_DIR* environment variables
- run *logisland.sh* launcher script with your job conf file.

```
cd /opt
sudo wget https://github.com/Hurence/logisland/releases/download/v1.1.1/logisland-1.1.
↪1-bin.tar.gz

export SPARK_HOME=/opt/spark-2.1.0-bin-hadoop2.7/
export HADOOP_CONF_DIR=$SPARK_HOME/conf

sudo /opt/logisland-1.1.1/bin/logisland.sh --conf /home/hurence/tom/logisland-conf/v0.
↪10.0/future-factory.yml
```

1.8.2 Apache logs indexing

In the following getting started tutorial we'll drive you through the process of Apache log mining with LogIsland platform.

Note: It is possible to store data in different datastores. In this tutorial, we will see the case of ElasticSearch ,Solr and MongoDb.

- Apache logs indexing into elasticsearch
- Apache logs indexing into solr
- Apache logs indexing into mongodb

1.8.3 Apache logs indexing with elasticsearch

In the following getting started tutorial we'll drive you through the process of Apache log mining with LogIsland platform. The final data will be stored in elasticsearch

This tutorial is very similar to :

- Apache logs indexing into solr
- Apache logs indexing into mongodb

Note: Please note that you should not launch silmutaneously several docker-compose because we are exposing local port in them. So running several at the same time would be conflicting. So be sure to have killed all your currently running containers.

1.Install required components

- You either use docker-compose with available docker-compose-index-apache-logs-es.yml file in the tar.gz assembly in the conf folder.

  In this case you can skip this section

- Or you can launch the job in your cluster, but in this case you will have to make changes to job conf file so it works in your environment.

  In this case please make sure to already have installed elasticsearch modules (depending on which base you will use).

If not you can just do it through the components.sh command line:

```
bin/components.sh -i com.hurence.logisland:logisland-processor-elasticsearch:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-service-elasticsearch_5_4_0-
↪client:1.1.1
```

Note: In the following sections we will use docker-compose to run the job. (please install it before pursuing if you are not using your own cluster)

2. Logisland job setup

The logisland job that we will use is **./conf/index-apache-logs-es.yml** The logisland docker-compose file that we will use is **./conf/docker-compose-index-apache-logs-es.yml**

We will start by explaining each part of the config file.

An Engine is needed to handle the stream processing. This `conf/index-apache-logs-es.yml` configuration file defines a stream processing job setup. The first section configures the Spark engine (we will use a KafkaStreamProcessingEngine) to run in local mode with 2 cpu cores and 2G of RAM.

```
engine:
  component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine
  type: engine
  documentation: Index some apache logs with logisland
  configuration:
    spark.app.name: IndexApacheLogsDemo
    spark.master: local[2]
    spark.driver.memory: 1G
    spark.driver.cores: 1
    spark.executor.memory: 2G
    spark.executor.instances: 4
    spark.executor.cores: 2
    spark.yarn.queue: default
    spark.yarn.maxAppAttempts: 4
    spark.yarn.am.attemptFailuresValidityInterval: 1h
    spark.yarn.max.executor.failures: 20
    spark.yarn.executor.failuresValidityInterval: 1h
    spark.task.maxFailures: 8
    spark.serializer: org.apache.spark.serializer.KryoSerializer
    spark.streaming.batchDuration: 1000
    spark.streaming.backpressure.enabled: false
    spark.streaming.unpersist: false
    spark.streaming.blockInterval: 500
    spark.streaming.kafka.maxRatePerPartition: 3000
    spark.streaming.timeout: -1
    spark.streaming.kafka.maxRetries: 3
    spark.streaming.ui.retainedBatches: 200
    spark.streaming.receiver.writeAheadLog.enable: false
    spark.ui.port: 4050
```

The *controllerServiceConfigurations* part is here to define all services that be shared by processors within the whole job, here an Elasticsearch service that will be used later in the `BulkAddElasticsearch` processor.

```
- controllerService: elasticsearch_service
  component: com.hurence.logisland.service.elasticsearch.Elasticsearch_5_4_0_
↪ClientService
```

(continues on next page)

(continued from previous page)

```
  type: service
  documentation: elasticsearch service
  configuration:
    hosts: ${ES_HOSTS}
    cluster.name: ${ES_CLUSTER_NAME}
    batch.size: 5000
```

Note: As you can see it uses environment variable so make sure to set them. (if you use the docker-compose file of this tutorial it is already done for you)

Inside this engine you will run a Kafka stream of processing, so we setup input/output topics and Kafka/Zookeeper hosts. Here the stream will read all the logs sent in `logisland_raw` topic and push the processing output into `logisland_events` topic.

Note: We want to specify an Avro output schema to validate our ouput records (and force their types accordingly). It's really for other streams to rely on a schema when processing records from a topic.

We can define some serializers to marshall all records from and to a topic.

```
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that converts raw apache logs into structured log records
  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
    kafka.metadata.broker.list: ${KAFKA_BROKERS}
    kafka.zookeeper.quorum: ${ZK_QUORUM}
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
```

Note: As you can see it uses environment variable so make sure to set them. (if you use the docker-compose file of this tutorial it is already done for you)

Within this stream a `SplitText` processor takes a log line as a String and computes a `Record` as a sequence of fields.

```
# parse apache logs into logisland records
- processor: apache_parser
  component: com.hurence.logisland.processor.SplitText
  type: parser
  documentation: a parser that produce events from an apache log REGEX
  configuration:
    record.type: apache_log
    value.regex: (\S+)\s+(\S+)\s+(\S+)\s+\[([\w:\/]+\s[+\-]\d{4})\]\s+
↪"(\S+)\s+(\S+)\s*(\S*)"\s+(\S+)\s+(\S+)
```

(continues on next page)

(continued from previous page)

```
    value.fields: src_ip,identd,user,record_time,http_method,http_query,http_version,
↪http_status,bytes_out
```

This stream will process log entries as soon as they will be queued into *logisland_raw* Kafka topics, each log will be parsed as an event which will be pushed back to Kafka in the `logisland_events` topic.

The second processor will handle `Records` produced by the `SplitText` to index them into elasticsearch

```
# all the parsed records are added to elasticsearch by bulk
- processor: es_publisher
  component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch
  type: processor
  documentation: a processor that indexes processed events in elasticsearch
  configuration:
    elasticsearch.client.service: elasticsearch_service
    default.index: logisland
    default.type: event
    timebased.index: yesterday
    es.index.field: search_index
    es.type.field: record_type
```

3. Launch the job

For this tutorial we will handle some apache logs with a splitText parser and send them to Elastiscearch. Launch your docker container with this command (we suppose you are in the root of the tar gz assembly) :

```
sudo docker-compose -f ./conf/docker-compose-index-apache-logs-es.yml up -d
```

Make sure all container are running and that there is no error.

```
sudo docker-compose ps
```

Those containers should be visible and running

"' CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES 0d9e02b22c38 docker.elastic.co/kibana/kibana:5.4.0 "/bin/sh -c /usr/loc..." 13 seconds ago Up 8 seconds 0.0.0.0:5601->5601/tcp conf_kibana_1 ab15f4b5198c docker.elastic.co/elasticsearch/elasticsearch:5.4.0 "/bin/bash bin/es-do..." 13 seconds ago Up 7 seconds 0.0.0.0:9200->9200/tcp, 0.0.0.0:9300->9300/tcp conf_elasticsearch_1 a697e45d2d1a hurence/logisland:1.1.1 "tail -f bin/logisla..." 13 seconds ago Up 9 seconds 0.0.0.0:4050->4050/tcp, 0.0.0.0:8082->8082/tcp, 0.0.0.0:9999->9999/tcp conf_logisland_1 db80cdf23b45 hurence/zookeeper "/bin/sh -c '/usr/sb..." 13 seconds ago Up 10 seconds 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp, 7072/tcp conf_zookeeper_1 7aa7a87dd16b hurence/kafka:0.10.2.2-scala-2.11 "start-kafka.sh" 13 seconds ago Up 5 seconds 0.0.0.0:9092->9092/tcp conf_kafka_1

"'

```
sudo docker logs conf_kibana_1
sudo docker logs conf_elasticsearch_1
sudo docker logs conf_logisland_1
sudo docker logs conf_zookeeper_1
sudo docker logs conf_kafka_1
```

Should not return errors or any suspicious messages

you can now run the job inside the logisland container

```
sudo docker exec -ti conf_logisland_1 ./bin/logisland.sh --conf ./conf/index-apache-
↪logs-es.yml
```

The last logs should be something like :

2019-03-19 16:08:47 INFO StreamProcessingRunner:95 - awaitTermination for engine 1 2019-03-19 16:08:47 WARN SparkContext:66 - Using an existing SparkContext; some configuration may not take effect.

4. Inject some Apache logs into the system

Now we're going to send some logs to `logisland_raw` Kafka topic.

If you don't have your own httpd logs available, you can use some freely available log files from NASA-HTTP web site access:

- Jul 01 to Jul 31, ASCII format, 20.7 MB gzip compressed
- Aug 04 to Aug 31, ASCII format, 21.8 MB gzip compressed

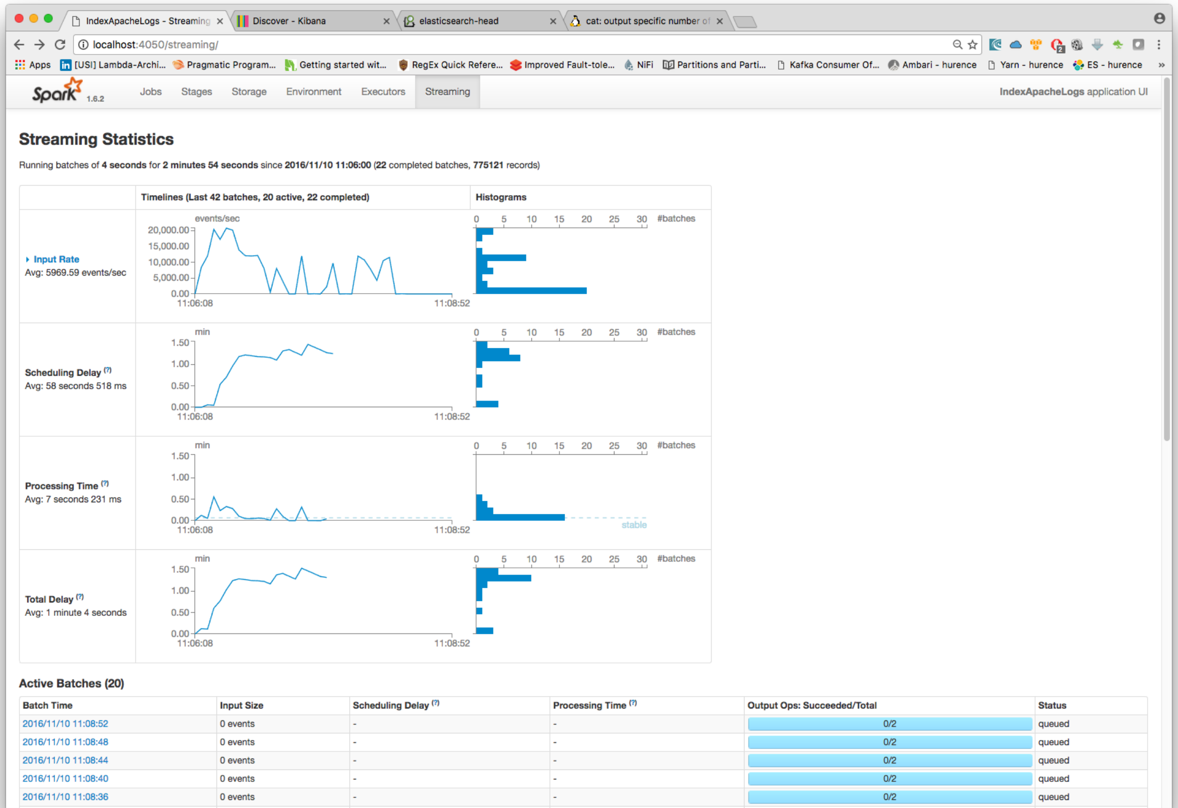
Let's send the first 500 lines of NASA http access over July 1995 to LogIsland with kafka scripts (available in our logisland container) to `logisland_raw` Kafka topic.

In another terminal run those commands

```
sudo docker exec -ti conf_logisland_1 bash
cd /tmp
wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz
gunzip NASA_access_log_Jul95.gz
head -n 500 NASA_access_log_Jul95 | ${KAFKA_HOME}/bin/kafka-console-producer.sh --
↪broker-list kafka:9092 --topic logisland_raw
```

5. Monitor your spark jobs and Kafka topics

Now go to http://localhost:4050/streaming/ to see how fast Spark can process your data



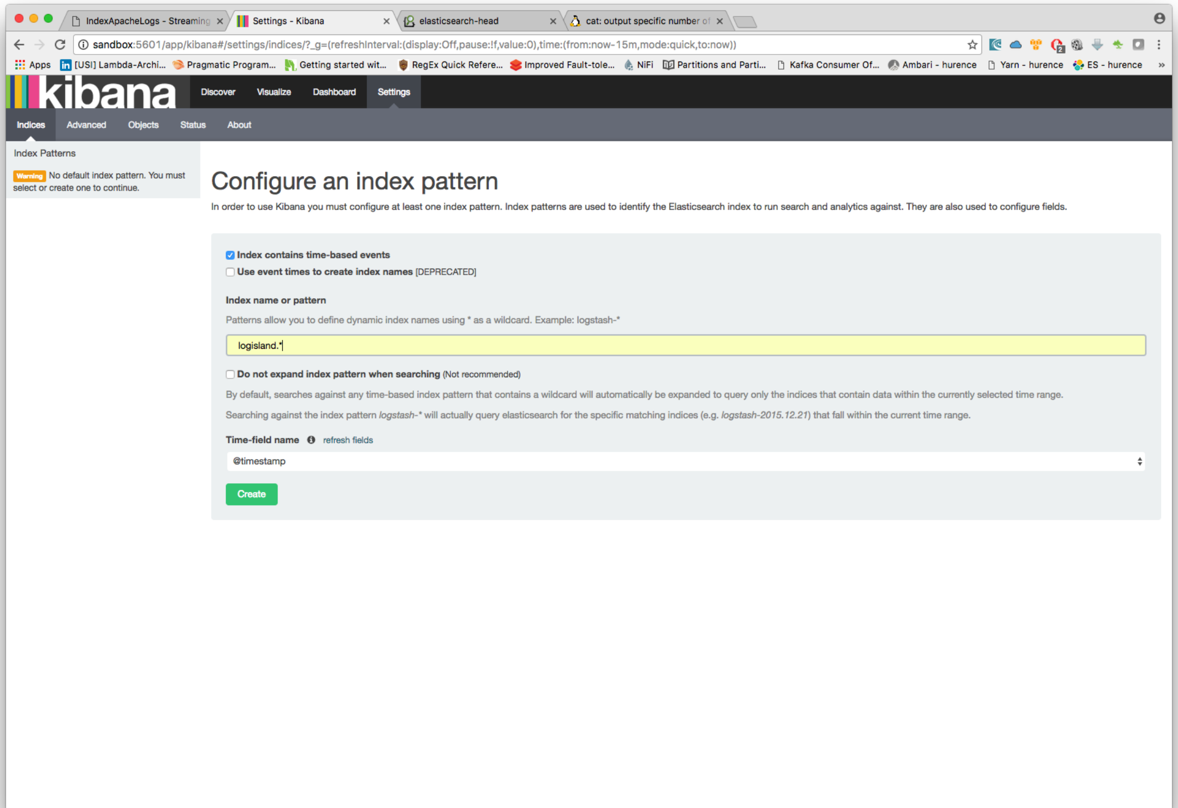
6. Inspect the logs

Kibana

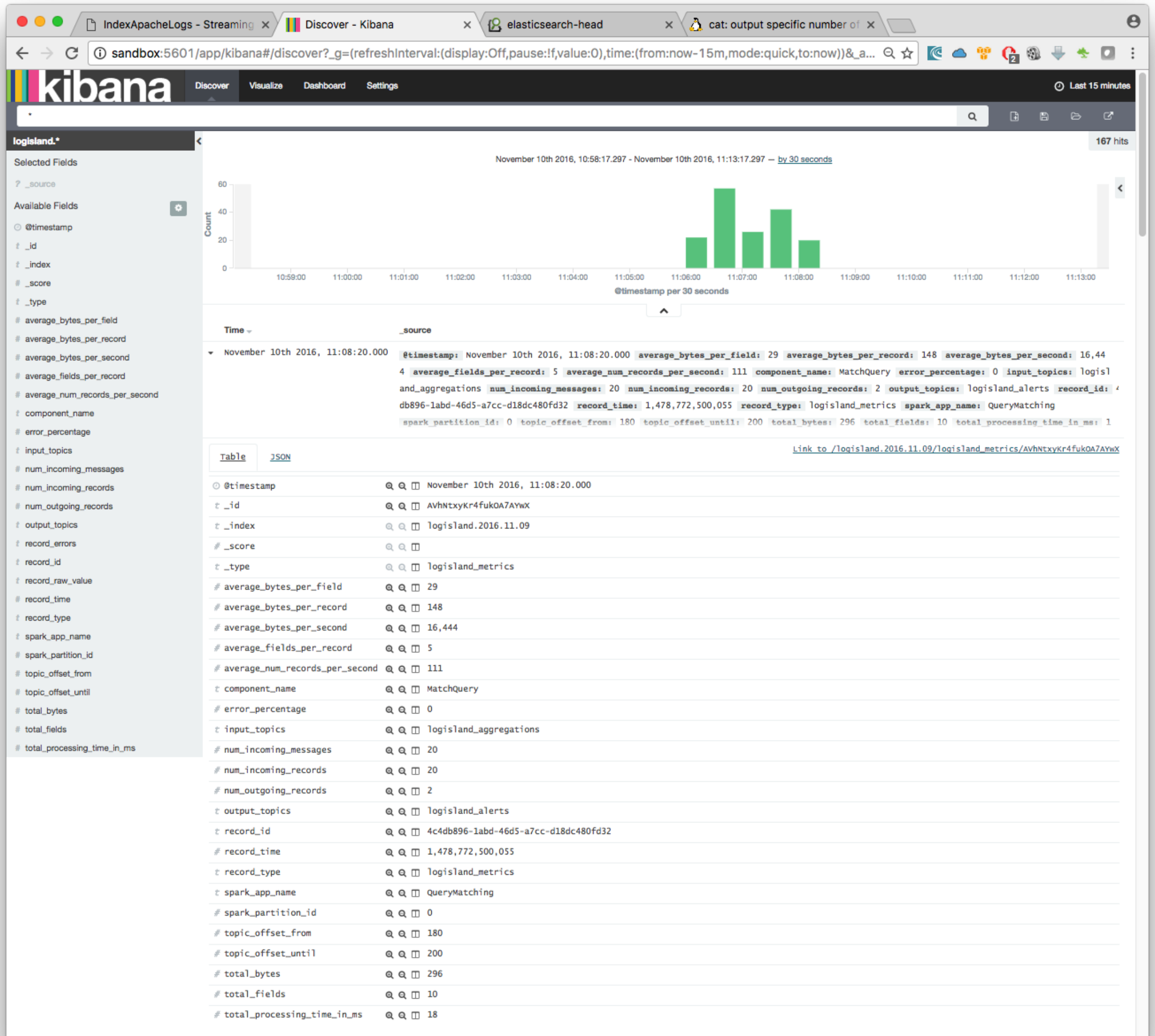
With ElasticSearch, you can use Kibana. We included one in our docker-compose file.

Open up your browser and go to http://localhost:5601/ and you should be able to explore your apache logs.

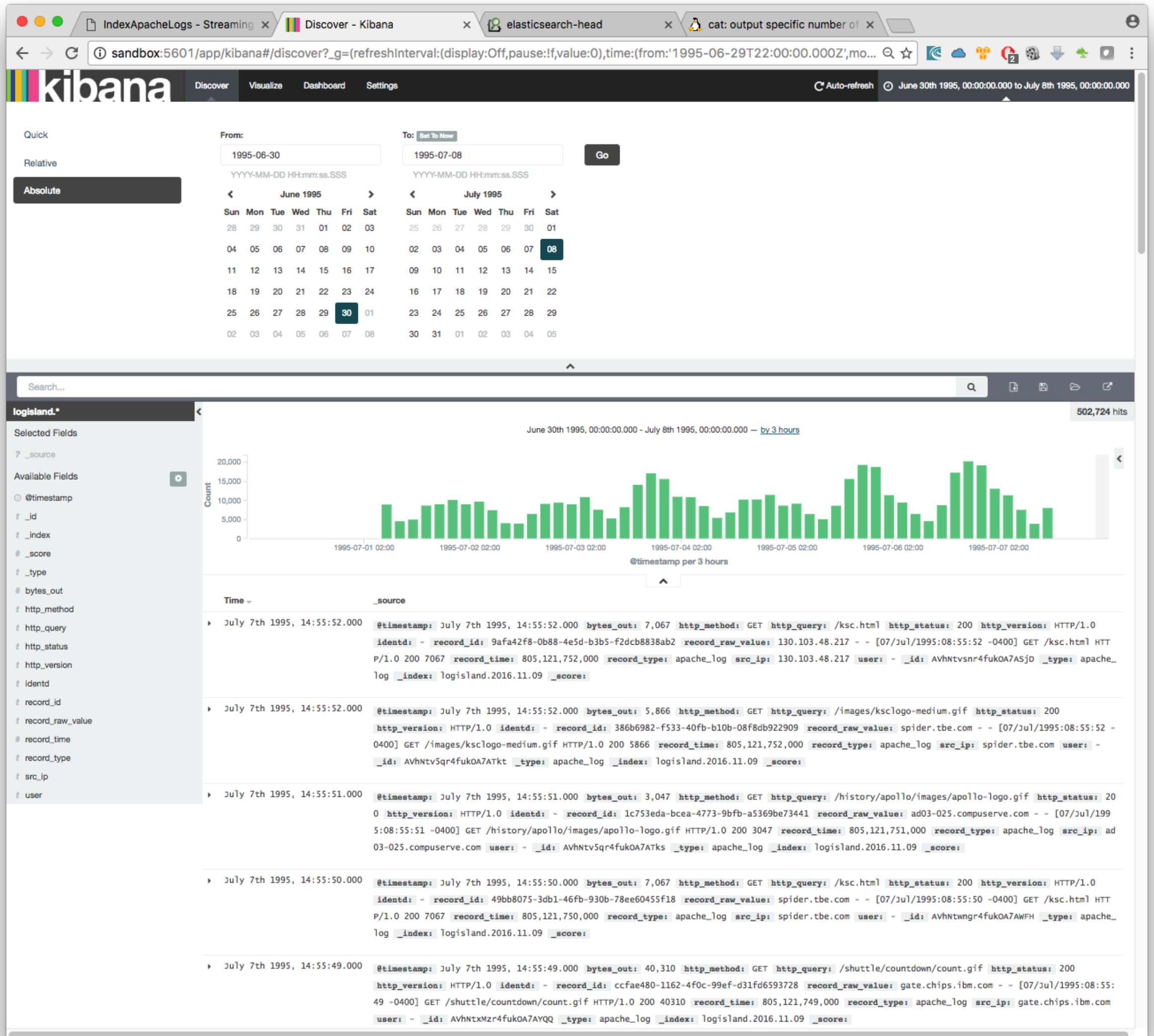
Configure a new index pattern with `logisland.*` as the pattern name and `@timestamp` as the time value field.



Then if you go to Explore panel for the latest 15' time window you'll only see logisland process_metrics events which give you insights about the processing bandwidth of your streams.



As we explore data logs from july 1995 we'll have to select an absolute time filter from 1995-06-30 to 1995-07-08 to see the events.



3. Stop the job

You can Ctr+c the console where you launched logisland job. Then to kill all containers used run :

```
sudo docker-compose -f ./conf/docker-compose-index-apache-logs-es.yml down
```

Make sure all container have disappeared.

```
sudo docker ps
```

1.8.4 Apache logs indexing with mongo

In the following getting started tutorial we'll drive you through the process of Apache log mining with LogIsland platform. The final data will be stored in mongo

This tutorial is very similar to :

- Apache logs indexing into solr
- Apache logs indexing into elasticsearch

Note: Please note that you should not launch silmutaneously several docker-compose because we are exposing local port in them. So running several at the same time would be conflicting. So be sure to have killed all your currently running containers.

1.Install required components

- You either use docker-compose with available docker-compose-index-apache-logs-mongo.yml file in the tar.gz assembly in the conf folder.

  In this case you can skip this section

- Or you can launch the job in your cluster, but in this case you will have to make changes to job conf file so it works in your environment.

  In this case please make sure to already have installed mongo modules (depending on which base you will use).

If not you can just do it through the components.sh command line:

```
bin/components.sh -i com.hurence.logisland:logisland-service-mongodb-client:1.1.1
```

Note: In the following sections we will use docker-compose to run the job. (please install it before pursuing if you are not using your own cluster)

---

2. Logisland job setup

The logisland job that we will use is **./conf/index-apache-logs-mongo.yml** The logisland docker-compose file that we will use is **./conf/docker-compose-index-apache-logs-mongo.yml**

We will start by explaining each part of the config file.

An Engine is needed to handle the stream processing. This `conf/index-apache-logs-mongo.yml` configuration file defines a stream processing job setup. The first section configures the Spark engine (we will use a KafkaStreamProcessingEngine) to run in local mode with 2 cpu cores and 2G of RAM.

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

(continues on next page)

(continued from previous page)

```
    spark.yarn.am.attemptFailuresValidityInterval: 1h
    spark.yarn.max.executor.failures: 20
    spark.yarn.executor.failuresValidityInterval: 1h
    spark.task.maxFailures: 8
    spark.serializer: org.apache.spark.serializer.KryoSerializer
    spark.streaming.batchDuration: 1000
    spark.streaming.backpressure.enabled: false
    spark.streaming.unpersist: false
    spark.streaming.blockInterval: 500
    spark.streaming.kafka.maxRatePerPartition: 3000
    spark.streaming.timeout: -1
    spark.streaming.kafka.maxRetries: 3
    spark.streaming.ui.retainedBatches: 200
    spark.streaming.receiver.writeAheadLog.enable: false
    spark.ui.port: 4050
```

The *controllerServiceConfigurations* part is here to define all services that be shared by processors within the whole job, here an mongo service that will be used later in the `TODO` processor.

```
- controllerService: datastore_service
  component: com.hurence.logisland.service.mongodb.MongoDBControllerService
  type: service
  documentation: "Mongo 3.8.0 service"
  configuration:
    mongo.uri: ${MONGO_URI}
    mongo.db.name: logisland
    mongo.collection.name: apache
    # possible values ACKNOWLEDGED, UNACKNOWLEDGED, FSYNCED, JOURNALED, REPLICA_
↪ACKNOWLEDGED, MAJORITY
    mongo.write.concern: ACKNOWLEDGED
    flush.interval: 2000
    batch.size: 100
```

Note: As you can see it uses environment variable so make sure to set them. (if you use the docker-compose file of this tutorial it is already done for you)

Inside this engine you will run a Kafka stream of processing, so we setup input/output topics and Kafka/Zookeeper hosts. Here the stream will read all the logs sent in `logisland_raw` topic and push the processing output into `logisland_events` topic.

Note: We want to specify an Avro output schema to validate our ouput records (and force their types accordingly). It's really for other streams to rely on a schema when processing records from a topic.

We can define some serializers to marshall all records from and to a topic.

```
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that converts raw apache logs into structured log records
  configuration:
    kafka.input.topics: logisland_raw
    kafka.output.topics: logisland_events
    kafka.error.topics: logisland_errors
```

(continues on next page)

(continued from previous page)

```
    kafka.input.topics.serializer: none
    kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: ${KAFKA_BROKERS}
    kafka.zookeeper.quorum: ${ZK_QUORUM}
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
```

Note: As you can see it uses environment variable so make sure to set them. (if you use the docker-compose file of this tutorial it is already done for you)

Within this stream a `SplitText` processor takes a log line as a String and computes a `Record` as a sequence of fields.

```
# parse apache logs into logisland records
- processor: apache_parser
  component: com.hurence.logisland.processor.SplitText
  type: parser
  documentation: a parser that produce events from an apache log REGEX
  configuration:
    record.type: apache_log
    value.regex: (\S+)\s+(\S+)\s+(\S+)\s+\[([\w:\/]+\s[+\-]\d{4})\]\s+
↪"(\S+)\s+(\S+)\s*(\S*)"\s+(\S+)\s+(\S+)
    value.fields: src_ip,identd,user,record_time,http_method,http_query,http_version,
↪http_status,bytes_out
```

This stream will process log entries as soon as they will be queued into *logisland_raw* Kafka topics, each log will be parsed as an event which will be pushed back to Kafka in the `logisland_events` topic.

The second processor will handle `Records` produced by the `SplitText` to index them into solr

all the parsed records are added to mongo by bulk - processor: mongo_publisher

component: com.hurence.logisland.processor.datastore.BulkPut type: processor documentation: “indexes processed events in Mongo” configuration:

datastore.client.service: datastore_service

3. Launch the job

1. Run docker-compose

For this tutorial we will handle some apache logs with a splitText parser and send them to Elastiscearch. Launch your docker container with this command (we suppose you are in the root of the tar gz assembly) :

```
sudo docker-compose -f ./conf/docker-compose-index-apache-logs-es.yml up -d
```

Make sure all container are running and that there is no error.

```
sudo docker-compose ps
```

Those containers should be visible and running

‘‘‘ CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES 0d9e02b22c38 docker.elastic.co/kibana/kibana:5.4.0 “/bin/sh -c /usr/loc…” 13 seconds ago Up 8 seconds 0.0.0.0:5601->5601/tcp conf_kibana_1 ab15f4b5198c docker.elastic.co/elasticsearch/elasticsearch:5.4.0 “/bin/bash bin/es-do…” 13 seconds ago Up 7 seconds 0.0.0.0:9200->9200/tcp, 0.0.0.0:9300->9300/tcp conf_elasticsearch_1 a697e45d2d1a hurence/logisland:1.1.1 “tail -f bin/logisla…” 13 seconds ago Up 9 seconds 0.0.0.0:4050->4050/tcp, 0.0.0.0:8082->8082/tcp, 0.0.0.0:9999->9999/tcp conf_logisland_1 db80cdf23b45 hurence/zookeeper “/bin/sh -c ‘/usr/sb…” 13 seconds ago Up 10 seconds 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp, 7072/tcp conf_zookeeper_1 7aa7a87dd16b hurence/kafka:0.10.2.2-scala-2.11 “start-kafka.sh” 13 seconds ago Up 5 seconds 0.0.0.0:9092->9092/tcp conf_kafka_1

‘‘‘

```
sudo docker logs conf_kibana_1
sudo docker logs conf_elasticsearch_1
sudo docker logs conf_logisland_1
sudo docker logs conf_zookeeper_1
sudo docker logs conf_kafka_1
```

Should not return errors or any suspicious messages

2. Initializing mongo db

Note: You have to create the db logisland with the collection apache.

```
# open the mongo shell inside mongo container
sudo docker exec -ti conf_mongo_1 mongo

> use logisland
switched to db logisland

> db.apache.insert({src_ip:"19.123.12.67", identd:"-", user:"-", bytes_out:12344,␣
↪http_method:"POST", http_version:"2.0", http_query:"/logisland/is/so?great=true",
↪http_status:"404" })
WriteResult({ "nInserted" : 1 })

> db.apache.find()
```

{ “_id” : ObjectId(“5b4f3c4a5561b53b7e862b57”), “src_ip” : “19.123.12.67”, “identd” : “-“, “user” : “-“, “bytes_out” : 12344, “http_method” : “POST”, “http_version” : “2.0”, “http_query” : “/logisland/is/so?great=true”, “http_status” : “404” }

3. Run logisland job

you can now run the job inside the logisland container

```
sudo docker exec -ti conf_logisland_1 ./bin/logisland.sh --conf ./conf/index-apache-
↪logs-mongo.yml
```

The last logs should be something like :

2019-03-19 16:08:47 INFO StreamProcessingRunner:95 - awaitTermination for engine 1 2019-03-19 16:08:47 WARN SparkContext:66 - Using an existing SparkContext; some configuration may not take effect.

4. Inject some Apache logs into the system

Now we're going to send some logs to `logisland_raw` Kafka topic.

If you don't have your own httpd logs available, you can use some freely available log files from NASA-HTTP web site access:

- Jul 01 to Jul 31, ASCII format, 20.7 MB gzip compressed
- Aug 04 to Aug 31, ASCII format, 21.8 MB gzip compressed

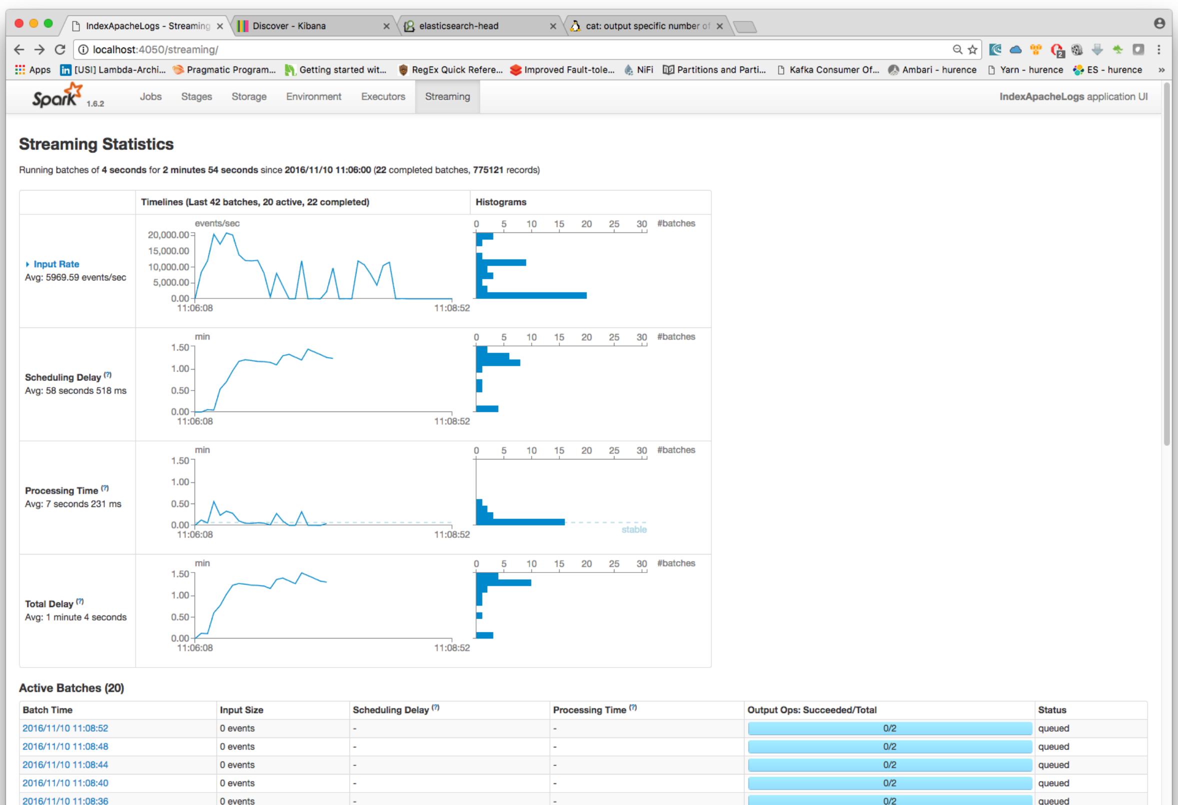
Let's send the first 500 lines of NASA http access over July 1995 to LogIsland with kafka scripts (available in our logisland container) to `logisland_raw` Kafka topic.

In another terminal run those commands

```
sudo docker exec -ti conf_logisland_1 bash
cd /tmp
wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz
gunzip NASA_access_log_Jul95.gz
head -n 500 NASA_access_log_Jul95 | ${KAFKA_HOME}/bin/kafka-console-producer.sh --
↪broker-list kafka:9092 --topic logisland_raw
```

5. Monitor your spark jobs and Kafka topics

Now go to http://localhost:4050/streaming/ to see how fast Spark can process your data



6. Inspect the logs

With mongo you can directly use the shell:

```
> db.apache.find()
```

{ "_id" : "507adf3e-3162-4ff0-843a-253e01a6df69", "src_ip" : "129.94.144.152", "record_id" : "507adf3e-3162-4ff0-843a-253e01a6df69", "http_method" : "GET", "record_value" : "129.94.144.152 - - [01/Jul/1995:00:00:17 -0400] "GET /images/ksclogo-medium.gif HTTP/1.0" 304 0", "http_query" : "/images/ksclogo-medium.gif", "bytes_out" : "0", "identd" : "-", "http_version" : "HTTP/1.0", "http_status" : "304", "record_time" : NumberLong("804571.1.100"), "user" : "-", "record_type" : "apache_log" } { "_id" : "c44a9d09-52b9-4ada-8126-39c70c90fdd3", "src_ip" : "ppp-mia-30.shadow.net", "record_id" : "c44a9d09-52b9-4ada-8126-39c70c90fdd3", "http_method" : "GET", "record_value" : "ppp-mia-30.shadow.net - - [01/Jul/1995:00:00:27 -0400] "GET / HTTP/1.0" 200 7074", "http_query" : "/", "bytes_out" : "7074", "identd" : "-", "http_version" : "HTTP/1.0", "http_status" : "200", "record_time" : NumberLong("804571227000"), "user" : "-", "record_type" : "apache_log" } …

3. Stop the job

You can Ctr+c the console where you launched logisland job. Then to kill all containers used run :

```
sudo docker-compose -f ./conf/docker-compose-index-apache-logs-es.yml down
```

Make sure all container have disappeared.

```
sudo docker ps
```

1.8.5 Apache logs indexing with solr

In the following getting started tutorial we'll drive you through the process of Apache log mining with LogIsland platform. The final data will be stored in solr

This tutorial is very similar to :

- Apache logs indexing into mongodb
- Apache logs indexing into elasticsearch

Note: Please note that you should not launch silmutaneously several docker-compose because we are exposing local port in them. So running several at the same time would be conflicting. So be sure to have killed all your currently running containers.

1.Install required components

- You either use docker-compose with available docker-compose-index-apache-logs-es.yml file in the tar.gz assembly in the conf folder.

  In this case you can skip this section

- Or you can launch the job in your cluster, but in this case you will have to make changes to job conf file so it works in your environment.

  In this case please make sure to already have installed solr modules (depending on which base you will use).

If not you can just do it through the components.sh command line:

```
bin/components.sh -i com.hurence.logisland:logisland-service-mongodb-client:1.1.1
```

Note: In the following sections we will use docker-compose to run the job. (please install it before pursuing if you are not using your own cluster)

2. Logisland job setup

The logisland job that we will use is **./conf/index-apache-logs-solr.yml** The logisland docker-compose file that we will use is **./conf/docker-compose-index-apache-logs-solr.yml**

We will start by explaining each part of the config file.

An Engine is needed to handle the stream processing. This `conf/index-apache-logs-solr.yml` configuration file defines a stream processing job setup. The first section configures the Spark engine (we will use a KafkaStreamProcessingEngine) to run in local mode with 2 cpu cores and 2G of RAM.

```
engine:
  component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine
  type: engine
  documentation: Index some apache logs with logisland
  configuration:
    spark.app.name: IndexApacheLogsDemo
    spark.master: local[2]
    spark.driver.memory: 1G
    spark.driver.cores: 1
    spark.executor.memory: 2G
    spark.executor.instances: 4
    spark.executor.cores: 2
    spark.yarn.queue: default
    spark.yarn.maxAppAttempts: 4
    spark.yarn.am.attemptFailuresValidityInterval: 1h
    spark.yarn.max.executor.failures: 20
    spark.yarn.executor.failuresValidityInterval: 1h
    spark.task.maxFailures: 8
    spark.serializer: org.apache.spark.serializer.KryoSerializer
    spark.streaming.batchDuration: 1000
    spark.streaming.backpressure.enabled: false
    spark.streaming.unpersist: false
    spark.streaming.blockInterval: 500
    spark.streaming.kafka.maxRatePerPartition: 3000
    spark.streaming.timeout: -1
    spark.streaming.kafka.maxRetries: 3
    spark.streaming.ui.retainedBatches: 200
    spark.streaming.receiver.writeAheadLog.enable: false
    spark.ui.port: 4050
```

The *controllerServiceConfigurations* part is here to define all services that be shared by processors within the whole job, here an Solr service that will be used later in the `TODO` processor.

```
# Datastore service using Solr 6.6.2 - 5.5.5 also available
- controllerService: datastore_service
  component: com.hurence.logisland.service.solr.Solr_6_6_2_ClientService
  type: service
```

(continues on next page)

(continued from previous page)

```
  documentation: "SolR 6.6.2 service"
  configuration:
    solr.cloud: false
    solr.connection.string: ${SOLR_CONNECTION}
    solr.collection: solr-apache-logs
    solr.concurrent.requests: 4
    flush.interval: 2000
    batch.size: 1000
```

Note: As you can see it uses environment variable so make sure to set them. (if you use the docker-compose file of this tutorial it is already done for you)

Inside this engine you will run a Kafka stream of processing, so we setup input/output topics and Kafka/Zookeeper hosts. Here the stream will read all the logs sent in `logisland_raw` topic and push the processing output into `logisland_events` topic.

Note: We want to specify an Avro output schema to validate our ouput records (and force their types accordingly). It's really for other streams to rely on a schema when processing records from a topic.

We can define some serializers to marshall all records from and to a topic.

```
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that converts raw apache logs into structured log records
  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
    kafka.metadata.broker.list: ${KAFKA_BROKERS}
    kafka.zookeeper.quorum: ${ZK_QUORUM}
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
```

Note: As you can see it uses environment variable so make sure to set them. (if you use the docker-compose file of this tutorial it is already done for you)

Within this stream a `SplitText` processor takes a log line as a String and computes a `Record` as a sequence of fields.

```
# parse apache logs into logisland records
- processor: apache_parser
  component: com.hurence.logisland.processor.SplitText
  type: parser
  documentation: a parser that produce events from an apache log REGEX
  configuration:
    record.type: apache_log
```

(continues on next page)

(continued from previous page)

```
    value.regex: (\S+)\s+(\S+)\s+(\S+)\s+\[([\w:\/]+\s[+\-]\d{4})\]\s+
↪"(\S+)\s+(\S+)\s*(\S*)"\s+(\S+)\s+(\S+)
    value.fields: src_ip,identd,user,record_time,http_method,http_query,http_version,
↪http_status,bytes_out
```

This stream will process log entries as soon as they will be queued into *logisland_raw* Kafka topics, each log will be parsed as an event which will be pushed back to Kafka in the `logisland_events` topic.

The second processor will handle `Records` produced by the `SplitText` to index them into solr

```
# all the parsed records are added to solr by bulk
- processor: solr_publisher
  component: com.hurence.logisland.processor.datastore.BulkPut
  type: processor
  documentation: "indexes processed events in SolR"
  configuration:
    datastore.client.service: datastore_service
```

3. Launch the job

1. Run docker-compose

For this tutorial we will handle some apache logs with a splitText parser and send them to Elastiscearch. Launch your docker container with this command (we suppose you are in the root of the tar gz assembly) :

```
sudo docker-compose -f ./conf/docker-compose-index-apache-logs-solr.yml up -d
```

Make sure all container are running and that there is no error.

```
sudo docker-compose ps
```

Those containers should be visible and running

"‘ CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES 0d9e02b22c38 docker.elastic.co/kibana/kibana:5.4.0 "/bin/sh -c /usr/loc…" 13 seconds ago Up 8 seconds 0.0.0.0:5601->5601/tcp conf_kibana_1 ab15f4b5198c docker.elastic.co/elasticsearch/elasticsearch:5.4.0 "/bin/bash bin/es-do…" 13 seconds ago Up 7 seconds 0.0.0.0:9200->9200/tcp, 0.0.0.0:9300->9300/tcp conf_elasticsearch_1 a697e45d2d1a hurence/logisland:1.1.1 "tail -f bin/logisla…" 13 seconds ago Up 9 seconds 0.0.0.0:4050->4050/tcp, 0.0.0.0:8082->8082/tcp, 0.0.0.0:9999->9999/tcp conf_logisland_1 db80cdf23b45 hurence/zookeeper "/bin/sh -c '/usr/sb…" 13 seconds ago Up 10 seconds 2888/tcp, 3888/tcp, 0.0.0.0:2181->2181/tcp, 7072/tcp conf_zookeeper_1 7aa7a87dd16b hurence/kafka:0.10.2.2-scala-2.11 "start-kafka.sh" 13 seconds ago Up 5 seconds 0.0.0.0:9092->9092/tcp conf_kafka_1

"‘

```
sudo docker logs conf_kibana_1
sudo docker logs conf_elasticsearch_1
sudo docker logs conf_logisland_1
sudo docker logs conf_zookeeper_1
sudo docker logs conf_kafka_1
```

Should not return errors or any suspicious messages

2. Initializing solr db

We will now set up our solr database. First create the 'solr-apache-logs' collection

```
sudo docker exec -it --user=solr conf_solr_1 bin/solr create_core -c solr-apache-logs
```

The core/collection should have thos fields (corresponding to apache logs parsed fields) [`src_ip`, `identd`, `user`, `bytes_out`,] `http_method`, `http_version`, `http_query`, `http_status`

Otherwise for simplicity you can add a dynamic field called '*' and of type string for this collection with the web ui : http://localhost:8983/solr

Select the solr-apache-logs collection, go to schema and add your fields.

3. Run logisland job

you can now run the job inside the logisland container

```
sudo docker exec -ti conf_logisland_1 ./bin/logisland.sh --conf ./conf/index-apache-
↪logs-solr.yml
```

The last logs should be something like :

2019-03-19 16:08:47 INFO StreamProcessingRunner:95 - awaitTermination for engine 1 2019-03-19 16:08:47 WARN SparkContext:66 - Using an existing SparkContext; some configuration may not take effect.

4. Inject some Apache logs into the system

Now we're going to send some logs to `logisland_raw` Kafka topic.

If you don't have your own httpd logs available, you can use some freely available log files from NASA-HTTP web site access:

- Jul 01 to Jul 31, ASCII format, 20.7 MB gzip compressed
- Aug 04 to Aug 31, ASCII format, 21.8 MB gzip compressed

Let's send the first 500 lines of NASA http access over July 1995 to LogIsland with kafka scripts (available in our logisland container) to `logisland_raw` Kafka topic.

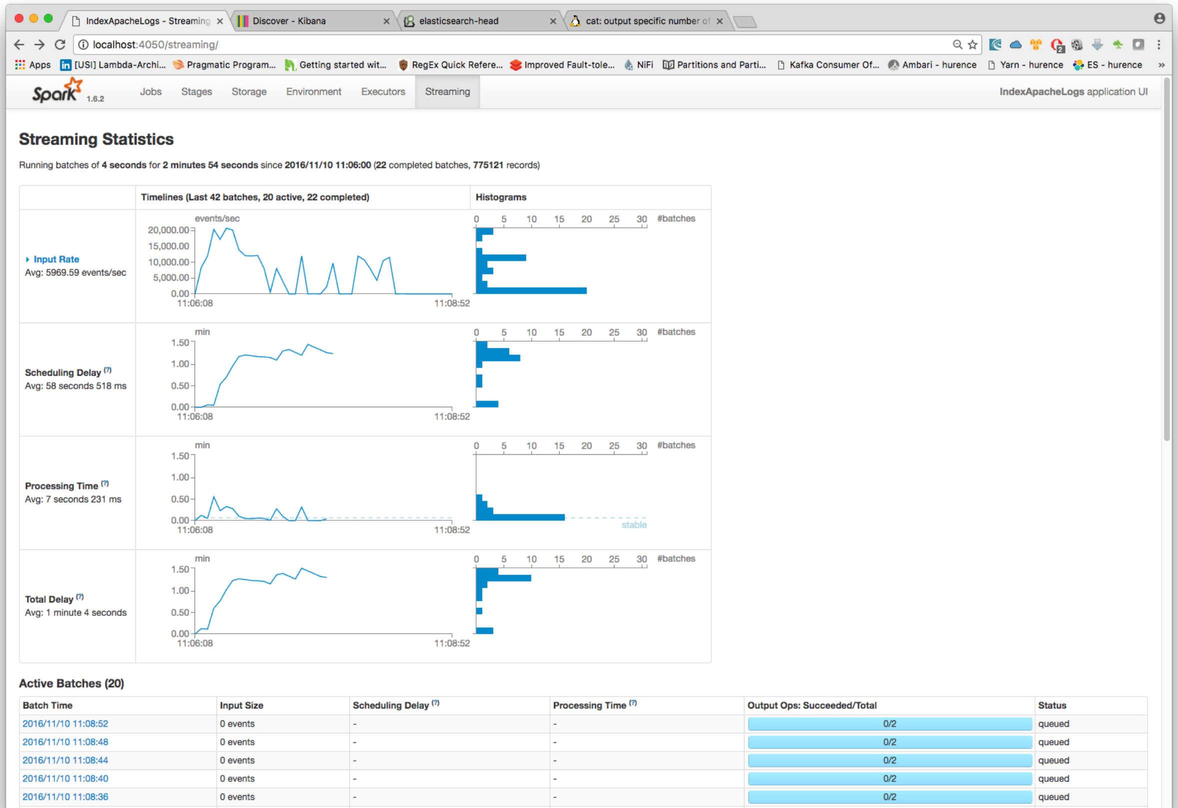
In another terminal run those commands

```
sudo docker exec -ti conf_logisland_1 bash
cd /tmp
wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz
gunzip NASA_access_log_Jul95.gz
head -n 500 NASA_access_log_Jul95 | ${KAFKA_HOME}/bin/kafka-console-producer.sh --
↪broker-list kafka:9092 --topic logisland_raw
```

The logisland job should output logs, verify that there is no error, otherwise there is chances that your solr collection is not well configured.

5. Monitor your spark jobs and Kafka topics

Now go to http://localhost:4050/streaming/ to see how fast Spark can process your data

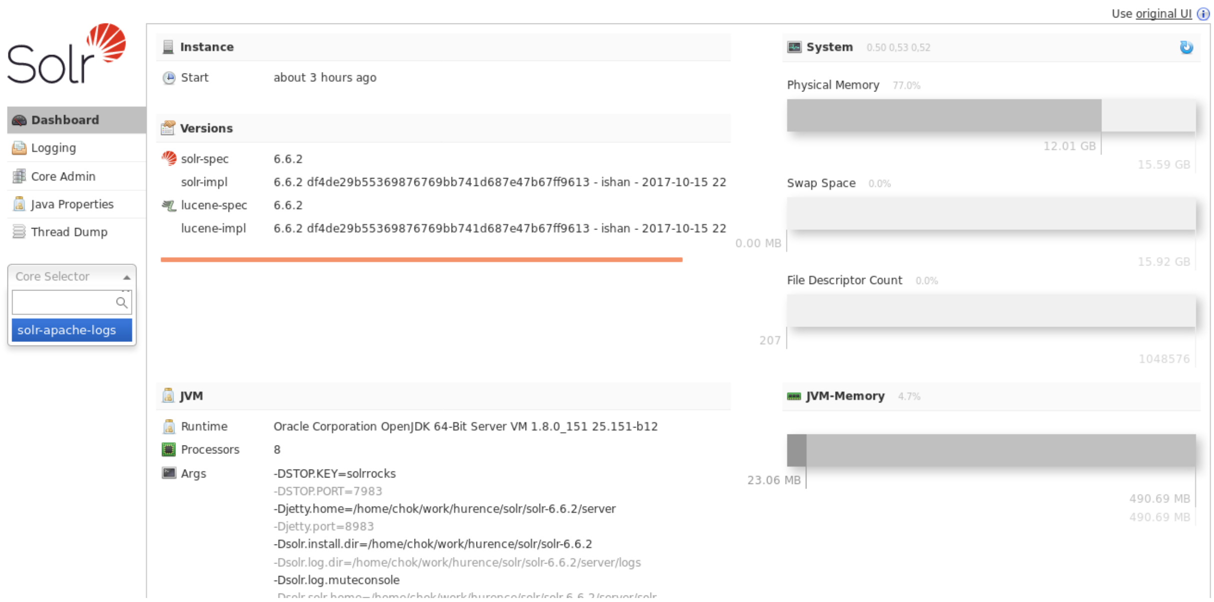


6. Inspect the logs

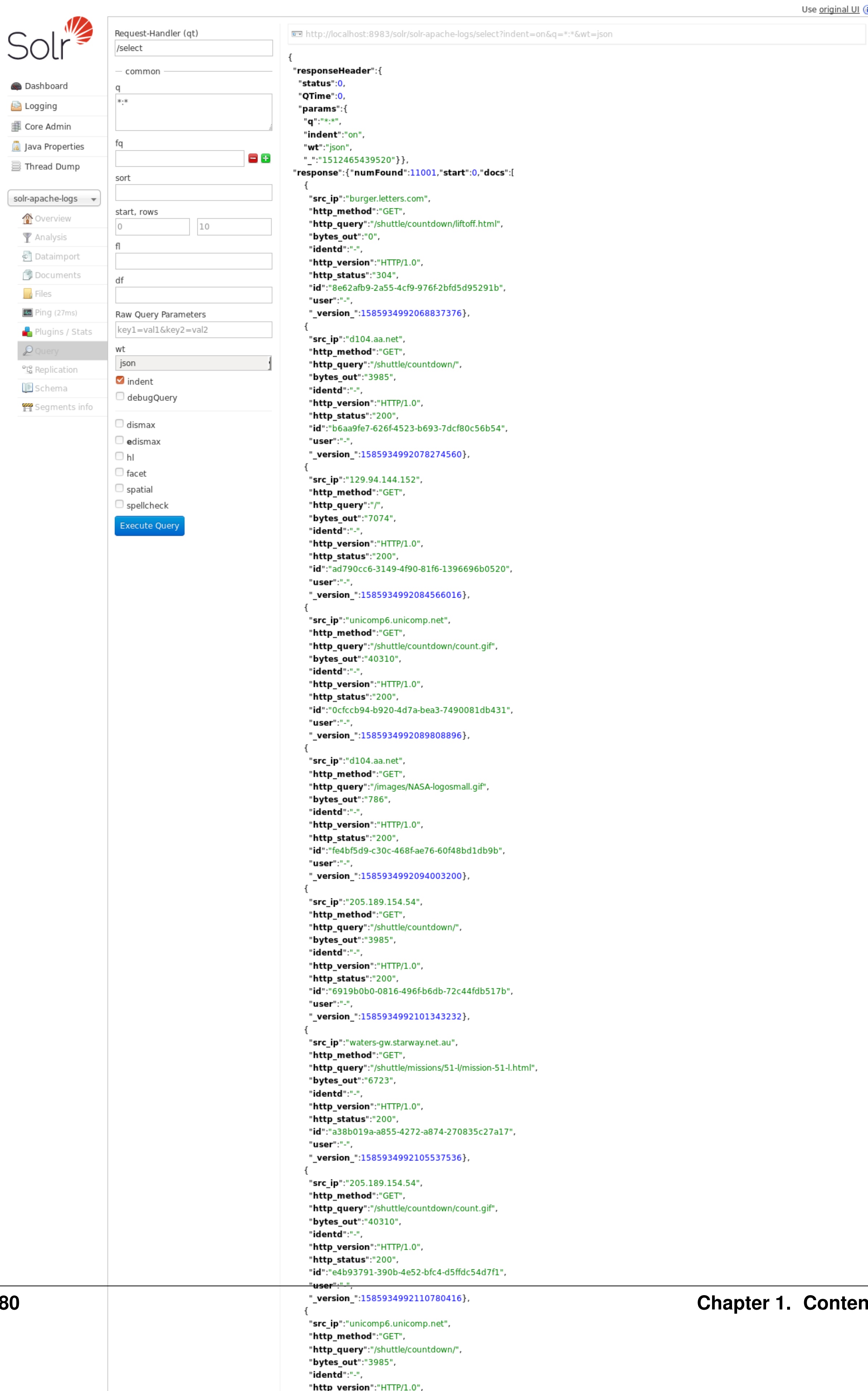
With Solr, you can directly use the solr web ui.

Open up your browser and go to http://localhost:8983/solr and you should be able to view your apache logs.

In non cloud mode, use the core selector, to select the core `` `solr-apache-logs` ``:



Then, go to query and by clicking to Execute Query, you will see some data from your Apache logs :


Use original UI
Solr
Dashboard
Logging
Core Admin
Java Properties
Thread Dump
solr-apache-logs
Overview
Analysis
Dataimport
Documents
Files
Ping (27ms)
Plugins / Stats
Query
Replication
Schema
Segments info
Request-Handler (qt)
/select
common
q
:
fq
sort
start, rows
0
10
fl
df
Raw Query Parameters
key1=val1&key2=val2
wt
json
indent
debugQuery
dismax
edismax
hl
facet
spatial
spellcheck
Execute Query
http://localhost:8983/solr/solr-apache-logs/select?indent=on&q=*:*&wt=json
{
"responseHeader":{
"status":0,
"QTime":0,
"params":{
"q":"*:*",
"indent":"on",
"wt":"json",
"_":"1512465439520"}},
"response":{"numFound":11001,"start":0,"docs":[
{
"src_ip":"burger.letters.com",
"http_method":"GET",
"http_query":"/shuttle/countdown/liftoff.html",
"bytes_out":"0",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"304",
"id":"8e62afb9-2a55-4cf9-976f-2bfd5d95291b",
"user":"-",
"_version_":1585934992068837376},
{
"src_ip":"d104.aa.net",
"http_method":"GET",
"http_query":"/shuttle/countdown/",
"bytes_out":"3985",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"b6aa9fe7-626f-4523-b693-7dcf80c56b54",
"user":"-",
"_version_":1585934992078274560},
{
"src_ip":"129.94.144.152",
"http_method":"GET",
"http_query":"/",
"bytes_out":"7074",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"ad790cc6-3149-4f90-81f6-1396696b0520",
"user":"-",
"_version_":1585934992084566016},
{
"src_ip":"unicomp6.unicomp.net",
"http_method":"GET",
"http_query":"/shuttle/countdown/count.gif",
"bytes_out":"40310",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"0cfccb94-b920-4d7a-bea3-7490081db431",
"user":"-",
"_version_":1585934992089808896},
{
"src_ip":"d104.aa.net",
"http_method":"GET",
"http_query":"/images/NASA-logosmall.gif",
"bytes_out":"786",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"fe4bf5d9-c30c-468f-ae76-60f48bd1db9b",
"user":"-",
"_version_":1585934992094003200},
{
"src_ip":"205.189.154.54",
"http_method":"GET",
"http_query":"/shuttle/countdown/",
"bytes_out":"3985",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"6919b0b0-0816-496f-b6db-72c44fdb517b",
"user":"-",
"_version_":1585934992101343232},
{
"src_ip":"waters-gw.starway.net.au",
"http_method":"GET",
"http_query":"/shuttle/missions/51-l/mission-51-l.html",
"bytes_out":"6723",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"a38b019a-a855-4272-a874-270835c27a17",
"user":"-",
"_version_":1585934992105537536},
{
"src_ip":"205.189.154.54",
"http_method":"GET",
"http_query":"/shuttle/countdown/count.gif",
"bytes_out":"40310",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"e4b93791-390b-4e52-bfc4-d5ffdc54d7f1",
"user":"-",
"_version_":1585934992110780416},
{
"src_ip":"unicomp6.unicomp.net",
"http_method":"GET",
"http_query":"/shuttle/countdown/",
"bytes_out":"3985",
"identd":"-",
"http_version":"HTTP/1.0",

3. Stop the job

You can Ctr+c the console where you launched logisland job. Then to kill all containers used run :

```
sudo docker-compose -f ./conf/docker-compose-index-apache-logs-solr.yml down
```

Make sure all container have disappeared.

```
sudo docker ps
```

1.8.6 Store Apache logs to Redis K/V store

In the following getting started tutorial we'll drive you through the process of Apache log mining with LogIsland platform.

Note: Be sure to know of to launch a logisland Docker environment by reading the prerequisites section

---

Note, it is possible to store data in different datastores. In this tutorial, we will see the case of Redis, if you need more in-depth explanations you can read the previous tutorial on indexing apache logs to elasticsearch or solr : **'index-apache-logs.html'_** .

1. Logisland job setup

The logisland job for this tutorial is already packaged in the tar.gz assembly and you can find it here :

```
docker exec -i -t logisland vim conf/store-to-redis.yml
```

We will start by explaining each part of the config file.

The *controllerServiceConfigurations* part is here to define all services that be shared by processors within the whole job, here a Redis KV cache service that will be used later in the `BulkPut` processor.

```yaml
- controllerService: datastore_service
  component: com.hurence.logisland.redis.service.RedisKeyValueCacheService
  type: service
  documentation: redis datastore service
  configuration:
    connection.string: localhost:6379
    redis.mode: standalone
    database.index: 0
    communication.timeout: 10 seconds
    pool.max.total: 8
    pool.max.idle: 8
    pool.min.idle: 0
    pool.block.when.exhausted: true
    pool.max.wait.time: 10 seconds
    pool.min.evictable.idle.time: 60 seconds
    pool.time.between.eviction.runs: 30 seconds
    pool.num.tests.per.eviction.run: -1
    pool.test.on.create: false
    pool.test.on.borrow: false
    pool.test.on.return: false
    pool.test.while.idle: true
    record.recordSerializer: com.hurence.logisland.serializer.JsonSerializer
```

Here the stream will read all the logs sent in `logisland_raw` topic and push the processing output into `logisland_events` topic.

Note: We want to specify an Avro output schema to validate our ouput records (and force their types accordingly). It's really for other streams to rely on a schema when processing records from a topic.

We can define some serializers to marshall all records from and to a topic.

```
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that converts raw apache logs into structured log records
  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
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
```

Within this stream a `SplitText` processor takes a log line as a String and computes a `Record` as a sequence of fields.

```
# parse apache logs
- processor: apache_parser
  component: com.hurence.logisland.processor.SplitText
  type: parser
  documentation: a parser that produce events from an apache log REGEX
  configuration:
    value.regex: (\S+)\s+(\S+)\s+(\S+)\s+\[([\w:\/]+\s[+\-]\d{4})\]\s+
↪"(\S+)\s+(\S+)\s*(\S*)"\s+(\S+)\s+(\S+)
    value.fields: src_ip,identd,user,record_time,http_method,http_query,http_version,
↪http_status,bytes_out
```

This stream will process log entries as soon as they will be queued into *logisland_raw* Kafka topics, each log will be parsed as an event which will be pushed back to Kafka in the `logisland_events` topic.

The second processor will handle `Records` produced by the `SplitText` to index them into datastore previously defined (Redis)

```
# all the parsed records are added to datastore by bulk
- processor: datastore_publisher
  component: com.hurence.logisland.processor.datastore.BulkPut
  type: processor
  documentation: "indexes processed events in datastore"
  configuration:
    datastore.client.service: datastore_service
```

2. Launch the script

For this tutorial we will handle some apache logs with a splitText parser and send them to Redis Connect a shell to your logisland container to launch the following streaming jobs.

For ElasticSearch :

```
docker exec -i -t logisland bin/logisland.sh --conf conf/store-to-redis.yml
```

3. Inject some Apache logs into the system

Now we're going to send some logs to `logisland_raw` Kafka topic.

We could setup a logstash or flume agent to load some apache logs into a kafka topic but there's a super useful tool in the Kafka ecosystem : kafkacat, a *generic command line non-JVM Apache Kafka producer and consumer* which can be easily installed.

If you don't have your own httpd logs available, you can use some freely available log files from NASA-HTTP web site access:

- Jul 01 to Jul 31, ASCII format, 20.7 MB gzip compressed
- Aug 04 to Aug 31, ASCII format, 21.8 MB gzip compressed

Let's send the first 500000 lines of NASA http access over July 1995 to LogIsland with kafkacat to `logisland_raw` Kafka topic

```
cd /tmp
wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz
gunzip NASA_access_log_Jul95.gz
head -500000 NASA_access_log_Jul95 | kafkacat -b sandbox:9092 -t logisland_raw
```

4. Inspect the logs

For this part of the tutorial we will use redis-py a Python client for Redis. You can install it by following instructions given on redis-py.

To install redis-py, simply:

```
$ sudo pip install redis
```

Getting Started, check if you can connect with Redis

```
>>> import redis
>>> r = redis.StrictRedis(host='localhost', port=6379, db=0)
>>> r.set('foo', 'bar')
>>> r.get('foo')
```

Then we want to grab some logs that have been collected to Redis. We first find some keys with a pattern and get the json content of one

```
>>> r.keys('1234*')
```

['123493eb-93df-4e57-a1c1-4a8e844fa92c', '123457d5-8ccc-4f0f-b4ba-d70967aa48eb', '12345e06-6d72-4ce8-8254-a7cc4bab5e31']

```
>>> r.get('123493eb-93df-4e57-a1c1-4a8e844fa92c')
```

'{n "id" : "123493eb-93df-4e57-a1c1-4a8e844fa92c",n "type" : "apache_log",n "creationDate" : 804574829000,n "fields" : {n "src_ip" : "204.191.209.4",n "record_id" : "123493eb-93df-4e57-a1c1-4a8e844fa92c",n "http_method" : "GET",n "http_query" : "/images/WORLD-logosmall.gif",n "bytes_out" : "669",n "identd" : "-",n "http_version" : "HTTP/1.0",n "record_raw_value" : "204.191.209.4 - - [01/Jul/1995:01:00:29 -0400] "GET /images/WORLD-logosmall.gif HTTP/1.0" 200 669",n "http_status" : "200",n "record_time" : 804574829000,n "user" : "-",n "record_type" : "apache_log"n }n}'

```
>>> import json
>>> record = json.loads(r.get('123493eb-93df-4e57-a1c1-4a8e844fa92c'))
>>> record['fields']['bytes_out']
```

1.8.7 Threshold based alerting on Apache logs with Redis K/V store

In a previous tutorial we saw how to use Redis K/V store as a cache storage. In this one we will practice the use of *ComputeTag*, *CheckThresholds* and *CheckAlerts* processor in conjunction with this Redis Cache.

The following job is made of 2 streaming parts :

1. A main stream which parses Apache logs and store them to a Redis cache .
2. A timer based stream which compute some new tags values based on cached records, check some thresholds cross and send alerts if needed.

Note: Be sure to know of to launch a logisland Docker environment by reading the prerequisites section

---

The full logisland job for this tutorial is already packaged in the tar.gz assembly and you can find it here :

```
docker exec -i -t conf_logisland_1 vim conf/threshold-alerting.yml
```

We will start by explaining each part of the config file.

1. Controller service part

The *controllerServiceConfigurations* part is here to define all services that be shared by processors within the whole job, here a Redis KV cache service that will be used later in the `BulkPut` processor.

```
- controllerService: datastore_service
  component: com.hurence.logisland.redis.service.RedisKeyValueCacheService
  type: service
  documentation: redis datastore service
  configuration:
    connection.string: localhost:6379
    redis.mode: standalone
    database.index: 0
    communication.timeout: 10 seconds
    pool.max.total: 8
    pool.max.idle: 8
    pool.min.idle: 0
    pool.block.when.exhausted: true
    pool.max.wait.time: 10 seconds
    pool.min.evictable.idle.time: 60 seconds
```

(continues on next page)

(continued from previous page)

```
    pool.time.between.eviction.runs: 30 seconds
    pool.num.tests.per.eviction.run: -1
    pool.test.on.create: false
    pool.test.on.borrow: false
    pool.test.on.return: false
    pool.test.while.idle: true
    record.recordSerializer: com.hurence.logisland.serializer.JsonSerializer
```

2. First stream : parse logs and compute tags

Here the stream will read all the logs sent in `logisland_raw` topic and push the processing output into `logisland_events` topic as Json serialized records.

```
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that converts raw apache logs into structured log records
  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
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
```

Within this stream a `SplitText` processor takes a log line as a String and computes a `Record` as a sequence of fields.

```
- processor: apache_parser
  component: com.hurence.logisland.processor.SplitText
  type: parser
  documentation: a parser that produce events from an apache log REGEX
  configuration:
    value.regex: (\S+)\s+(\S+)\s+(\S+)\s+\[([\w:\/]+\s[+\-]\d{4})\]\s+
↪"(\S+)\s+(\S+)\s*(\S*)"\s+(\S+)\s+(\S+)
    value.fields: src_ip,identd,user,record_time,http_method,http_query,http_version,
↪http_status,bytes_out
```

This stream will process log entries as soon as they will be queued into *logisland_raw* Kafka topics, each log will be parsed as an event which will be pushed back to Kafka in the `logisland_events` topic.

the next processing step is to assign *bytes_out* field as *record_value*

```
- processor: normalize_fields
  component: com.hurence.logisland.processor.NormalizeFields
  type: parser
  documentation: change field name 'bytes_out' to `record_value`
  configuration:
    conflict.resolution.policy: overwrite_existing
    record_value: bytes_out
```

the we modify *record_id* to set its value as *src_ip* field.

```
- processor: modify_id
  component: com.hurence.logisland.processor.ModifyId
  type: parser
  documentation: change current id to src_ip
  configuration:
    id.generation.strategy: fromFields
    fields.to.hash: src_ip
    java.formatter.string: "%1$s"
```

now we'll remove all the unwanted fields

```
- processor: remove_fields
  component: com.hurence.logisland.processor.RemoveFields
  type: parser
  documentation: remove useless fields
  configuration:
    fields.to.remove: src_ip,identd,user,http_method,http_query,http_version,http_
↪status,bytes_out
```

and then cast *record_value* as a double

```
- processor: cast
  component: com.hurence.logisland.processor.ConvertFieldsType
  type: parser
  documentation: cast values
  configuration:
    record_value: double
```

The next processing step wil compute a dynamic Tag value from a Javascript expression. Here a new record with an *record_id* set to *computed1* and as a *record_value* the resulting expression of *cache("logisland.hurence.com").value * 10.2*

```
- processor: compute_tag
  component: com.hurence.logisland.processor.alerting.ComputeTags
  type: processor
  documentation: |
    compute tags from given formulas.
    each dynamic property will return a new record according to the formula definition
    the record name will be set to the property name
    the record time will be set to the current timestamp
  configuration:
    datastore.client.service: datastore_service
    output.record.type: computed_tag
    max.cpu.time: 500
    max.memory: 64800000
    max.prepared.statements: 5
    allow.no.brace: false
    computed1: return cache("logisland.hurence.com").value * 10.2;
```

The last processor will handle all the `Records` of this stream to index them into datastore previously defined (Redis)

```
# all the parsed records are added to datastore by bulk
- processor: datastore_publisher
  component: com.hurence.logisland.processor.datastore.BulkPut
  type: processor
  documentation: "indexes processed events in datastore"
```

(continues on next page)

(continued from previous page)

```yaml
configuration:
  datastore.client.service: datastore_service
```

3. Second stream : check threshold cross and alerting

The second stream will read all the logs sent in `logisland_events` topic and push the processed outputs (threshold_cross & alerts records) into `logisland_alerts` topic as Json serialized records.

We won't comment the stream definition as it is really straightforward.

The first processor of this stream pipeline makes use of *CheckThresholds* component which will add a new record of type *threshold_cross* with a *record_id* set to *threshold1* if the JS expression *cache("computed1").value > 2000.0* is evaluated to true.

```yaml
- processor: compute_thresholds
  component: com.hurence.logisland.processor.alerting.CheckThresholds
  type: processor
  documentation: |
    compute threshold cross from given formulas.
    each dynamic property will return a new record according to the formula definition
    the record name will be set to the property name
    the record time will be set to the current timestamp

    a threshold_cross has the following properties : count, time, duration, value
  configuration:
    datastore.client.service: datastore_service
    output.record.type: threshold_cross
    max.cpu.time: 100
    max.memory: 12800000
    max.prepared.statements: 5
    record.ttl: 300000
    threshold1: cache("computed1").value > 2000.0
```

```yaml
- processor: compute_alerts1
  component: com.hurence.logisland.processor.alerting.CheckAlerts
  type: processor
  documentation: |
    compute threshold cross from given formulas.
    each dynamic property will return a new record according to the formula definition
    the record name will be set to the property name
    the record time will be set to the current timestamp
  configuration:
    datastore.client.service: datastore_service
    output.record.type: medium_alert
    alert.criticity: 1
    max.cpu.time: 100
    max.memory: 12800000
    max.prepared.statements: 5
    profile.activation.condition: cache("threshold1").value > 3000.0
    alert1: cache("threshold1").duration > 50.0
```

The last processor will handle all the `Records` of this stream to index them into datastore previously defined (Redis)

```yaml
- processor: datastore_publisher
  component: com.hurence.logisland.processor.datastore.BulkPut
```

(continues on next page)

(continued from previous page)

```
  type: processor
  documentation: "indexes processed events in datastore"
  configuration:
    datastore.client.service: datastore_service
```

4. Launch the script

Connect a shell to your logisland container to launch the following streaming jobs.

```
docker exec -i -t conf_logisland_1 bin/logisland.sh --conf conf/threshold-alerting.yml
```

5. Inject some Apache logs into the system

Now we're going to send some logs to `logisland_raw` Kafka topic.

We could setup a logstash or flume agent to load some apache logs into a kafka topic but there's a super useful tool in the Kafka ecosystem : kafkacat, a *generic command line non-JVM Apache Kafka producer and consumer* which can be easily installed.

If you don't have your own httpd logs available, you can use some freely available log files from NASA-HTTP web site access:

- Jul 01 to Jul 31, ASCII format, 20.7 MB gzip compressed
- Aug 04 to Aug 31, ASCII format, 21.8 MB gzip compressed

Let's send the first 500000 lines of NASA http access over July 1995 to LogIsland with kafkacat to `logisland_raw` Kafka topic

```
cd /tmp
wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz
gunzip NASA_access_log_Jul95.gz
head -500000 NASA_access_log_Jul95 | kafkacat -b sandbox:9092 -t logisland_raw
```

6. Inspect the logs and alerts

For this part of the tutorial we will use redis-py a Python client for Redis. You can install it by following instructions given on redis-py.

To install redis-py, simply:

```
$ sudo pip install redis
```

Getting Started, check if you can connect with Redis

```
>>> import redis
>>> r = redis.StrictRedis(host='localhost', port=6379, db=0)
>>> r.set('foo', 'bar')
>>> r.get('foo')
```

Then we want to grab some logs that have been collected to Redis. We first find some keys with a pattern and get the json content of one

```
>>> r.keys('1234*')
```

['123493eb-93df-4e57-a1c1-4a8e844fa92c', '123457d5-8ccc-4f0f-b4ba-d70967aa48eb', '12345e06-6d72-4ce8-8254-a7cc4bab5e31']

```
>>> r.get('123493eb-93df-4e57-a1c1-4a8e844fa92c')
```

'{n "id" : "123493eb-93df-4e57-a1c1-4a8e844fa92c",n "type" : "apache_log",n "creationDate" : 804574829000,n "fields" : {n "src_ip" : "204.191.209.4",n "record_id" : "123493eb-93df-4e57-a1c1-4a8e844fa92c",n "http_method" : "GET",n "http_query" : "/images/WORLD-logosmall.gif",n "bytes_out" : "669",n "identd" : "-",n "http_version" : "HTTP/1.0",n "record_raw_value" : "204.191.209.4 - - [01/Jul/1995:01:00:29 -0400] "GET /images/WORLD-logosmall.gif HTTP/1.0" 200 669",n "http_status" : "200",n "record_time" : 804574829000,n "user" : "-",n "record_type" : "apache_log"n }n}'

```
>>> import json
>>> record = json.loads(r.get('123493eb-93df-4e57-a1c1-4a8e844fa92c'))
>>> record['fields']['bytes_out']
```

1.8.8 Alerting & Query Matching

In the following tutorial we'll learn how to raise custom alerts on some http traffic (apache log records) based on lucene matching query criterion.

We assume that you already know how to parse and ingest Apache logs into logisland. If it's not the case please refer to the previous Apache logs indexing tutorial. We will use mainly the MatchQuery Processor.

Note: Be sure to know of to launch a logisland Docker environment by reading the prerequisites section

1.Install required components

For this tutorial please make sure to already have installed elasticsearch modules.

If not you can just do it through the components.sh command line:

```
bin/components.sh -i com.hurence.logisland:logisland-processor-elasticsearch:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-service-elasticsearch_5_4_0-
↪client:1.1.1
```

2. Logisland job setup

The logisland job for this tutorial is already packaged in the tar.gz assembly and you can find it here :

```
docker exec -i -t logisland vim conf/match-queries.yml
```

We will start by explaining each part of the config file.

The stream contains two processors quite identical (the first one converts raw logs to records and the second one index records to ES) to those encountered in the previous Apache logs indexing tutorial tutorial .

The third one makes use of the MatchQuery Processor. This processor provides user with dynamic query registration. This queries are expressed in the Lucene syntax.

Note: Please read the Lucene syntax guide for supported operations.

This processor will tag the record with `blacklisted_host` field if the query `src_ip:(+alyssa +prodigy)` matches and tag `montana_host` if `src_ip:montana`

```yaml
- processor: match_query
  component: com.hurence.logisland.processor.MatchQuery
  type: processor
  documentation: a parser that matches lucene queries on records
  configuration:
    policy.onmiss: forward
    policy.onmatch: all
    blacklisted_host: src_ip:(+alyssa +prodigy)
    montana_host: src_ip:montana
```

here is an example of matching record :


```json
{
    "@timestamp": "1995-07-01T09:02:18+02:00",
    "alert_match_name": [
      "montana_host"
    ],
    "alert_match_query": [
      "src_ip:montana"
    ],
    "bytes_out": "8677",
    "http_method": "GET",
    "http_query": "/shuttle/missions/missions.html",
    "http_status": "200",
    "http_version": "HTTP/1.0",
    "identd": "-",
    "record_id": "8e861956-af54-49fd-9043-94c143fc5a19",
    "record_raw_value": "ril.usda.montana.edu - - [01/Jul/1995:03:02:18 -0400] \"GET /
↪shuttle/missions/missions.html HTTP/1.0\" 200 8677",
    "record_time": 804582138000,
    "record_type": "apache_log",
    "src_ip": "ril.usda.montana.edu",
    "user": "-"
  }
```


3. Launch the script

For this tutorial we will handle some apache logs with a splitText parser and send them to Elastiscearch Connect a shell to your logisland container to launch the following streaming jobs.

```
docker exec -i -t logisland bin/logisland.sh --conf conf/match-queries.yml
```

4. Inject some Apache logs into the system

Now we're going to send some logs to `logisland_raw` Kafka topic.

We could setup a logstash or flume agent to load some apache logs into a kafka topic but there's a super useful tool in the Kafka ecosystem : kafkacat, a *generic command line non-JVM Apache Kafka producer and consumer* which can be easily installed.

If you don't have your own httpd logs available, you can use some freely available log files from NASA-HTTP web site access:

- Jul 01 to Jul 31, ASCII format, 20.7 MB gzip compressed
- Aug 04 to Aug 31, ASCII format, 21.8 MB gzip compressed

Let's send the first 500000 lines of NASA http access over July 1995 to LogIsland with kafkacat to `logisland_raw` Kafka topic

```
cd /tmp
wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz
gunzip NASA_access_log_Jul95.gz
head -500000 NASA_access_log_Jul95 | kafkacat -b sandbox:9092 -t logisland_raw
```

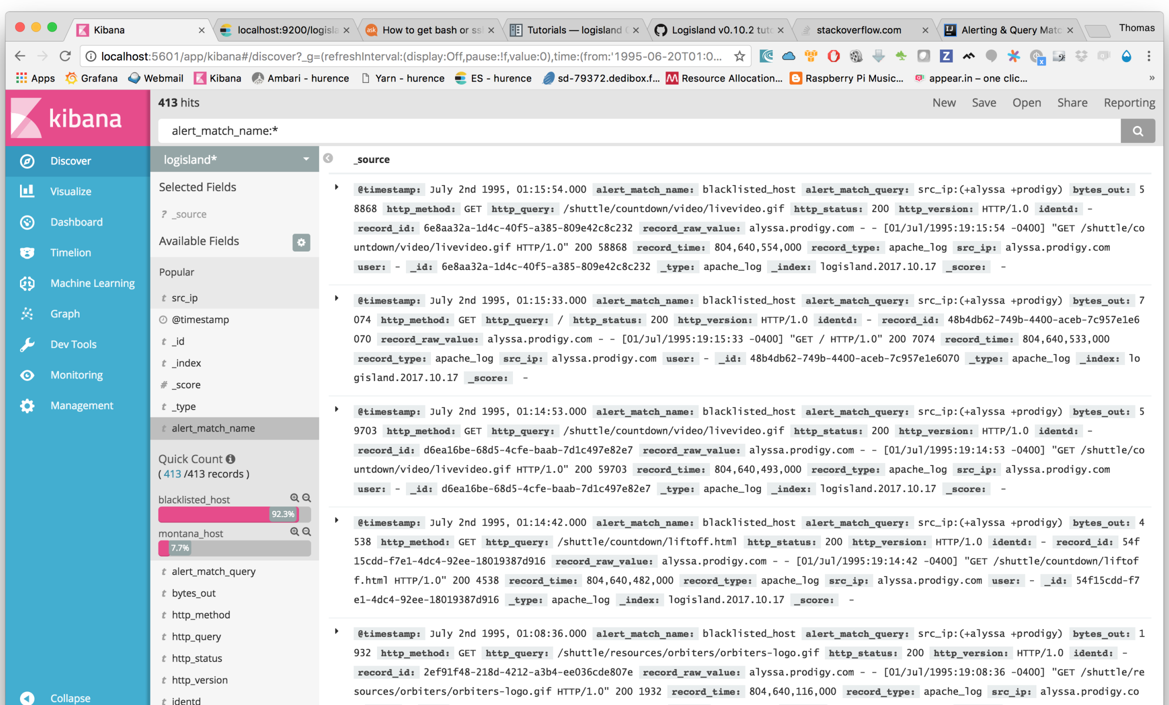
5. Check your alerts with Kibana

Check that you've match some criterias :

```
curl -XGET http://localhost:9200/logisland.2017.10.17/_search?pretty&q=alert_match_
↪name:montana_host
curl -XGET http://localhost:9200/logisland.2017.10.17/_search?pretty&q=alert_match_
↪name:blacklisted_host
```

Open up your browser and go to http://sandbox:5601/ and you should be able to explore your apache logs.

by adding filter on `alert_match_name:blacklisted_host` you'll only get request from `alyssa.prodigy.com` which is a host we where monitoring.



1.8.9 Event aggregation

In the following tutorial we'll learn how to generate time window metrics on some http traffic (apache log records) and how to raise custom alerts based on lucene matching query criterion.

We assume that you already know how to parse and ingest Apache logs into logisland. If it's not the case please refer to the previous Apache logs indexing tutorial. We will first add an SQLAggregator Stream to compute some metrics and then add a MatchQuery Processor.

Note: Be sure to know of to launch a logisland Docker environment by reading the prerequisites section

1.Install required components

For this tutorial please make sure to already have installed elasticsearch modules. If not you can just do it through the componentes.sh command line:

```
bin/components.sh -i com.hurence.logisland:logisland-processor-elasticsearch:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-service-elasticsearch_5_4_0-
↪client:1.1.1
```

2. Logisland job setup

The logisland job for this tutorial is already packaged in the tar.gz assembly and you can find it here :

```
docker exec -i -t logisland vim conf/aggregate-events.yml
```

We will start by explaining each part of the config file.

Our application will be composed of 4 streams :

The first one converts apache logs to typed records (please note the use of `ConvertFieldsType` processor)

The second one is the sql stream is a special one one use a KafkaRecordStreamSQLAggregator. This stream defines input/output topics names as well as Serializers, avro schema.

Note: The Avro schema is set for the input topic and must be same as the avro schema of the output topic for the stream that produces the records (please refer to Index Apache logs tutorial

The most important part of the *KafkaRecordStreamSQLAggregator* is its *sql.query* property which defines a query to apply on the incoming records for the given time window.

The following SQL query will be applied on sliding window of 10" of records.

```
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
```

which will consider `logisland_events` topic as SQL table and create 20 output Record with the fields avg_bytes_out, src_ip & record_time. the statement with record_time will ensure that the created Records will correspond to the effective input event time (not the actual time).

```
- stream: metrics_by_host
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamSQLAggregator
  type: stream
  documentation: a processor that links
  configuration:
    kafka.input.topics: logisland_events
    kafka.output.topics: logisland_aggregations
    kafka.error.topics: logisland_errors
    kafka.input.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 2
    kafka.topic.default.replicationFactor: 1
    window.duration: 10
    avro.input.schema: >
      {  "version":1,
         "type": "record",
         "name": "com.hurence.logisland.record.apache_log",
         "fields": [
           { "name": "record_errors",   "type": [ {"type": "array", "items": "string"}
↪,"null"] },
           { "name": "record_raw_key", "type": ["string","null"] },
           { "name": "record_raw_value", "type": ["string","null"] },
           { "name": "record_id",   "type": ["string"] },
           { "name": "record_time", "type": ["long"] },
           { "name": "record_type", "type": ["string"] },
           { "name": "src_ip",      "type": ["string","null"] },
           { "name": "http_method", "type": ["string","null"] },
           { "name": "bytes_out",   "type": ["long","null"] },
           { "name": "http_query",  "type": ["string","null"] },
           { "name": "http_version","type": ["string","null"] },
           { "name": "http_status", "type": ["string","null"] },
           { "name": "identd",      "type": ["string","null"] },
           { "name": "user",        "type": ["string","null"] }    ]}
    sql.query: >
      SELECT count(*) AS connections_count, avg(bytes_out) AS avg_bytes_out, src_ip
      FROM logisland_events
      GROUP BY src_ip
      ORDER BY event_count DESC
      LIMIT 20
    max.results.count: 1000
    output.record.type: top_client_metrics
```

Here we will compute every x seconds, the top twenty *src_ip* for connections count. The result of the query will be pushed into to *logisland_aggregations* topic as new *top_client_metrics* Record containing *connections_count* and *avg_bytes_out* fields.

the third match some criteria to send some alerts

```
- processor: match_query
  component: com.hurence.logisland.processor.MatchQuery
  type: processor
  documentation: a parser that produce alerts from lucene queries
  configuration:
```

(continues on next page)

(continued from previous page)

```
    numeric.fields: bytes_out,connections_count
    too_much_bandwidth: avg_bytes_out:[25000 TO 5000000]
    too_many_connections: connections_count:[150 TO 300]
    output.record.type: threshold_alert
```

3. Launch the script

For this tutorial we will handle some apache logs with a splitText parser and send them to Elastiscearch Connect a shell to your logisland container to launch the following streaming jobs.

```
docker exec -i -t logisland bin/logisland.sh --conf conf/aggregate-events.yml
```

4. Inject some Apache logs into the system

Now we're going to send some logs to `logisland_raw` Kafka topic.

We could setup a logstash or flume agent to load some apache logs into a kafka topic but there's a super useful tool in the Kafka ecosystem : kafkacat, a *generic command line non-JVM Apache Kafka producer and consumer* which can be easily installed.

If you don't have your own httpd logs available, you can use some freely available log files from NASA-HTTP web site access:

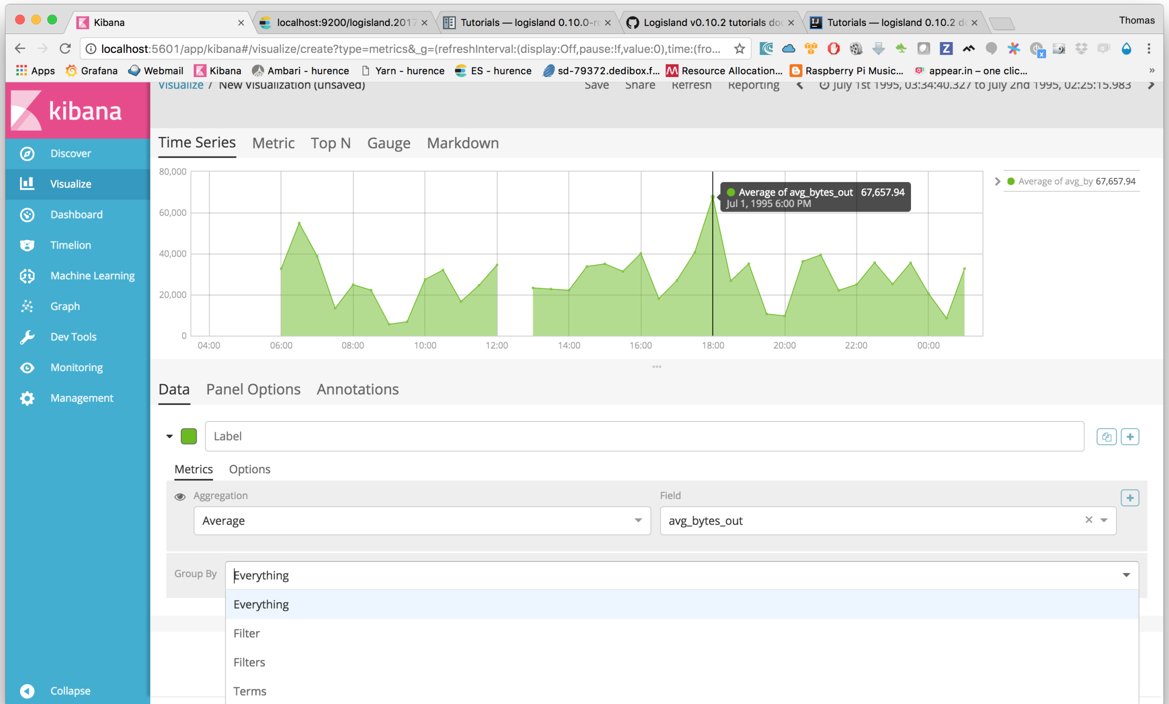
- Jul 01 to Jul 31, ASCII format, 20.7 MB gzip compressed
- Aug 04 to Aug 31, ASCII format, 21.8 MB gzip compressed

Let's send the first 500000 lines of NASA http access over July 1995 to LogIsland with kafkacat to `logisland_raw` Kafka topic

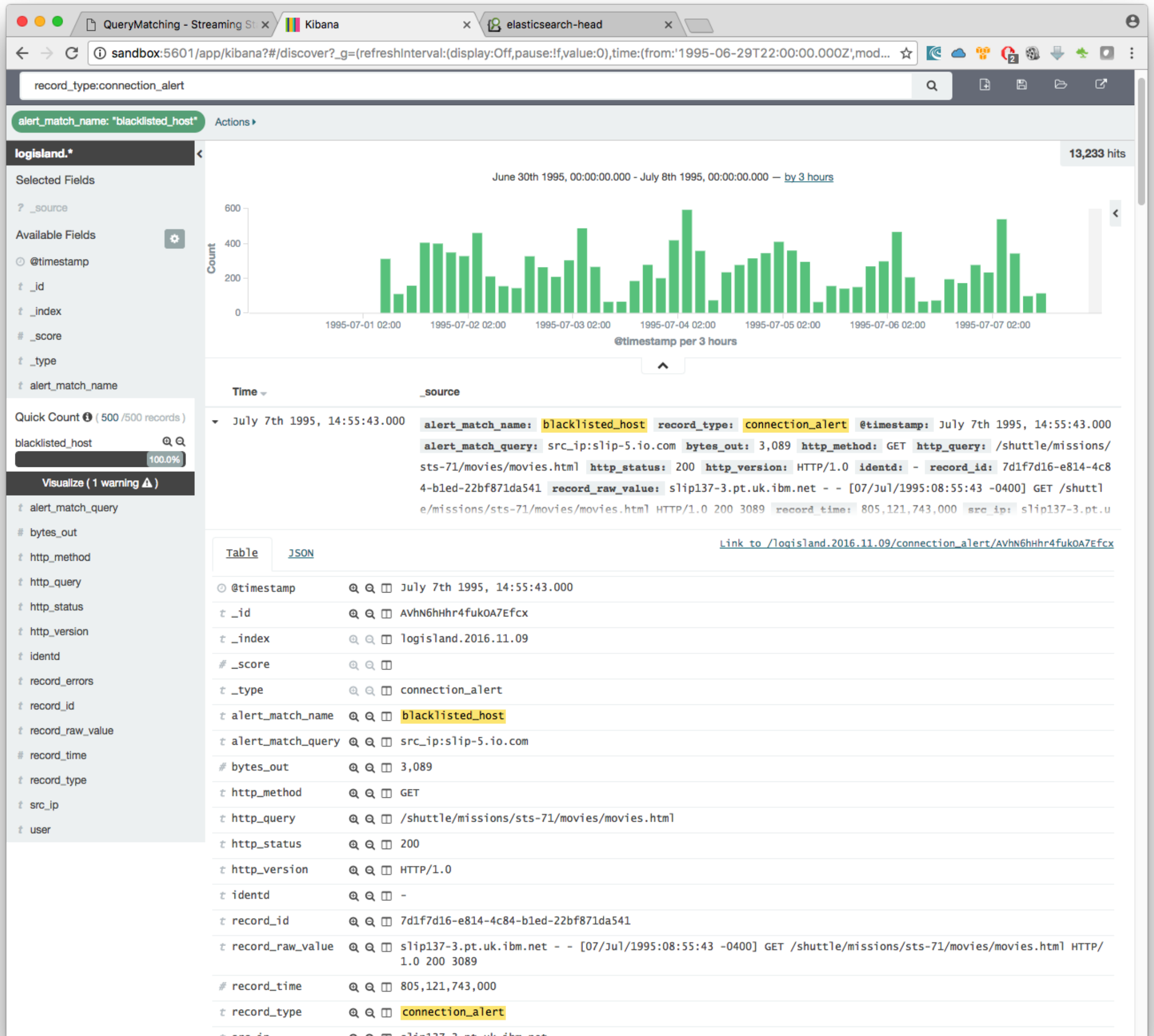
```
cd /tmp
wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz
gunzip NASA_access_log_Jul95.gz
head -500000 NASA_access_log_Jul95 | kafkacat -b sandbox:9092 -t logisland_raw
```

5. Check your alerts with Kibana

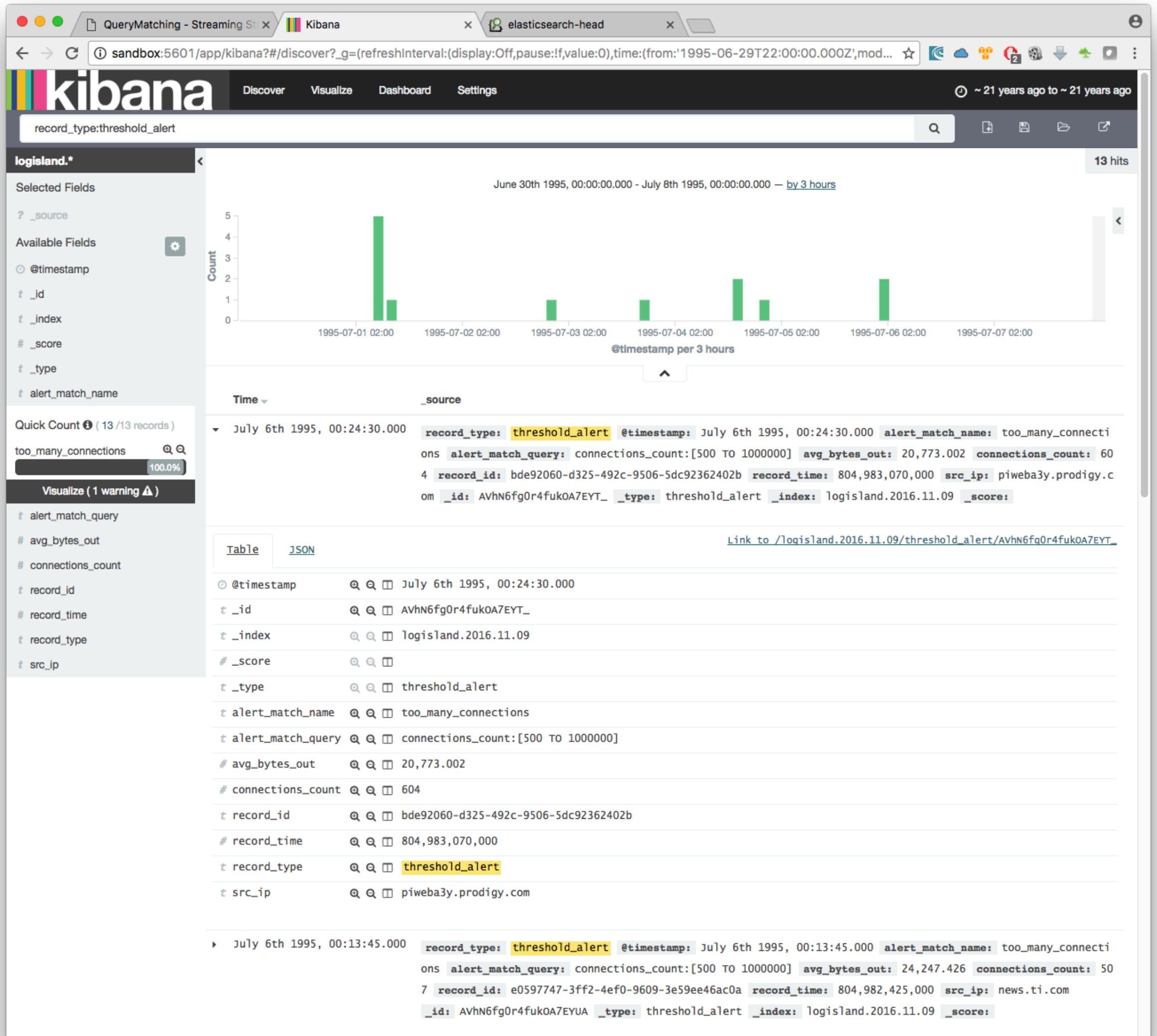
As we explore data logs from july 1995 we'll have to select an absolute time filter from 1995-06-30 to 1995-07-08 to see the events.



you can filter your events with `record_type:connection_alert` to get 71733 connections alerts matching your query



if we filter now on threshold alerts whith `record_type:threshold_alert` you'll get the 13 src_ip that have been catched by the threshold query.



1.8.10 Index Apache logs Enrichment

In the following tutorial we'll drive you through the process of enriching Apache logs with LogIsland platform.

One of the first steps when treating web access logs is to extract information from the User-Agent header string, in order to be able to classify traffic. The User-Agent string is part of the access logs from the web server (this is the last field in the example below).

Another step is to find the FQDN (full qualified domain name) from an ip address.

That string is packed with information from the visitor, when you know how to interpret it. However, the User-Agent string is not based on any standard, and it is not trivial to extract meaningful information from it. LogIsland provides a processor, based on the YAUAA library, that simplifies that treatement.

LogIsland provides a processor, based on InetAdress class from JDK 8, that use reverse Dns to determine FQDN from an IP.

Note: This class find FQDN from ip using IN-ADDR.ARPA (or IP6.ARPA for ipv6). If it finds a domain name, it verifies that it matches back the same address ip in order to prevent against IP spoofing attack. If you want to return the ip anyway, you should implement a new plugin using another library as dnsjava for example or open an issue for asking this feature.

We will reuse the Docker container hosting all the LogIsland services from the previous tutorial, and add the User-Agent as well as the IpToFqdn processor to the stream

Note: You can download the latest release of logisland and the YAML configuration file for this tutorial which can be also found under *$LOGISLAND_HOME/conf* directory.

1. Start LogIsland as a Docker container

LogIsland is packaged as a Docker container that you can build yourself or pull from Docker Hub.

You can find the steps to start the Docker image and start the LogIsland server in the previous tutorial. However, you'll start the server with a different configuration file (that already includes the necessary modifications)

Install required components

For this tutorial please make sure to already have installed required modules.

If not you can just do it through the components.sh command line:

```
bin/components.sh -i com.hurence.logisland:logisland-processor-elasticsearch:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-service-elasticsearch_2_4_0-
↪client:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-processor-enrichment:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-processor-useragent:1.1.1
```

Stream 1 : modify the stream to analyze the User-Agent string

Note: You can either apply the modifications from this section to the file *conf/index-apache-logs.yml* ot directly use the file *conf/enrich-apache-logs.yml* that already includes them.

The stream needs to be modified to

```
* modify the regex to add the referer and the User-Agent strings for the SplitText␣
↪processor
* modify the Avro schema to include the new fields returned by the UserAgentProcessor
* include the processing of the User-Agent string after the parsing of the logs
* include the processor IpToFqdn after the ParserUserAgent
* include a cache service to use with IpToFqdn processor
```

The example below shows how to include all of the fields supported by the processor.

Note: It is possible to remove unwanted fields from both the processor configuration and the Avro schema

Once the configuration file is updated, LogIsland must be restarted with that new configuration file.

```
bin/logisland.sh --conf <new_configuration_file>
```

2. Inject some Apache logs into the system

Now we're going to send some logs to `logisland_raw` Kafka topic.

We could setup a logstash or flume agent to load some apache logs into a kafka topic but there's a super useful tool in the Kafka ecosystem : kafkacat, a *generic command line non-JVM Apache Kafka producer and consumer* which can be easily installed (and is already present in the docker image).

If you don't have your own httpd logs available, you can use some freely available log files from Elastic web site

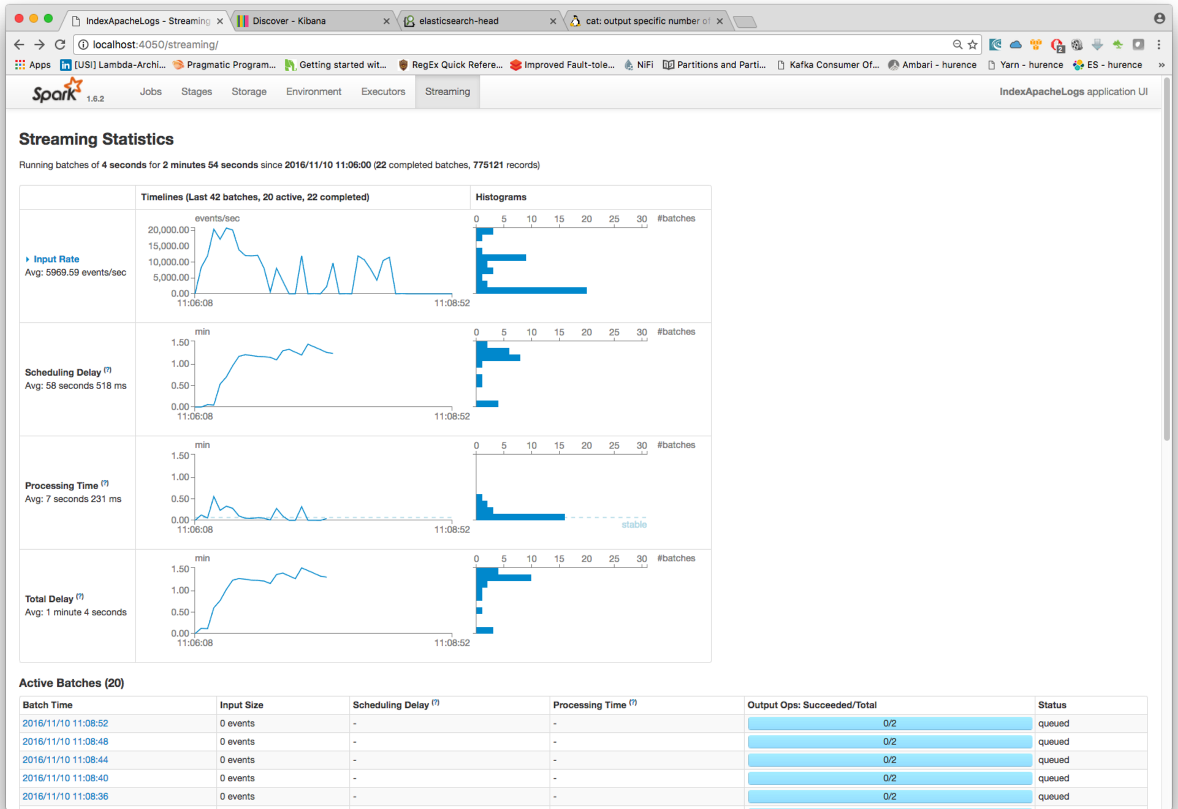
Let's send the first 500000 lines of access log to LogIsland with kafkacat to `logisland_raw` Kafka topic

```
docker exec -ti logisland bash
cd /tmp
wget https://raw.githubusercontent.com/elastic/examples/master/ElasticStack_apache/
↪apache_logs
head -500000 apache_logs | kafkacat -b sandbox:9092 -t logisland_raw
```

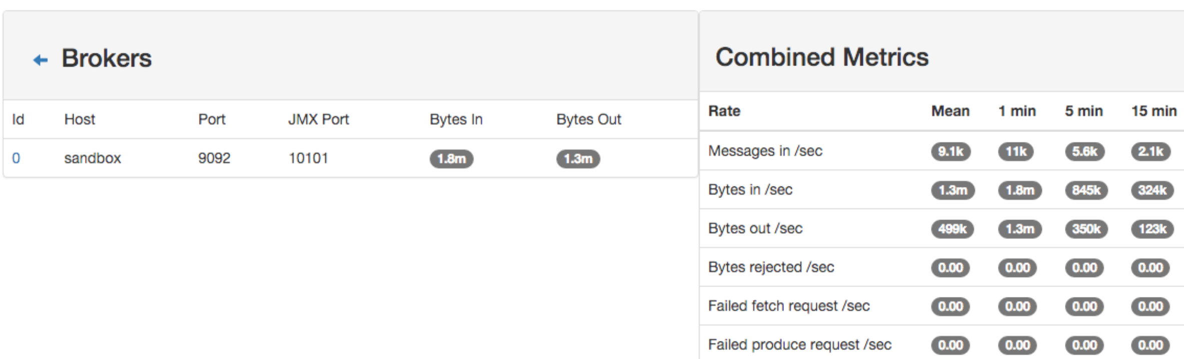
Note: The process should last around 280 seconds because reverse dns is a costly operation. After all data are processed, you can inject the same logs again and it should be very fast to process thanks to the cache that saved all matched ip.

3. Monitor your spark jobs and Kafka topics

Now go to http://sandbox:4050/streaming/ to see how fast Spark can process your data



Another tool can help you to tweak and monitor your processing http://sandbox:9000/



Brokers

Id	Host	Port	JMX Port	Bytes In	Bytes Out
0	sandbox	9092	10101	1.8m	1.3m

Combined Metrics

Rate	Mean	1 min	5 min	15 min
Messages in /sec	9.1k	11k	5.6k	2.1k
Bytes in /sec	1.3m	1.8m	845k	324k
Bytes out /sec	499k	1.3m	350k	123k
Bytes rejected /sec	0.00	0.00	0.00	0.00
Failed fetch request /sec	0.00	0.00	0.00	0.00
Failed produce request /sec	0.00	0.00	0.00	0.00

4. Use Kibana to inspect the logs

You've completed the enrichment of your logs using the User-Agent processor. The logs are now loaded into elasticSearch, in the following form :

```
curl -XGET http://localhost:9200/logisland.*/_search?pretty
```

```
{

    "_index": "logisland.2017.03.21",
    "_type": "apache_log",
    "_id": "4ca6a8b5-1a60-421e-9ae9-6c30330e497e",
    "_score": 1.0,
    "_source": {
        "@timestamp": "2015-05-17T10:05:43Z",
        "agentbuild": "Unknown",
        "agentclass": "Browser",
        "agentinformationemail": "Unknown",
        "agentinformationurl": "Unknown",
        "agentlanguage": "Unknown",
        "agentlanguagecode": "Unknown",
        "agentname": "Chrome",
        "agentnameversion": "Chrome 32.0.1700.77",
        "agentnameversionmajor": "Chrome 32",
        "agentsecurity": "Unknown",
        "agentuuid": "Unknown",
        "agentversion": "32.0.1700.77",
        "agentversionmajor": "32",
        "anonymized": "Unknown",
        "devicebrand": "Apple",
        "deviceclass": "Desktop",
        "devicecpu": "Intel",
        "devicefirmwareversion": "Unknown",
        "devicename": "Apple Macintosh",
        "deviceversion": "Unknown",
        "facebookcarrier": "Unknown",
        "facebookdeviceclass": "Unknown",
        "facebookdevicename": "Unknown",
        "facebookdeviceversion": "Unknown",
        "facebookfbop": "Unknown",
        "facebookfbss": "Unknown",
        "facebookoperatingsystemname": "Unknown",
        "facebookoperatingsystemversion": "Unknown",
        "gsainstallationid": "Unknown",
        "hackerattackvector": "Unknown",
        "hackertoolkit": "Unknown",
        "iecompatibilitynameversion": "Unknown",
        "iecompatibilitynameversionmajor": "Unknown",
        "iecompatibilityversion": "Unknown",
        "iecompatibilityversionmajor": "Unknown",
        "koboaffiliate": "Unknown",
        "koboplatformid": "Unknown",
        "layoutenginebuild": "Unknown",
        "layoutengineclass": "Browser",
        "layoutenginename": "Blink",
        "layoutenginenameversion": "Blink 32.0",
        "layoutenginenameversionmajor": "Blink 32",
        "layoutengineversion": "32.0",
        "layoutengineversionmajor": "32",
        "operatingsystemclass": "Desktop",
        "operatingsystemname": "Mac OS X",
        "operatingsystemnameversion": "Mac OS X 10.9.1",
        "operatingsystemversion": "10.9.1",
        "operatingsystemversionbuild": "Unknown",
```


(continues on next page)

(continued from previous page)


```
        "webviewappname": "Unknown",
        "webviewappnameversionmajor": "Unknown",
        "webviewappversion": "Unknown",
        "webviewappversionmajor": "Unknown",
        "bytes_out": 171717,
        "http_method": "GET",
        "http_query": "/presentations/logstash-monitorama-2013/images/kibana-
↪dashboard3.png",
        "http_referer": "http://semicomplete.com/presentations/logstash-monitorama-
↪2013/",
        "http_status": "200",
        "http_user_agent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1)␣
↪AppleWebKit/537.36 (KHTML, like Gecko) Chrome/32.0.1700.77 Safari/537.36",
        "http_version": "HTTP/1.1",
        "identd": "-",
        "record_id": "4ca6a8b5-1a60-421e-9ae9-6c30330e497e",
        "record_raw_value": "83.149.9.216 - - [17/May/2015:10:05:43 +0000] \"GET /
↪presentations/logstash-monitorama-2013/images/kibana-dashboard3.png HTTP/1.1\" 200␣
↪171717 \"http://semicomplete.com/presentations/logstash-monitorama-2013/\" \
↪"Mozilla/5.0 (Macintosh; Intel Mac OS X 10_9_1) AppleWebKit/537.36 (KHTML, like␣
↪Gecko) Chrome/32.0.1700.77 Safari/537.36\"",
        "record_time": 1431857143000,
        "record_type": "apache_log",
        "src_ip": "83.149.9.216",
        "user": "-"
    }
}
```


You can now browse your data in Kibana and build great dashboards

1.8.11 Time series sampling & Outliers detection

In the following tutorial we'll handle time series data from a sensor. We'll see how sample the datapoints in a visually non destructive way and

We assume that you are already familiar with logisland platform and that you have successfully done the previous tutorials.

Note: You can download the latest release of logisland and the YAML configuration file for this tutorial which can be also found under *$LOGISLAND_HOME/conf* directory.

1. Setup the time series collection Stream

The first Stream use a KafkaRecordStreamParallelProcessing and chain of a SplitText

The first Processor simply parse the csv lines while the second index them into the search engine. Please note the output schema.

```
# parsing time series
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that links
```

(continues on next page)

(continued from previous page)

```
  configuration:
    kafka.input.topics: logisland_ts_raw
    kafka.output.topics: logisland_ts_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
    avro.output.schema: >
      {  "version":1,
         "type": "record",
         "name": "com.hurence.logisland.record.cpu_usage",
         "fields": [
           { "name": "record_errors",   "type": [ {"type": "array", "items": "string"}
↪,"null"] },
           { "name": "record_raw_key", "type": ["string","null"] },
           { "name": "record_raw_value", "type": ["string","null"] },
           { "name": "record_id",   "type": ["string"] },
           { "name": "record_time", "type": ["long"] },
           { "name": "record_type", "type": ["string"] },
           { "name": "record_value",      "type": ["string","null"] } ]}
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
    - processor: apache_parser
      component: com.hurence.logisland.processor.SplitText
      type: parser
      documentation: a parser that produce events from an apache log REGEX
      configuration:
        record.type: apache_log
        value.regex: (\S+),(\S+)
        value.fields: record_time,record_value
```

2. Setup the Outliers detection Stream

The first Stream use a KafkaRecordStreamParallelProcessing and a DetectOutliers Processor

Note: It's important to see that we perform outliers detection in parallel. So if we would perform this detection for a particular grouping of record we would have used a KafkaRecordStreamSQLAggregator with a `GROUP BY` clause instead.

```
# detect outliers
- stream: detect_outliers
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that match query in parrallel
  configuration:
    kafka.input.topics: logisland_sensor_events
    kafka.output.topics: logisland_sensor_outliers_events
    kafka.error.topics: logisland_errors
    kafka.input.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
```

(continues on next page)

(continued from previous page)

```
    kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 2
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
    - processor: match_query
      component: com.hurence.logisland.processor.DetectOutliers
      type: processor
      documentation: a processor that detection something exotic in a continuous time␣
↪series values
      configuration:
        rotation.policy.type: by_amount
        rotation.policy.amount: 100
        rotation.policy.unit: points
        chunking.policy.type: by_amount
        chunking.policy.amount: 10
        chunking.policy.unit: points
        global.statistics.min: -100000
        min.amount.to.predict: 100
        zscore.cutoffs.normal: 3.5
        zscore.cutoffs.moderate: 5
        record.value.field: record_value
        record.time.field: record_time
        output.record.type: sensor_outlier
```

3. Setup the time series Sampling Stream

The first Stream use a KafkaRecordStreamParallelProcessing and a RecordSampler Processor

```
# sample time series
- stream: detect_outliers
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that match query in parrallel
  configuration:
    kafka.input.topics: logisland_sensor_events
    kafka.output.topics: logisland_sensor_sampled_events
    kafka.error.topics: logisland_errors
    kafka.input.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 2
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
    - processor: sampler
      component: com.hurence.logisland.processor.SampleRecords
      type: processor
      documentation: a processor that reduce the number of time series values
      configuration:
```

(continues on next page)

(continued from previous page)

```
        record.value.field: record_value
        record.time.field: record_time
        sampling.algorithm: average
        sampling.parameter: 10
```

4. Setup the indexing Stream

The last Stream use a KafkaRecordStreamParallelProcessing and chain of a SplitText and a BulkAddElasticsearch for indexing the whole records

```
# index records
- stream: indexing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that links
  configuration:
    kafka.input.topics: logisland_sensor_events,logisland_sensor_outliers_events,
↪logisland_sensor_sampled_events
    kafka.output.topics: none
    kafka.error.topics: logisland_errors
    kafka.input.topics.serializer: none
    kafka.input.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.output.topics.serializer: none
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
    - processor: es_publisher
      component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch
      type: processor
      documentation: a processor that trace the processed events
      configuration:
        elasticsearch.client.service: elasticsearch_service
        default.index: logisland
        default.type: event
        timebased.index: yesterday
        es.index.field: search_index
        es.type.field: record_type
```

4. Start logisland application

Connect a shell to your logisland container to launch the following stream processing job previously defined.

```
docker exec -ti logisland bash

#launch logisland streams
cd $LOGISLAND_HOME
bin/logisland.sh --conf conf/outlier-detection.yml

# send logs to kafka
cat cpu_utilization_asg_misconfiguration.csv | kafkacat -b sandbox:9092 -P -t␣
↪logisland_sensor_raw
```

(continues on next page)

(continued from previous page)

```
```

5. Check your alerts with Kibana

1.8.12 Bro/Logisland integration - Indexing Bro events

Bro and Logisland

Bro is a Network IDS (Intrusion Detection System) that can be deployed to monitor your infrastructure. Bro listens to the packets of your network and generates high level events from them. It can for instance generate an event each time there is a connection, a file transfer, a DNS query... anything that can be deduced from packet analysis.

Through its out-of-the-box ParseBroEvent processor, Logisland integrates with Bro and is able to receive and handle Bro events and notices coming from Bro. By analyzing those events with Logisland, you may do some correlations and for instance generate some higher level alarms or do whatever you want, in a scalable manner, like monitoring a huge infrastructure with hundreds of machines.

Bro comes with a scripting language that allows to also generate some higher level events from other events correlations. Bro calls such events 'notices'. For instance a notice can be generated when a user or bot tries to guess a password with brute forcing. Logisland is also able to receive and handle those notices.

For the purpose of this tutorial, we will show you how to receive Bro events and notices in Logisland and how to index them in ElasticSearch for network audit purpose. But you can imagine to plug any Logisland processors after the ParseBroEvent processor to build your own monitoring system or any other application based on Bro events and notices handling.

Tutorial environment

This tutorial will give you a better understanding of how Bro and Logisland integrate together.

We will start two Docker containers:

- 1 container hosting all the LogIsland services
- 1 container hosting Bro pre-loaded with Bro-Kafka plugin

We will launch two streaming processes and configure Bro to send events and notices to the Logisland system so that they are indexed in ElasticSearch.

It is important to understand that in a production environment Bro would be installed on machines where he is relevant for your infrastructure and will be configured to remotely point to the Logisland service (Kafka). But for easiness of this tutorial, we provide you with an easy mean of generating Bro events through our Bro Docker image.

This tutorial will guide you through the process of configuring Logisland for treating Bro events, and configuring Bro of the second container to send the events and notices to the Logisland service in the first container.

Note: You can download the latest release of Logisland and the YAML configuration file for this tutorial which can be also found under *$LOGISLAND_HOME/conf* directory in the Logsiland container.

1. Start the Docker container with LogIsland

LogIsland is packaged as a Docker image that you can build yourself or pull from Docker Hub. The docker image is built from a CentOs image with the following components already installed (among some others not useful for this tutorial):

- Kafka
- Spark
- Elasticsearch
- LogIsland

Pull the image from Docker Repository (it may take some time)

```
docker pull hurence/logisland
```

You should be aware that this Docker container is quite eager in RAM and will need at least 8G of memory to run smoothly. Now run the container

```
# run container
docker run \
    -it \
    -p 80:80 \
    -p 8080:8080 \
    -p 3000:3000 \
    -p 9200-9300:9200-9300 \
    -p 5601:5601 \
    -p 2181:2181 \
    -p 9092:9092 \
    -p 9000:9000 \
    -p 4050-4060:4050-4060 \
    --name logisland \
    -h sandbox \
    hurence/logisland bash

# get container ip
docker inspect logisland | grep IPAddress

# or if your are on mac os
docker-machine ip default
```

You should add an entry for **sandbox** (with the container ip) in your `/etc/hosts` as it will be easier to access to all web services in Logisland running container. Or you can use 'localhost' instead of 'sandbox' where applicable.

Note: If you have your own Spark and Kafka cluster, you can download the latest release and unzip on an edge node.

2.Install required components

For this tutorial please make sure to already have installed elasticsearch and excel modules.

If not you can just do it through the components.sh command line:

```
bin/components.sh -i com.hurence.logisland:logisland-processor-elasticsearch:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-service-elasticsearch_2_4_0-
↪client:1.1.1
```

3. Transform Bro events into Logisland records

For this tutorial we will receive Bro events and notices and send them to Elasticsearch. The configuration file for this tutorial is already present in the container at `$LOGISLAND_HOME/conf/index-bro-events.yml` and its content can be viewed here . Within the following steps, we will go through this configuration file and detail the sections and what they do.

Connect a shell to your Logisland container to launch a Logisland instance with the following streaming jobs:

```
docker exec -ti logisland bash
cd $LOGISLAND_HOME
bin/logisland.sh --conf conf/index-bro-events.yml
```

Note: Logisland is now started. If you want to go straight forward and do not care for the moment about the configuration file details, you can now skip the following sections and directly go to the *4. Start the Docker container with Bro* section.

Setup Spark/Kafka streaming engine

An Engine is needed to handle the stream processing. The `conf/index-bro-events.yml` configuration file defines a stream processing job setup. The first section configures the Spark engine (we will use a KafkaStreamProcessingEngine) as well as an Elasticsearch service that will be used later in the BulkAddElasticsearch processor.

```
engine:
  component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine
  type: engine
  documentation: Index Bro events with LogIsland
  configuration:
    spark.app.name: IndexBroEventsDemo
    spark.master: local[4]
    spark.driver.memory: 1G
    spark.driver.cores: 1
    spark.executor.memory: 2G
    spark.executor.instances: 4
    spark.executor.cores: 2
    spark.yarn.queue: default
    spark.yarn.maxAppAttempts: 4
    spark.yarn.am.attemptFailuresValidityInterval: 1h
    spark.yarn.max.executor.failures: 20
    spark.yarn.executor.failuresValidityInterval: 1h
    spark.task.maxFailures: 8
    spark.serializer: org.apache.spark.serializer.KryoSerializer
    spark.streaming.batchDuration: 4000
    spark.streaming.backpressure.enabled: false
    spark.streaming.unpersist: false
    spark.streaming.blockInterval: 500
    spark.streaming.kafka.maxRatePerPartition: 3000
    spark.streaming.timeout: -1
    spark.streaming.unpersist: false
    spark.streaming.kafka.maxRetries: 3
    spark.streaming.ui.retainedBatches: 200
    spark.streaming.receiver.writeAheadLog.enable: false
    spark.ui.port: 4050
```

(continues on next page)

(continued from previous page)

```
controllerServiceConfigurations:

  - controllerService: elasticsearch_service
    component: com.hurence.logisland.service.elasticsearch.Elasticsearch_2_4_0_
↪ClientService
    type: service
    documentation: elasticsearch 2.4.0 service implementation
    configuration:
      hosts: sandbox:9300
      cluster.name: elasticsearch
      batch.size: 20000

streamConfigurations:
```

Stream 1: Parse incoming Bro events

Inside this engine you will run a Kafka stream of processing, so we setup input/output topics and Kafka/Zookeeper hosts. Here the stream will read all the Bro events and notices sent in the `bro` topic and push the processing output into the `logisland_events` topic.

```
# Parsing
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: A processor chain that transforms Bro events into Logisland records
  configuration:
    kafka.input.topics: bro
    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
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 2
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
```

Within this stream there is a single processor in the processor chain: the `Bro` processor. It takes an incoming Bro event/notice JSON document and computes a Logisland `Record` as a sequence of fields that were contained in the JSON document.

```
# Transform Bro events into Logisland records
- processor: Bro adaptor
  component: com.hurence.logisland.processor.bro.ParseBroEvent
  type: parser
  documentation: A processor that transforms Bro events into LogIsland events
```

This stream will process Bro events as soon as they will be queued into the `bro` Kafka topic. Each log will be parsed as an event which will be pushed back to Kafka in the `logisland_events` topic.

Stream 2: Index the processed records into Elasticsearch

The second Kafka stream will handle `Records` pushed into the `logisland_events` topic to index them into ElasticSearch. So there is no need to define an output topic. The input topic is enough:

```yaml
# Indexing
- stream: indexing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: processor
  documentation: A processor chain that pushes bro events to ES
  configuration:
    kafka.input.topics: logisland_events
    kafka.output.topics: none
    kafka.error.topics: logisland_errors
    kafka.input.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.output.topics.serializer: none
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 2
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
```

The only processor in the processor chain of this stream is the `BulkAddElasticsearch` processor.

```yaml
# Bulk add into ElasticSearch
- processor: ES Publisher
  component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch
  type: processor
  documentation: A processor that pushes Bro events into ES
  configuration:
    elasticsearch.client.service: elasticsearch_service
    default.index: bro
    default.type: events
    timebased.index: today
    es.index.field: search_index
    es.type.field: record_type
```

The `default.index: bro` configuration parameter tells the processor to index events into an index starting with the `bro` string. The `timebased.index: today` configuration parameter tells the processor to use the current date after the index prefix. Thus the index name is of the form `/bro.2017.02.23`.

Finally, the `es.type.field: record_type` configuration parameter tells the processor to use the record field `record_type` of the incoming record to determine the ElasticSearch type to use within the index.

We will come back to these settings and what they do in the section where we see examples of events to illustrate the workflow.

4. Start the Docker container with Bro

For this tutorial, we provide Bro as a Docker image that you can build yourself or pull from Docker Hub. The docker image is built from an Ubuntu image with the following components already installed:

- Bro

- Bro-Kafka plugin

Note: Due to the fact that Bro requires a Kafka plugin to be able to send events to Kafka and that building the Bro-Kafka plugin requires some substantial steps (need Bro sources), for this tutorial, we are only focusing on configuring Bro, and consider it already compiled and installed with its Bro-Kafka plugin (this is the case in our Bro docker image). But looking at the Dockerfile we made to build the Bro Docker image and which is located here, you will have an idea on how to install Bro and Bro-Kafka plugin binaries on your own systems.

Pull the Bro image from Docker Repository:

Warning: If the Bro image is not yet available in the Docker Hub: please build our Bro Docker image yourself as described in the link above for the moment.

```
docker pull hurence/bro
```

Start a Bro container from the Bro image:

```
# run container
docker run -it --name bro -h bro hurence/bro

# get container ip
docker inspect bro | grep IPAddress

# or if your are on mac os
docker-machine ip default
```

5. Configure Bro to send events to Kafka

In the following steps, if you want a new shell to your running bro container, do as necessary:

```
docker exec -ti bro bash
```

Make the sandbox hostname reachable

Kafka in the Logisland container broadcasts his hostname which we have set up being `sandbox`. For this hostname to be reachable from the Bro container, we must declare the IP address of the Logisland container. In the Bro container, edit the `/etc/hosts` file and add the following line at the end of the file, using the right IP address:

```
172.17.0.2  sandbox
```

Note: Be sure to use the IP address of your Logisland container.

Note: Any potential communication problem of the Bro-Kafka plugin will be displayed in the `/usr/local/bro/spool/bro/stderr.log` log file. Also, you should not need this, but the advertised name used by Kafka is declared in the `/usr/local/kafka/config/server.properties` file (in the Logisland container), in the `advertised.host.name` property. Any modification to this property requires a Kafka server restart.

Edit the Bro config file

We will configure Bro so that it loads the Bro-Kafka plugin at startup. We will also point to Kafka of the Logisland container and define the event types we want to push to Logisland.

Edit the config file of bro:

```
vi $BRO_HOME/share/bro/site/local.bro
```

At the beginning of the file, add the following section (take care to respect indentation):

```
@load Bro/Kafka/logs-to-kafka.bro
    redef Kafka::kafka_conf = table(
        ["metadata.broker.list"] = "sandbox:9092",
        ["client.id"] = "bro"
    );
    redef Kafka::topic_name = "bro";
    redef Kafka::logs_to_send = set(Conn::LOG, DNS::LOG, SSH::LOG, Notice::LOG);
    redef Kafka::tag_json = T;
```

Let's detail a bit what we did:

This line tells Bro to load the Bro-Kafka plugin at startup (the next lines are configuration for the Bro-Kafka plugin):

```
@load Bro/Kafka/logs-to-kafka.bro
```

These lines make the Bro-Kafka plugin point to the Kafka instance in the Logisland container (host, port, client id to use). These are communication settings:

```
redef Kafka::kafka_conf = table(
    ["metadata.broker.list"] = "sandbox:9092",
    ["client.id"] = "bro"
    );
```

This line tells the Kafka topic name to use. It is important that it is the same as the input topic of the ParseBroEvent processor in Logisland:

```
redef Kafka::topic_name = "bro";
```

This line tells the Bro-Kafka plugin what type of events should be intercepted and sent to Kafka. For this tutorial we send Connections, DNS and SSH events. We are also interested in any notice (alert) that Bro can generate. For a complete list of possibilities, see the Bro documentation for events and notices. If you want all possible events and notices available by default to be sent into Kafka, just comment this line:

```
redef Kafka::logs_to_send = set(Conn::LOG, DNS::LOG, SSH::LOG, Notice::LOG);
```

This line tells the Bro-Kafka plugin to add the event type in the Bro JSON document it sends. This is required and expected by the Bro Processor as it uses this field to tag the record with his type. This also tells Logisland which ElasticSearch index type to use for storing the event:

```
redef Kafka::tag_json = T;
```

Start Bro

To start bro, we use the `broctl` command that is already in the path of the container. It starts an interactive session to control bro:

```
broctl
```

Then start the bro service: use the `deploy` command in broctl session:

```
Welcome to BroControl 1.5-9

Type "help" for help.

[BroControl] > deploy
checking configurations ...
installing ...
removing old policies in /usr/local/bro/spool/installed-scripts-do-not-touch/site ...
removing old policies in /usr/local/bro/spool/installed-scripts-do-not-touch/auto ...
creating policy directories ...
installing site policies ...
generating standalone-layout.bro ...
generating local-networks.bro ...
generating broctl-config.bro ...
generating broctl-config.sh ...
stopping ...
bro not running
starting ...
starting bro ...
```

Note: The `deploy` command is a shortcut to the `check`, `install` and `restart` commands. Everytime you modify the `$BRO_HOME/share/bro/site/local.bro` configuration file, you must re-issue a `deploy` command so that changes are taken into account.

6. Generate some Bro events and notices

Now that everything is in place you can generate some network activity in the Bro container to generate some events and see them indexed in ElasticSearch.

Monitor Kafka topic

We will generate some events but first we want to see them in Kafka to be sure Bro has forwarded them to Kafka. Connect to the Logisland container:

```
docker exec -ti logisland bash
```

Then use the `kafkacat` command to listen to messages incoming in the `bro` topic:

```
kafkacat -b localhost:9092 -t bro -o end
```

Let the command run. From now on, any incoming event from Bro and entering Kafka will be also displayed in this shell.

Issue a DNS query

Open a shell to the Bro container:

```
docker exec -ti bro bash
```

Then use the `ping` command to trigger an underlying DNS query:

```
ping www.wikipedia.org
```

You should see in the listening `kafkacat` shell an incoming JSON Bro event of type `dns`.

Here is a pretty print version of this event. It should look like this one:

```
{
  "dns": {
    "AA": false,
    "TTLs": [599],
    "id.resp_p": 53,
    "rejected": false,
    "query": "www.wikipedia.org",
    "answers": ["91.198.174.192"],
    "trans_id": 56307,
    "rcode": 0,
    "id.orig_p": 60606,
    "rcode_name": "NOERROR",
    "TC": false,
    "RA": true,
    "uid": "CJkHd3UABb4W7mx8b",
    "RD": false,
    "id.orig_h": "172.17.0.2",
    "proto": "udp",
    "id.resp_h": "8.8.8.8",
    "Z": 0,
    "ts": 1487785523.12837
  }
}
```

The Bro Processor should have processed this event which should have been handled next by the BulkAddElasticsearch processor and finally the event should have been stored in ElasticSearch in the Logisland container.

To see this stored event, we will query ElasticSearch with the `curl` command. Let's browse the `dns` type in any index starting with `bro`:

```
curl http://sandbox:9200/bro*/dns/_search?pretty
```

Note: Do not forget to change sandbox with the IP address of the Logisland container if needed.

You should be able to localize in the response from ElasticSearch a DNS event which looks like:

```
{
  "_index" : "bro.2017.02.23",
  "_type" : "dns",
  "_id" : "6aecfa3a-6a9e-4911-a869-b4e4599a69c1",
  "_score" : 1.0,
  "_source" : {
    "@timestamp": "2017-02-23T17:45:36Z",
    "AA": false,
    "RA": true,
    "RD": false,
```

(continues on next page)

(continued from previous page)

```
    "TC": false,
    "TTLs": [599],
    "Z": 0,
    "answers": ["91.198.174.192"],
    "id_orig_h": "172.17.0.2",
    "id_orig_p": 60606,
    "id_resp_h": "8.8.8.8",
    "id_resp_p": 53,
    "proto": "udp",
    "query": "www.wikipedia.org",
    "rcode": 0,
    "rcode_name": "NOERROR",
    "record_id": "1947d1de-a65e-42aa-982f-33e9c66bfe26",
    "record_time": 1487785536027,
    "record_type": "dns",
    "rejected": false,
    "trans_id": 56307,
    "ts": 1487785523.12837,
    "uid": "CJkHd3UABb4W7mx8b"
  }
}
```

You should see that this JSON document is stored in a indexed of the form `/bro.XXXX.XX.XX/dns`:

```
"_index" : "bro.2017.02.23",
"_type" : "dns",
```

Here, as the Bro event is of type `dns`, the event has been indexed using the `dns` ES type in the index. This allows to easily search only among events of a particular type.

The ParseBroEvent processor has used the first level field `dns` of the incoming JSON event from Bro to add a `record_type` field to the record he has created. This field has been used by the BulkAddElasticsearch processor to determine the index type to use for storing the record.

The `@timestamp` field is added by the BulkAddElasticsearch processor before pushing the record into ES. Its value is derived from the `record_time` field which has been added with also the `record_id` field by Logisland when the event entered Logisland. The `ts` field is the Bro timestamp which is the time when the event was generated in the Bro system.

Other second level fields of the incoming JSON event from Bro have been set as first level fields in the record created by the Bro Processor. Also any field that had a "." chacracter has been transformed to use a "_" character. For instance the `id.orig_h` field has been renamed into `id_orig_h`.

That is basically all the job the Bro Processor does. It's a small adaptation layer for Bro events. Now if you look in the Bro documentation and know the Bro event format, you can be able to know the format of a matching record going out of the ParseBroEvent processor. You should then be able to write some Logsisland processors to handle any record going out of the Bro Processor.

Issue a Bro Notice

As a Bro notice is the result of analysis of many events, generating a real notice event with Bro is a bit more complicated if you want to generate it with real traffic. Fortunately, Bro has the ability to generate events also from `pcap` files. These are "*packect capture*" files. They hold the recording of a real network traffic. Bro analyzes the packets in those files and generate events as if he was listening to real traffic.

In the Bro container, we have preloaded some `pcap` files in the `$PCAP_HOME` directory. Go into this directory:

```
cd $PCAP_HOME
```

The `ssh.pcap` file in this directory is a capture of a network traffic in which there is some SSH traffic with an attempt to guess a user password. The analysis of such traffic generates a Bro `SSH::Password_Guessing` notice.

Let's launch the following command to make Bro analyze the packets in the `ssh.pcap` file and generate this notice:

```
bro -r ssh.pcap local
```

In your previous `kafkacat` shell, you should see some `ssh` events that represent the SSH traffic. You should also see a `notice` event like this one:


```
{
  "notice": {
    "ts":1320435875.879278,
    "note":"SSH::Password_Guessing",
    "msg":"172.16.238.1 appears to be guessing SSH passwords (seen in 30 connections).
↪",
    "sub":"Sampled servers:  172.16.238.136, 172.16.238.136, 172.16.238.136, 172.16.
↪238.136, 172.16.238.136",
    "src":"172.16.238.1",
    "peer_descr":"bro",
    "actions":["Notice::ACTION_LOG"],
    "suppress_for":3600.0,
    "dropped":false
  }
}
```


Then, like for the DNS event, it should also have been indexed in the `notice` index type in ElastiSearch. Browse documents in this type like this:

```
curl http://sandbox:9200/bro*/notice/_search?pretty
```

Note: Do not forget to change sandbox with the IP address of the Logisland container if needed.

In the response, you should see a notice event like this:


```
{
   "_index" : "bro.2017.02.23",
   "_type" : "notice",
   "_id" : "76ab556b-167d-4594-8ee8-b05594cab8fc",
   "_score" : 1.0,
   "_source" : {
     "@timestamp" : "2017-02-23T10:45:08Z",
     "actions" : [ "Notice::ACTION_LOG" ],
     "dropped" : false,
     "msg" : "172.16.238.1 appears to be guessing SSH passwords (seen in 30␣
↪connections).",
     "note" : "SSH::Password_Guessing",
     "peer_descr" : "bro",
     "record_id" : "76ab556b-167d-4594-8ee8-b05594cab8fc",
     "record_time" : 1487933108041,
     "record_type" : "notice",
     "src" : "172.16.238.1",
     "sub" : "Sampled servers:  172.16.238.136, 172.16.238.136, 172.16.238.136, 172.
↪16.238.136, 172.16.238.136",
```


(continues on next page)

(continued from previous page)

```
    "suppress_for" : 3600.0,
    "ts" : 1.320435875879278E9
  }
}
```

We are done with this first approach of Bro integration with LogIsland.

As we configured Bro to also send SSH and Connection events to Kafka, you can have a look at the matching generated events in ES by browsing the `ssh` and `conn` index types:

```
# Browse SSH events
curl http://sandbox:9200/bro*/ssh/_search?pretty
# Browse Connection events
curl http://sandbox:9200/bro*/conn/_search?pretty
```

If you wish, you can also add some additional event types to be sent to Kafka in the Bro config file and browse the matching indexed events in ES using the same kind of `curl` commands just by changing the type in the query (do not forget to re-deploy Bro after configuration file modifications).

1.8.13 Netflow/Logisland integration - Handling Netflow traffic

Netflow and Logisland

Netflow is a feature introduced on Cisco routers that provides the ability to collect IP network traffic. We can distinguish 2 components:

- Flow exporter: aggregates packets into flows and exports flow records (binary format) towards flow collectors
- Flow collector: responsible for reception, storage and pre-processing of flow data received from a flow exporter

The collected data are therefore available for analysis purpose (intrusion detection, traffic analysis…)

Network Flows: A network flow can be defined in many ways. Cisco standard NetFlow version 5 defines a flow as a unidirectional sequence of packets that all share the following 7 values:

1. Ingress interface (SNMP ifIndex)
2. Source IP address
3. Destination IP address
4. IP protocol
5. Source port for UDP or TCP, 0 for other protocols
6. Destination port for UDP or TCP, type and code for ICMP, or 0 for other protocols
7. IP Type of Service

NetFlow Record

A NetFlow record can contain a wide variety of information about the traffic in a given flow. NetFlow version 5 (one of the most commonly used versions, followed by version 9) contains the following:

- Input interface index used by SNMP (ifIndex in IF-MIB).
- Output interface index or zero if the packet is dropped.
- Timestamps for the flow start and finish time, in milliseconds since the last boot.
- Number of bytes and packets observed in the flow

- Layer 3 headers:
  - Source & destination IP addresses
  - ICMP Type and Code.
  - IP protocol
  - Type of Service (ToS) value
- Source and destination port numbers for TCP, UDP, SCTP
- For TCP flows, the union of all TCP flags observed over the life of the flow.
- Layer 3 Routing information:
  - IP address of the immediate next-hop (not the BGP nexthop) along the route to the destination
  - Source & destination IP masks (prefix lengths in the CIDR notation)

Through its out-of-the-box Netflow processor, Logisland integrates with Netflow (V5) and is able to receive and handle Netflow events coming from a netflow collector. By analyzing those events with Logisland, you may do some analysis for example for intrusion detection or traffic analysis.

In this tutorial, we will show you how to generate some Netflow traffic in LogIsland and how to index them in ElasticSearch and vizualize them in Kinbana. More complexe treatment could bv done by plugging any Logisland processors after the Netflow processor.

Tutorial environment

This tutorial aims to show how to handle Netflow traffic within LogIsland.

For the purpose of this tutorial, we will generate Netflow traffic using nfgen. This tool will simulate a netflow traffic and send binary netflow records on port 2055 of sandbox. A nifi instance running on sandbox will listen on that port for incoming traffic and push the binary events to a kafka broker.

We will launch two streaming processes, one for generating the corresponding Netflow LogIsland records and the second one to index them in ElasticSearch.

Note: It is important to understand that in real environment Netflow traffic will be triggered by network devices (router, switches,. . .), so you will have to get the netflow traffic from the defined collectors, and send the corresponding record (formatted in JSON format as described before) to the Logisland service (Kafka).

Note: You can download the latest release of Logisland and the YAML configuration file for this tutorial which can also be found under *$LOGISLAND_HOME/conf* directory in the LogIsland container.

---

1. Start LogIsland as a Docker container

LogIsland is packaged as a Docker container that you can build yourself or pull from Docker Hub. The docker container is built from a Centos 6.4 image with the following tools enabled (among others)

- Kafka
- Spark
- Elasticsearch
- Kibana

- LogIsland

Pull the image from Docker Repository (it may take some time)

```
docker pull hurence/logisland
```

You should be aware that this Docker container is quite eager in RAM and will need at least 8G of memory to run smoothly. Now run the container

```
# run container
docker run \
    -it \
    -p 80:80 \
    -p 8080:8080 \
    -p 2055:2055 \
    -p 3000:3000 \
    -p 9200-9300:9200-9300 \
    -p 5601:5601 \
    -p 2181:2181 \
    -p 9092:9092 \
    -p 9000:9000 \
    -p 4050-4060:4050-4060 \
    --name logisland \
    -h sandbox \
    hurence/logisland bash

# get container ip
docker inspect logisland

# or if your are on mac os
docker-machine ip default
```

you should add an entry for **sandbox** (with the container ip) in your `/etc/hosts` as it will be easier to access to all web services in logisland running container.

Note: If you have your own Spark and Kafka cluster, you can download the latest release and unzip on an edge node.

2. Configuration steps

First we have to peform some configuration steps on sandbox (to configure and start elasticsearch and nifi). We will create a dynamic template in ElasticSearch (to better handle the field mapping) using the following command:

```
docker exec -ti logisland bash

[root@sandbox /]# curl -XPUT localhost:9200/_template/netflow -d '{
  "template" : "netflow.*",
  "settings": {
    "index.refresh_interval": "5s"
  },
  "mappings" : {
    "netflowevent" : {
      "numeric_detection": true,
      "_all" : {"enabled" : false},
      "properties" : {
        "dOctets": {"index": "analyzed", "type": "long" },
```

(continues on next page)

(continued from previous page)

```
        "dPkts": { "index": "analyzed", "type": "long" },
        "dst_as": { "index": "analyzed", "type": "long" },
        "dst_mask": { "index": "analyzed", "type": "long" },
        "dst_ip4": { "index": "analyzed", "type": "ip" },
        "dst_port": { "index": "analyzed", "type": "long" },
        "first":{"index": "analyzed", "type": "long" },
        "input":{"index": "analyzed", "type": "long" },
        "last":{"index": "analyzed", "type": "long" },
        "nexthop":{"index": "analyzed", "type": "ip" },
        "output":{"index": "analyzed", "type": "long" },
        "nprot":{"index": "analyzed", "type": "long" },
        "record_time":{"index": "analyzed", "type": "date","format": "strict_date_
↪optional_time||epoch_millis" },
        "src_as":{"index": "analyzed", "type": "long" },
        "src_mask":{"index": "analyzed", "type": "long" },
        "src_ip4": { "index": "analyzed", "type": "ip" },
        "src_port":{"index": "analyzed", "type": "long" },
        "flags":{"index": "analyzed", "type": "long" },
        "tos":{"index": "analyzed", "type": "long" },
        "unix_nsecs":{"index": "analyzed", "type": "long" },
        "unix_secs":{"index": "analyzed", "type": "date","format": "strict_date_
↪optional_time||epoch_second" }
      }
    }
  }
}'
```

In order to send netflow V5 event (binary format) to `logisland_raw` Kafka topic, we will use a nifi instance which will simply listen for netflow traffic on a UDP port (we keep here the default netflow port 2055) and push these netflow records to a kafka broker (sandbox:9092 with topic `netflow`).

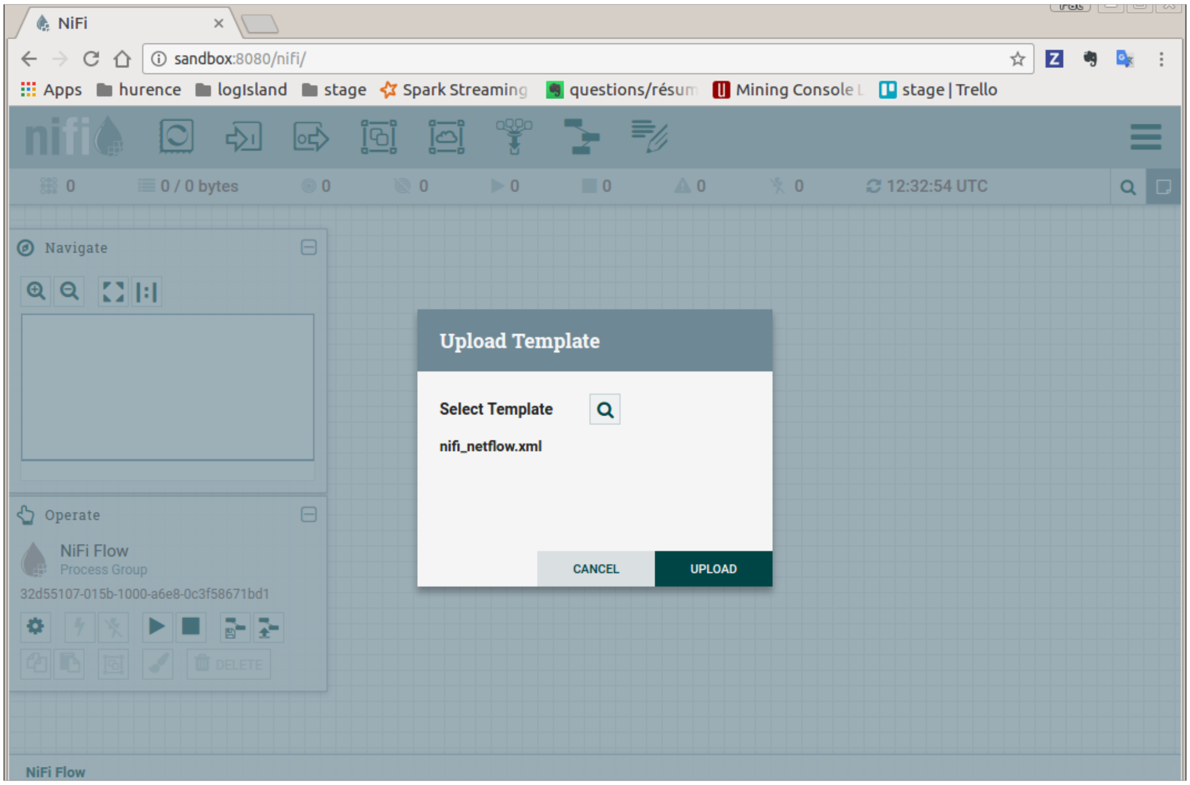
1. Start nifi

   ```
   docker exec -ti logisland bash
   cd /usr/local/nifi-1.1.1
   bin/nifi.sh start
   ```

   browse http://sandbox:8080/nifi/

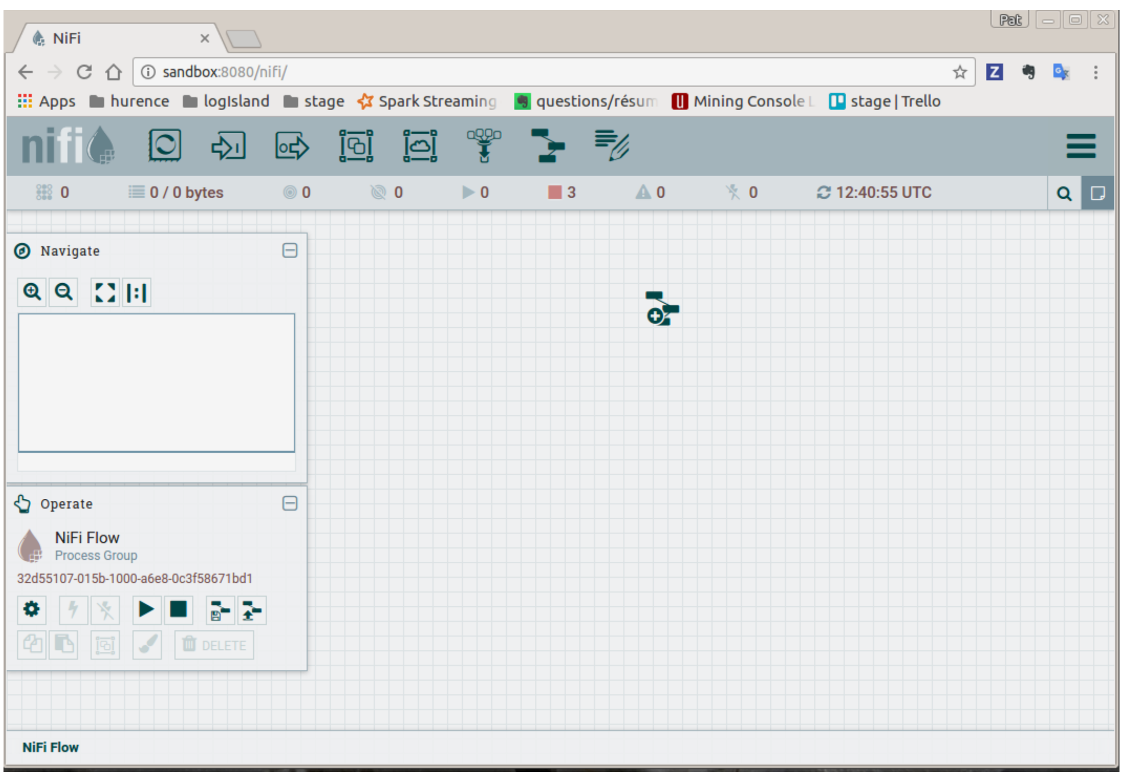
2. Import flow template

   Download this nifi template and import it using "Upload Template" in "Operator" toolbox.

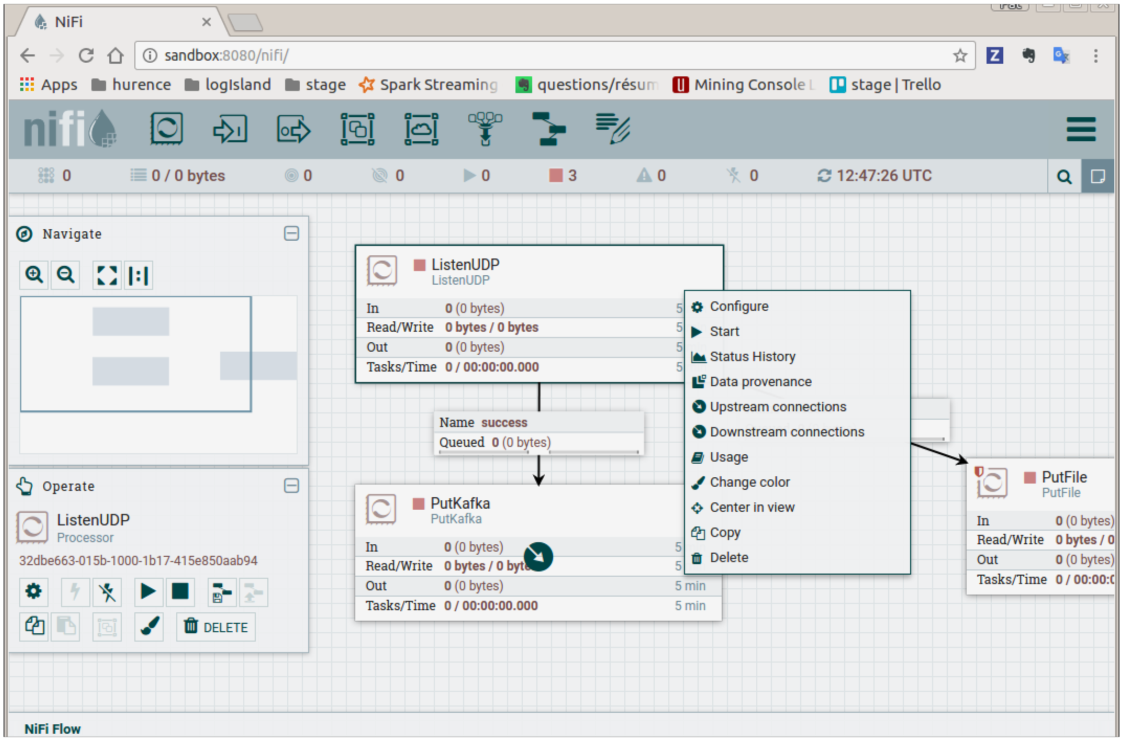


3. Use this template to create the nifi flow

   Drag the nifi toolbar template icon in the nifi work area and choose "nifi_netflow" template, the press "ADD" button



You finally have the following nifi flow



4. start nifi processors

   Select listenUDP processor of nifi flow, right click on it and press "Start". Do the same for putKafka processor.

   Note: the PutFile processor is only for debugging purpose. It dumps netflow records to /tmp/netflow directory (that should be previously created). So you normally don't have to start it for that demo.

   ---

3. Parse Netflow records

For this tutorial we will handle netflow binary events, generate corresponding logisland records and store them to Elasticsearch

Connect a shell to your logisland container to launch the following streaming jobs.

```
docker exec -ti logisland bash
cd $LOGISLAND_HOME
bin/logisland.sh --conf conf/index-netflow-events.yml
```

Setup Spark/Kafka streaming engine

An Engine is needed to handle the stream processing. This `conf/index-netflow-events.yml` configuration file defines a stream processing job setup. The first section configures the Spark engine (we will use a KafkaStreamProcessingEngine) as well as an Elasticsearch service that will be used later in the BulkAddElasticsearch processor.

```
engine:
  component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine
  type: engine
  documentation: Index Netflow events with LogIsland
  configuration:
    spark.app.name: IndexNetFlowEventsDemo
    spark.master: local[4]
    spark.driver.memory: 1G
    spark.driver.cores: 1
    spark.executor.memory: 2G
    spark.executor.instances: 4
    spark.executor.cores: 2
    spark.yarn.queue: default
    spark.yarn.maxAppAttempts: 4
    spark.yarn.am.attemptFailuresValidityInterval: 1h
    spark.yarn.max.executor.failures: 20
    spark.yarn.executor.failuresValidityInterval: 1h
    spark.task.maxFailures: 8
    spark.serializer: org.apache.spark.serializer.KryoSerializer
    spark.streaming.batchDuration: 4000
    spark.streaming.backpressure.enabled: false
    spark.streaming.unpersist: false
    spark.streaming.blockInterval: 500
    spark.streaming.kafka.maxRatePerPartition: 3000
    spark.streaming.timeout: -1
    spark.streaming.unpersist: false
    spark.streaming.kafka.maxRetries: 3
    spark.streaming.ui.retainedBatches: 200
    spark.streaming.receiver.writeAheadLog.enable: false
    spark.ui.port: 4050

  controllerServiceConfigurations:

    - controllerService: elasticsearch_service
      component: com.hurence.logisland.service.elasticsearch.Elasticsearch_2_4_0_
↪ClientService
      type: service
      documentation: elasticsearch 2.4.0 service implementation
      configuration:
        hosts: sandbox:9300
        cluster.name: elasticsearch
        batch.size: 20000

  streamConfigurations:
```

Stream 1 : parse incoming Netflow (Binary format) lines

Inside this engine you will run a Kafka stream of processing, so we setup input/output topics and Kafka/Zookeeper hosts. Here the stream will read all the logs sent in `logisland_raw` topic and push the processing output into `logisland_events` topic.

We can define some serializers to marshall all records from and to a topic.

```
# Parsing
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
```

(continues on next page)

(continued from previous page)

```
  type: stream
  documentation: A processor chain that transforms Netflow events into Logisland␣
↪records
  configuration:
    kafka.input.topics: netflow
    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
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 2
    kafka.topic.default.replicationFactor: 2
  processorConfigurations:
```

Within this stream there is a single processor in the processor chain: the Netflow processor. It takes an incoming Netflow event/notice binary record, parses it and computes a Logisland Record as a sequence of fields that were contained in the binary record.

```
# Transform Netflow events into Logisland records
     - processor: Netflow adaptor
       component: com.hurence.logisland.processor.netflow.ParseNetflowEvent
       type: parser
       documentation: A processor that transforms Netflow events into LogIsland events
       configuration:
         debug: false
         enrich.record: false
```

This stream will process log entries as soon as they will be queued into `logisland_raw` Kafka topics, each log will be parsed as an event which will be pushed back to Kafka in the `logisland_events` topic.

Stream 2: Index the processed records into Elasticsearch

The second Kafka stream will handle `Records` pushed into the `logisland_events` topic to index them into ElasticSearch. So there is no need to define an output topic:

```
# Indexing
- stream: indexing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: processor
  documentation: A processor chain that pushes netflow events to ES
  configuration:
    kafka.input.topics: logisland_events
    kafka.output.topics: none
    kafka.error.topics: logisland_errors
    kafka.input.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.output.topics.serializer: none
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 2
```

(continues on next page)

(continued from previous page)

```
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
```

The only processor in the processor chain of this stream is the `BulkAddElasticsearch` processor.

```
# Bulk add into ElasticSearch
- processor: ES Publisher
  component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch
  type: processor
  documentation: A processor that pushes Netflow events into ES
  configuration:
    elasticsearch.client.service: elasticsearch_service
    default.index: netflow
    default.type: events
    timebased.index: today
    es.index.field: search_index
    es.type.field: record_type
```

The `default.index: netflow` configuration parameter tells the processor to index events into an index starting with the `netflow` string. The `timebased.index: today` configuration parameter tells the processor to use the current date after the index prefix. Thus the index name is of the form `/netflow.2017.03.30`.

Finally, the `es.type.field: record_type` configuration parameter tells the processor to use the record field `record_type` of the incoming record to determine the ElasticSearch type to use within the index.

4. Inject Netflow events into the system

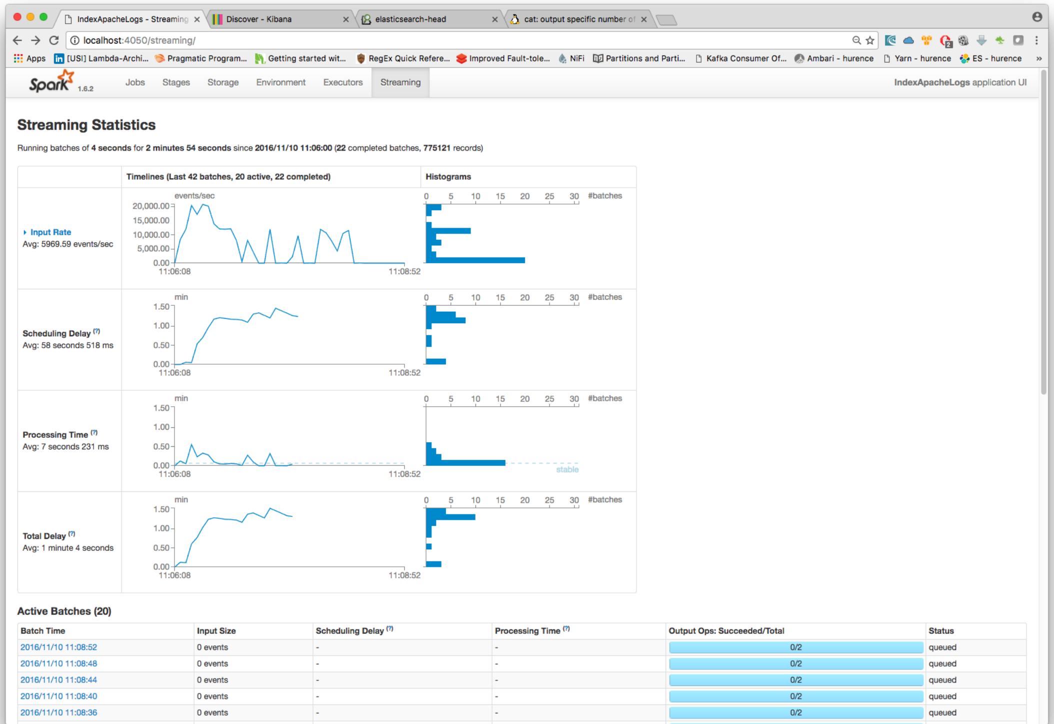
Generate Netflow events to port 2055 of localhost

Now that we have our nifi flow listening on port 2055 from Netflow (V5) traffic and push them to kafka, we have to generate netflow traffic. For the purpose of this tutorial, as already mentioned, we will install and use a netflow traffic generator (but you can use whatever other way to generate Netflow V5 traffic to port 2055)

```
docker exec -ti logisland bash
cd /tmp
wget https://github.com/pazdera/NetFlow-Exporter-Simulator/archive/master.zip
unzip master.zip
cd NetFlow-Exporter-Simulator-master/
make
./nfgen   #this command will generate netflow V5 traffic and send it to local port␣
↪2055.
```

5. Monitor your spark jobs and Kafka topics

Now go to http://sandbox:4050/streaming/ to see how fast Spark can process your data

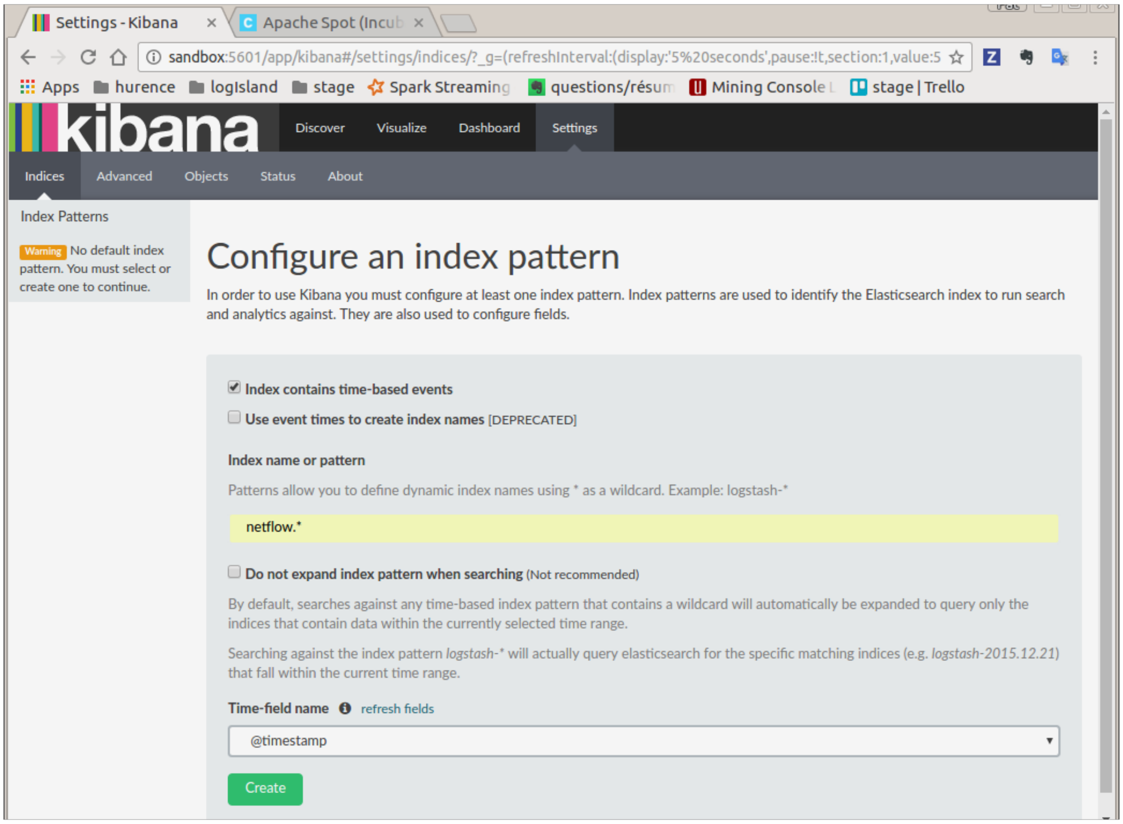


6. Use Kibana to inspect events

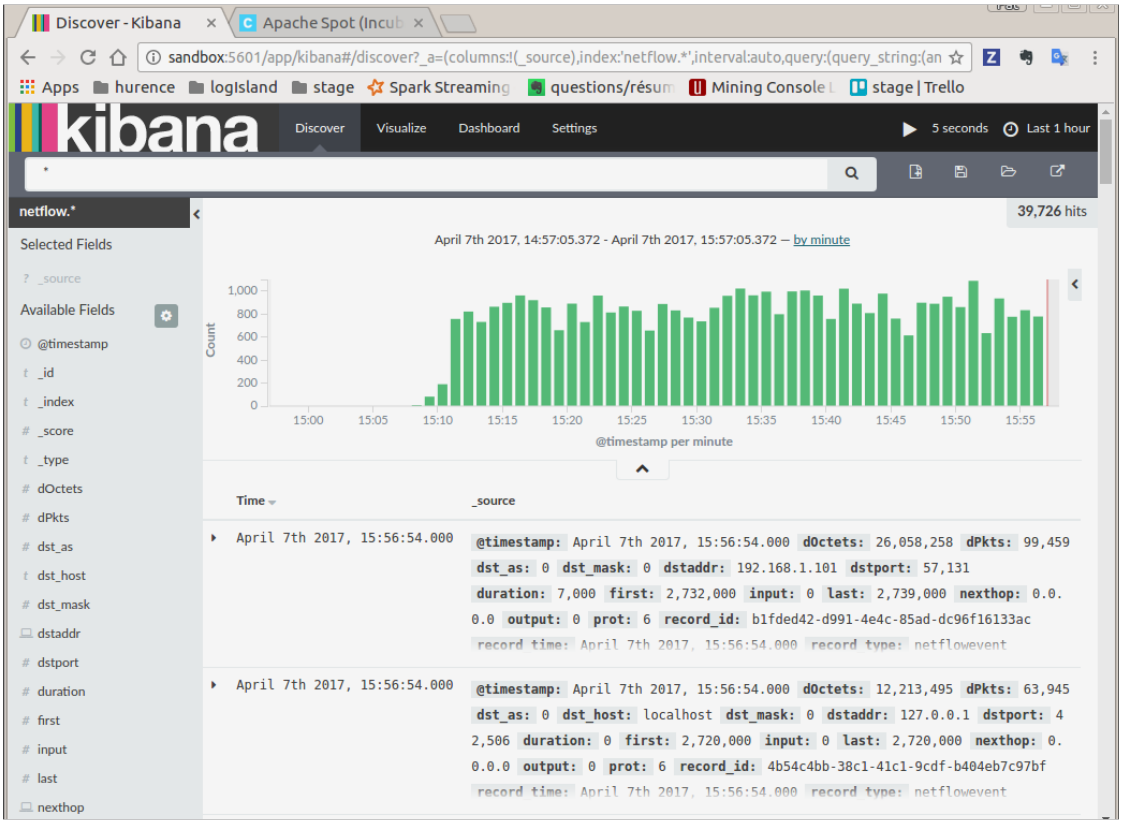
Inspect Netflow events under `Discover` tab

Open your browser and go to http://sandbox:5601/

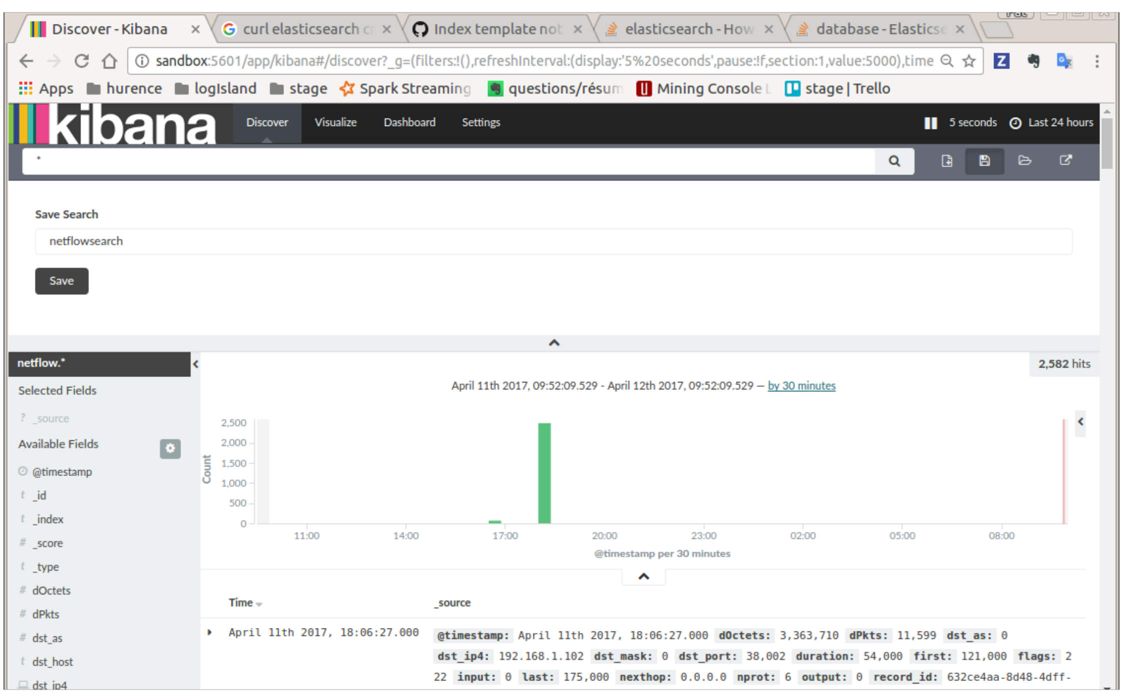
Configure a new index pattern with `netflow.*` as the pattern name and `@timestamp` as the time value field.



Then browse "Discover" tab, you should be able to explore your Netflow events.



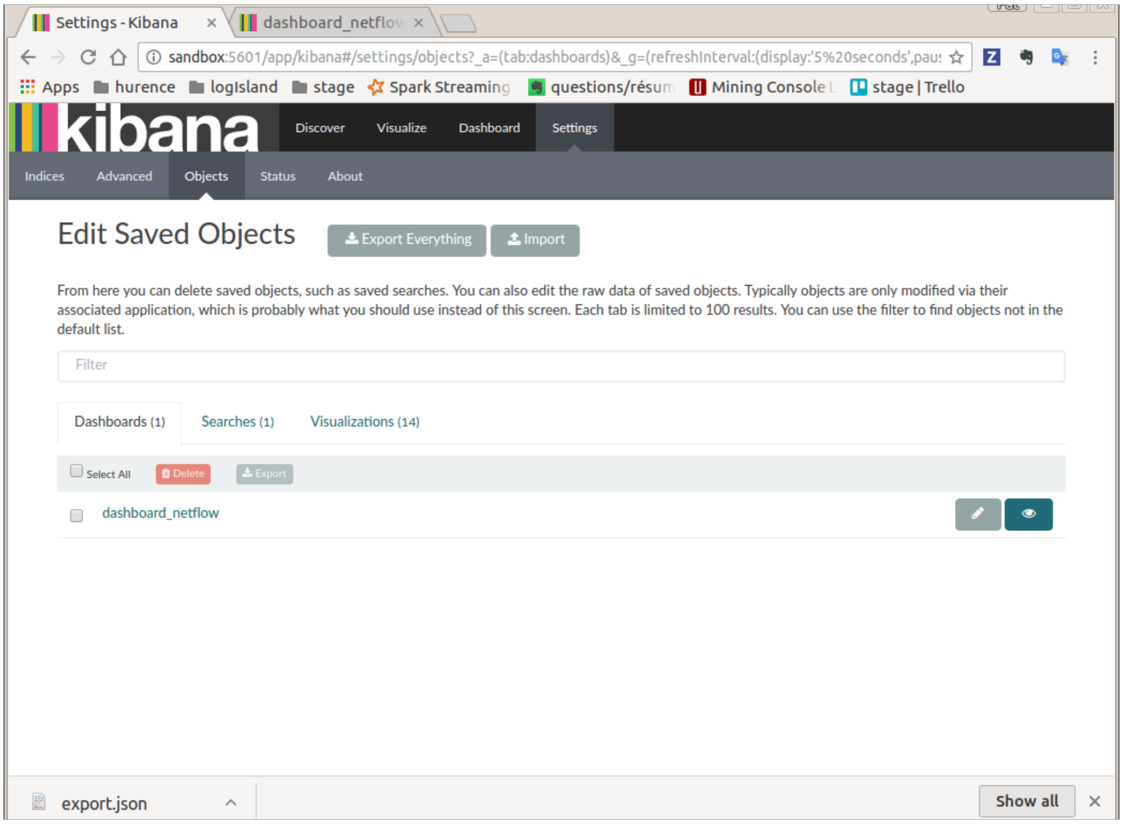
You have now to save your search by clicking the save icon. Save this search as "netflowsearch"



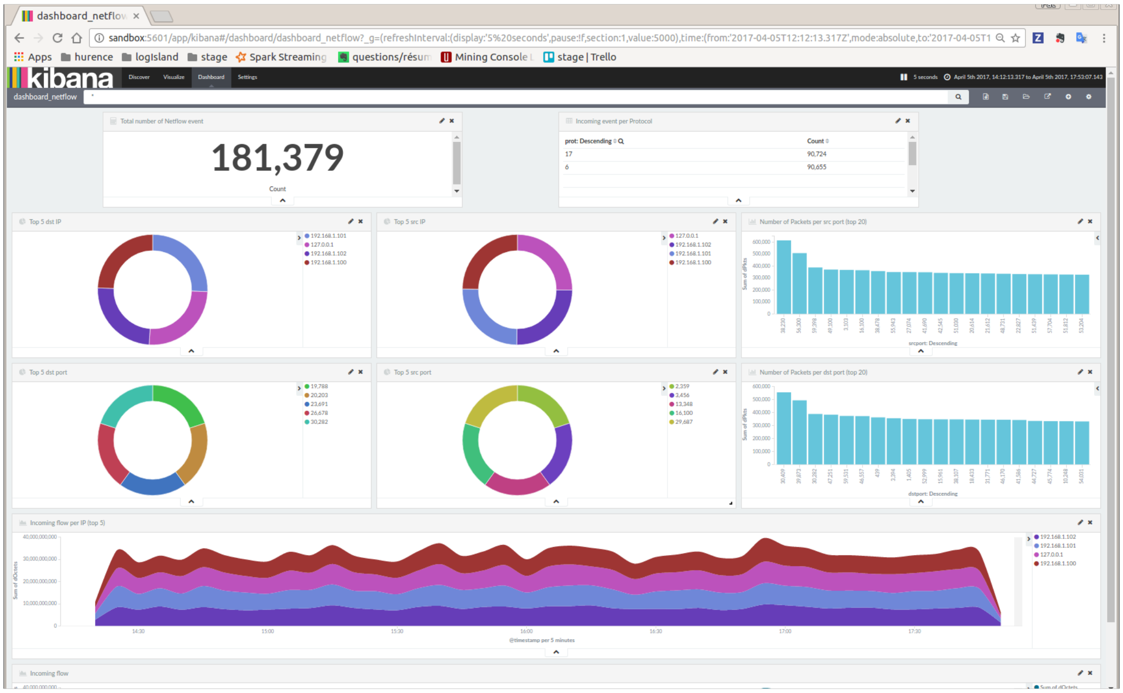
Display network information in kibana dashboard

First, you need to import the predefined Kibana dashboard (download this file first) under `Settings` tab, `Objetcs` subtab.

Select `Import` and load previously saved netflow_dashboard.json dashboard (it also contains required Kibana visualizations)



Then visit `Dashboard` tab, and open `dashboard_netflow` dashboard by clicking on `Load Saved Dashboard`. You should be able to visualize information about the generated traffic (choose the right time window, corresponding to the time of your traffic, in the right upper corner of kibana page)



1.8.14 Capturing Network packets in Logisland

1. Network Packets

A network packet is a formatted unit of data carried by a network from one computer (or device) to another. For example, a web page or an email are carried as a series of packets of a certain size in bytes. Each packet carries the information that will help it get to its destination : the sender's IP address, the intended receiver's IP address, something that tells the network how many packets the message has been broken into, . . .

Packet Headers

1. Protocol headers (IP, TCP, . . .)

This information is stored in different layers called "headers", encapsulating the packet payload. For example, a TCP/IP packet is wrapped in a TCP header, which is in turn encapsulated in an IP header.

The individual packets for a given file or message may travel different routes through the Internet. When they have all arrived, they are reassembled by the TCP layer at the receiving end.

2. PCAP format specific headers

Packets can be either analysed in real-time (stream mode) or stored in files for upcoming analysis (batch mode). In the latter case, the packets are stored in the pcap format, adding some specific headers :

- a global header is added in the beginning of the pcap file
- a packet header is added in front of each packet

In this tutorial we are going to **capture packets in live stream mode**

Why capturing network packets ?

Packet sniffing, or packet analysis, is the process of capturing any data transmitted over the local network and searching for any information that may be useful for :

- Troubleshooting network problems
- Detecting network intrusion attempts
- Detecting network misuse by internal and external users
- Monitoring network bandwidth utilization
- Monitoring network and endpoint security status
- Gathering and report network statistics

Packets information collected by Logisland

LogIsland parses all the fields of IP protocol headers, namely :

1. IP Header fields

- IP version : ip_version
- Internet Header Length : ip_internet_header_length
- Type of Service : ip_type_of_service
- Datagram Total Length : ip_datagram_total_length
- Identification : ip_identification
- Flags : ip_flags
- Fragment offset : ip_fragment_offset
- Time To Live : ip_time_to_live
- Protocol : protocol
- Header Checksum : ip_checksum
- Source IP address : src_ip
- Destination IP address : dst_ip
- Options : ip_options (variable size)
- Padding : ip_padding (variable size)

2. TCP Header fields

- Source port number : src_port
- Destination port number : dest_port
- Sequence Number : tcp_sequence_number
- Acknowledgment Number : tcp_acknowledgment_number
- Data offset : tcp_data_offset
- Flags : tcp_flags
- Window size : tcp_window_size
- Checksum : tcp_checksum

- Urgent Pointer : tcp_urgent_pointer
- Options : tcp_options (variable size)
- Padding : tcp_padding (variable size)

3. UDP Header fields

- Source port number : src_port
- Destination port number : dest_port
- Segment total length : udp_segment_total_length
- Checksum : udp_checksum

2. Tutorial environment

This tutorial aims to show how to capture live Network Packets and process then in LogIsland. Through its out-of-the-box ParseNetworkPacket processor, LogIsland is able to receive and handle network packets captured by a packet sniffer tool. Using LogIsland, you will be able to inspect those packets for network security, optimization or monitoring reasons.

In this tutorial, we will show you how to capture network packets, process those packets in LogIsland, index them in ElasticSearch and then display them in Kibana.

We will launch two streaming processors, one for parsing Network Packets into LogIsland packet records, and one to index those packet records in ElasticSearch.

Packet Capture Tool

For the purpose of this tutorial, we are going to capture network packets (off-the-wire) into a kafka topic using pycapa Apache probe, a tool based on Pcapy, a Python extension module that interfaces with the libpcap packet capture library.

For information, it is also possible to use the fastcapa Apache probe, based on DPDK, intended for high-volume packet capture.

Note: You can download the latest release of LogIsland and the YAML configuration file for this tutorial which can be also found under *$LOGISLAND_HOME/conf* directory in the LogIsland container.

---

3. Start LogIsland as a Docker container

LogIsland is packaged as a Docker container that you can build yourself or pull from Docker Hub. The docker container is built from a Centos 6.4 image with the following tools enabled (among others)

- Kafka
- Spark
- Elasticsearch
- Kibana
- LogIsland

Pull the image from Docker Repository (it may take some time)

```
docker pull hurence/logisland
```

You should be aware that this Docker container is quite eager in RAM and will need at least 8G of memory to run smoothly. Now run the container

```
# run container
docker run \
    -it \
    -p 80:80 \
    -p 8080:8080 \
    -p 3000:3000 \
    -p 9200-9300:9200-9300 \
    -p 5601:5601 \
    -p 2181:2181 \
    -p 9092:9092 \
    -p 9000:9000 \
    -p 4050-4060:4050-4060 \
    --name logisland \
    -h sandbox \
    hurence/logisland bash

# get container ip
docker inspect logisland

# or if your are on mac os
docker-machine ip default
```

you should add an entry for **sandbox** (with the container ip) in your `/etc/hosts` as it will be easier to access to all web services in logisland running container.

Note: If you have your own Spark and Kafka cluster, you can download the latest release and unzip on an edge node.

4. Parse Network Packets

In this tutorial we will capture network packets, process those packets in LogIsland and index them in ElasticSearch.

Connect a shell to your logisland container to launch LogIsland streaming jobs :

```
docker exec -ti logisland bash
cd $LOGISLAND_HOME
bin/logisland.sh --conf conf/index-network-packets.yml
```

Setup Spark/Kafka streaming engine

An Engine is needed to handle the stream processing. This `conf/index-network-packets.yml` configuration file defines a stream processing job setup. The first section configures the Spark engine, we will use a KafkaStreamProcessingEngine :

```
engine:
  component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine
  type: engine
  documentation: Parse network packets with LogIsland
```

(continues on next page)

(continued from previous page)

```
configuration:
  spark.app.name: ParseNetworkPacketDemo
  spark.master: local[4]
  spark.driver.memory: 1G
  spark.driver.cores: 1
  spark.executor.memory: 2G
  spark.executor.instances: 4
  spark.executor.cores: 2
  spark.yarn.queue: default
  spark.yarn.maxAppAttempts: 4
  spark.yarn.am.attemptFailuresValidityInterval: 1h
  spark.yarn.max.executor.failures: 20
  spark.yarn.executor.failuresValidityInterval: 1h
  spark.task.maxFailures: 8
  spark.serializer: org.apache.spark.serializer.KryoSerializer
  spark.streaming.batchDuration: 4000
  spark.streaming.backpressure.enabled: false
  spark.streaming.unpersist: false
  spark.streaming.blockInterval: 500
  spark.streaming.kafka.maxRatePerPartition: 3000
  spark.streaming.timeout: -1
  spark.streaming.unpersist: false
  spark.streaming.kafka.maxRetries: 3
  spark.streaming.ui.retainedBatches: 200
  spark.streaming.receiver.writeAheadLog.enable: false
  spark.ui.port: 4050

controllerServiceConfigurations:

  - controllerService: elasticsearch_service
    component: com.hurence.logisland.service.elasticsearch.Elasticsearch_2_4_0_
↪ClientService
    type: service
    documentation: elasticsearch 2.4.0 service implementation
    configuration:
      hosts: sandbox:9300
      cluster.name: elasticsearch
      batch.size: 4000

streamConfigurations:
```

Stream 1 : parse incoming Network Packets

Inside this engine you will run a Kafka stream of processing, so we setup input/output topics and Kafka/Zookeeper hosts. Here the stream will read all the logs sent in `logisland_input_packets_topic` topic and push the processed packet records into `logisland_parsed_packets_topic` topic.

We can define some serializers to marshall all records from and to a topic.

```
# Parsing
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: A processor chain that parses network packets into Logisland records
  configuration:
```

(continues on next page)

(continued from previous page)

```
    kafka.input.topics: logisland_input_packets_topic
    kafka.output.topics: logisland_parsed_packets_topic
    kafka.error.topics: logisland_error_packets_topic
    kafka.input.topics.serializer: com.hurence.logisland.serializer.
↪BytesArraySerializer
    kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 2
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
```

Within this stream there is a single processor in the processor chain: the ParseNetworkPacket processor. It takes an incoming network packet, parses it and computes a LogIsland record as a sequence of fields corresponding to packet headers fields.

```
# Transform network packets into LogIsland packet records
- processor: ParseNetworkPacket processor
  component: com.hurence.logisland.processor.networkpacket.ParseNetworkPacket
  type: parser
  documentation: A processor that parses network packets into LogIsland records
  configuration:
    debug: true
    flow.mode: stream
```

This stream will process network packets as soon as they will be queued into `logisland_input_packets_topic` Kafka topic, each packet will be parsed as a record which will be pushed back to Kafka in the `logisland_parsed_packets_topic` topic.

Stream 2: Index the processed records into Elasticsearch

The second Kafka stream will handle `Records` pushed into the `logisland_parsed_packets_topic` topic to index them into ElasticSearch. So there is no need to define an output topic:

```
# Indexing
- stream: indexing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: processor
  documentation: a processor that pushes events to ES
  configuration:
    kafka.input.topics: logisland_parsed_packets_topic
    kafka.output.topics: none
    kafka.error.topics: logisland_error_packets_topic
    kafka.input.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.output.topics.serializer: none
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 2
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
```

The only processor in the processor chain of this stream is the `BulkAddElasticsearch` processor.

```yaml
# Bulk add into ElasticSearch
- processor: ES Publisher
  component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch
  type: processor
  documentation: A processor that pushes network packet records into ES
  configuration:
    elasticsearch.client.service: elasticsearch_service
    default.index: packets_index
    default.type: events
    timebased.index: today
    es.index.field: search_index
    es.type.field: record_type
```

The `default.index: packets_index` configuration parameter tells the elasticsearch processor to index records into an index starting with the `packets_index` string. The `timebased.index: today` configuration parameter tells the processor to use the current date after the index prefix. Thus the index name is of the form `/packets_index.2017.03.30`.

Finally, the `es.type.field: record_type` configuration parameter tells the processor to use the record field `record_type` of the incoming record to determine the ElasticSearch type to use within the index.

5. Stream network packets into the system

Let's install and use the Apache pycapa probe to capture and send packets to kafka topics in real time.

Install pycapa probe

All required steps to install pycapa probe are explained in this site, but here are the main installation steps :

1. Install libpcap, pip (python-pip) and python-devel :

```
yum install libpcap
yum install python-pip
yum install python-devel
```

2. Build pycapa probe from Metron repo

```
cd /tmp
git clone https://github.com/apache/incubator-metron.git
cd incubator-metron/metron-sensors/pycapa
pip install -r requirements.txt
python setup.py install
```

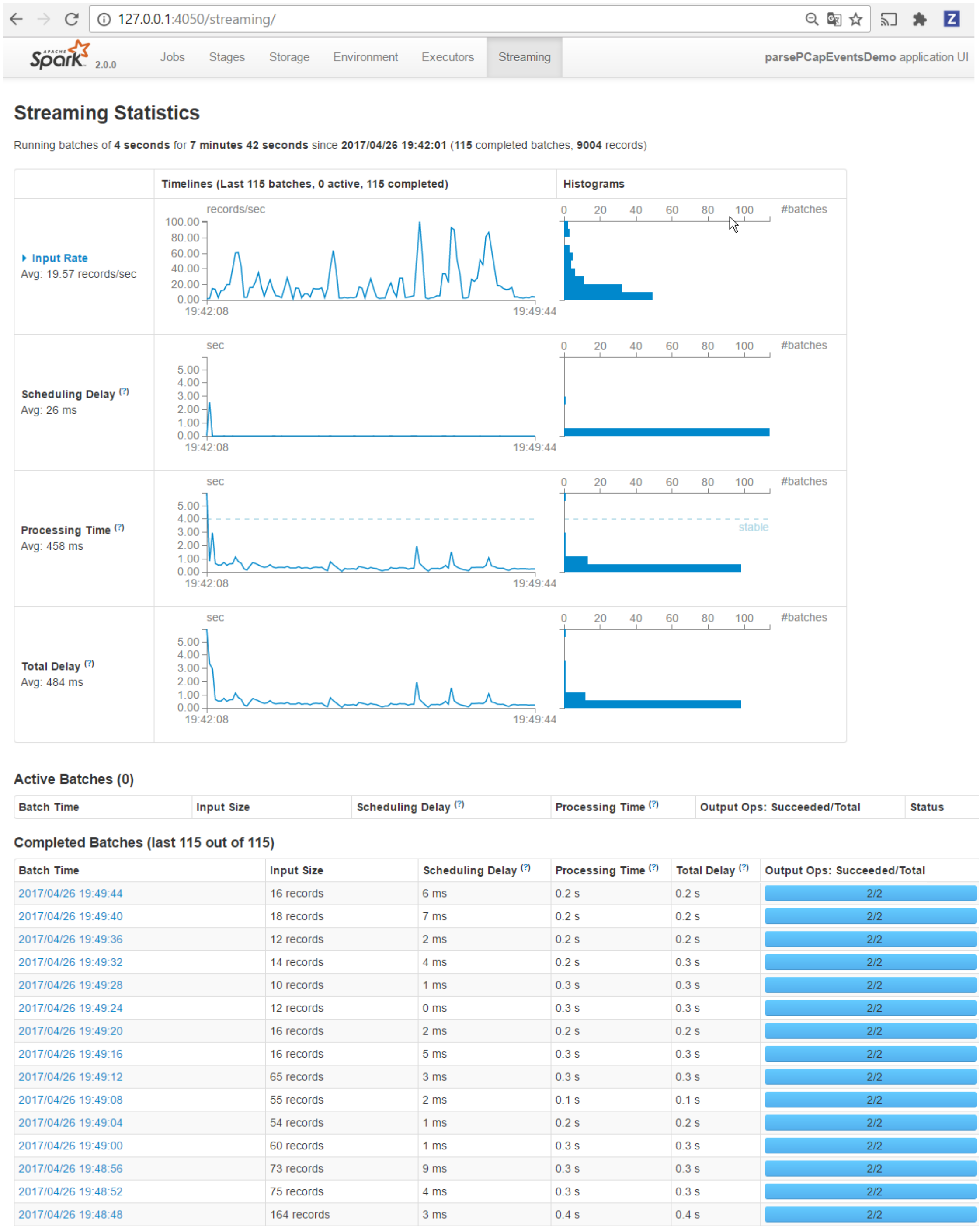
Capture network packets

To start capturing network packets into the topic `logisland_input_packets_topic` using pycapa probe, use the following command :

```
pycapa --producer --kafka sandbox:9092 --topic logisland_input_packets_topic -i eth0
```

6. Monitor your spark jobs and Kafka topics

Now go to http://sandbox:4050/streaming/ to see how fast Spark can process your data

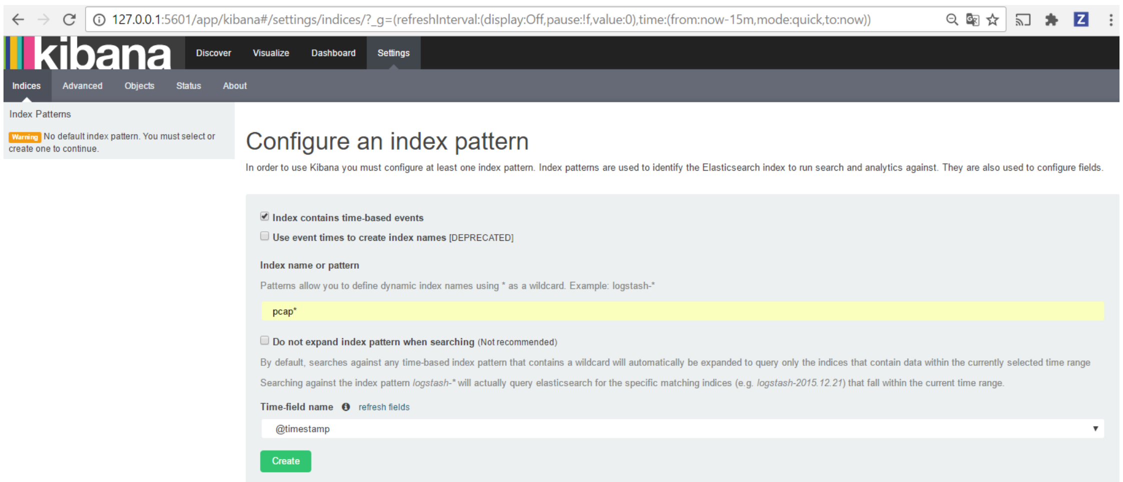


7. Use Kibana to inspect records

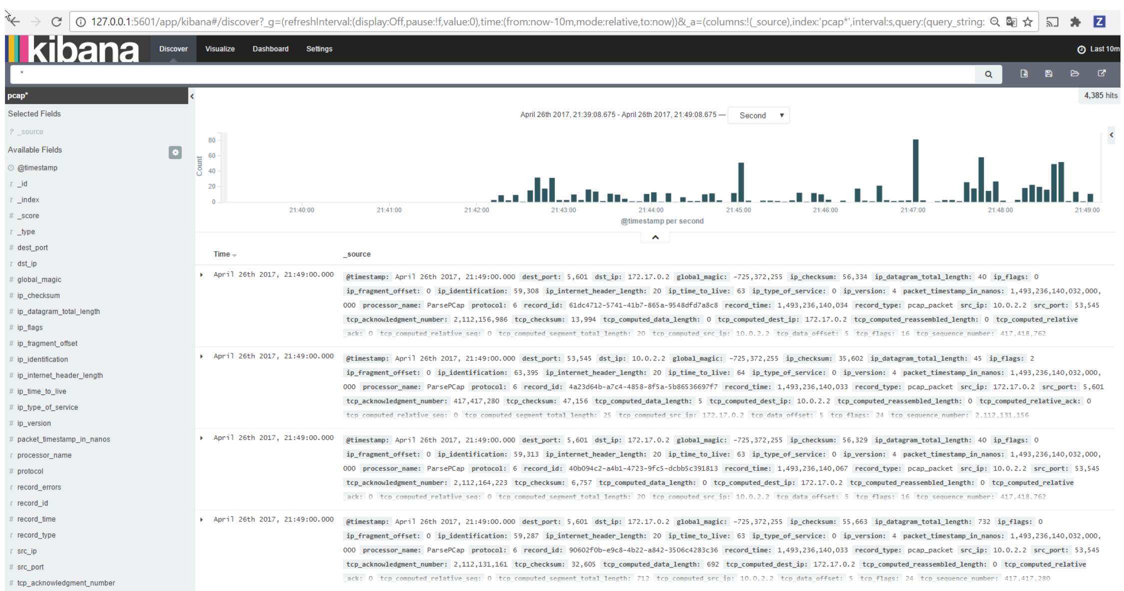
Inspect Network Packets under `Discover` tab

Open your browser and go to http://sandbox:5601/

Configure a new index pattern with `packets.*` as the pattern name and `@timestamp` as the time value field.



Then browse "Discover" tab, you should be able to explore your network packet records :



1.8.15 Generate Unique Ids

We will add a stage to the "index-apache-logs" tutorial. We will ensure every Record has a unique Id before injecting into Es. This way we are sure to not have documentAlreadyException or to have two records that overwrite themselves.

Note: If you are not familiar with logisland yet. You should really read "index-apache-logs" tutorial before this one.

We assume we are at the stage just before injecting apache logs into ES from "index-apache-logs"

Stream 1 : parse incoming apache log lines

Inside this engine you will run a Kafka stream of processing, so we setup input/output topics and Kafka/Zookeeper hosts. Here the stream will read all the logs sent in `logisland_raw` topic and push the processing output into `logisland_events` topic.

Note: We want to specify an Avro output schema to validate our ouput records (and force their types accordingly). It's really for other streams to rely on a schema when processing records from a topic.

We can define some serializers to marshall all records from and to a topic.

```
# parsing
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that links
  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
    avro.output.schema: >
      {  "version":1,
         "type": "record",
         "name": "com.hurence.logisland.record.apache_log",
         "fields": [
           { "name": "record_errors",   "type": [ {"type": "array", "items": "string"}
↪,"null"] },
           { "name": "record_raw_key", "type": ["string","null"] },
           { "name": "record_raw_value", "type": ["string","null"] },
           { "name": "record_id",   "type": ["string"] },
           { "name": "record_time", "type": ["long"] },
           { "name": "record_type", "type": ["string"] },
           { "name": "src_ip",      "type": ["string","null"] },
           { "name": "http_method", "type": ["string","null"] },
           { "name": "bytes_out",   "type": ["long","null"] },
           { "name": "http_query",  "type": ["string","null"] },
           { "name": "http_version","type": ["string","null"] },
           { "name": "http_status", "type": ["string","null"] },
           { "name": "identd",      "type": ["string","null"] },
           { "name": "user",        "type": ["string","null"] }    ]}
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
  processorConfigurations:
```

Within this stream a `SplitText` processor takes a log line as a String and computes a `Record` as a sequence of fields.

```
# parse apache logs
- processor: apache_parser
  component: com.hurence.logisland.processor.SplitText
```

(continues on next page)

(continued from previous page)

```
  type: parser
  documentation: a parser that produce events from an apache log REGEX
  configuration:
    value.regex: (\S+)\s+(\S+)\s+(\S+)\s+\[([\w:\/]+\s[+\-]\d{4})\]\s+
↪"(\S+)\s+(\S+)\s*(\S*)"\s+(\S+)\s+(\S+)
    value.fields: src_ip,identd,user,record_time,http_method,http_query,http_version,
↪http_status,bytes_out
```

Within this stream a ModifyId processor takes Record ouput from SplitText processor and computes a new Id for them using the value of their field "record_raw_value" that should content the original line string of the apache log. It will hash it using "SHA-256" java implementation algorithm, using the charset "UTF-8".

parse apache logs - processor: apache_parser

> component: com.hurence.logisland.processor.ModifyId type: parser documentation: a parser that modify record Ids configuration:
>
> > id.generation.strategy: hashFields hash.charset: UTF-8 fields.to.hash: record_raw_value hash.algorithm: SHA-256

This stream will process log entries as soon as they will be queued into *logisland_raw* Kafka topics, each log will be parsed as an event which will be pushed back to Kafka in the `logisland_events` topic.

Then you can process to your indexation in Elasticsearch as in "index-apache-logs" example.

1.8.16 Index JMS messages

In the following getting started tutorial, we'll explain you how to read messages from a JMS topic or queue and index them into an elasticsearch store.

The JMS data will leverage the JMS connector available as part of logisland connect.

Note: Be sure to know of to launch a logisland Docker environment by reading the prerequisites section

For kafka connect related information please follow as well the connectors section.

1. Installing ActiveMQ

In this tutorial we'll use Apache ActiveMQ.

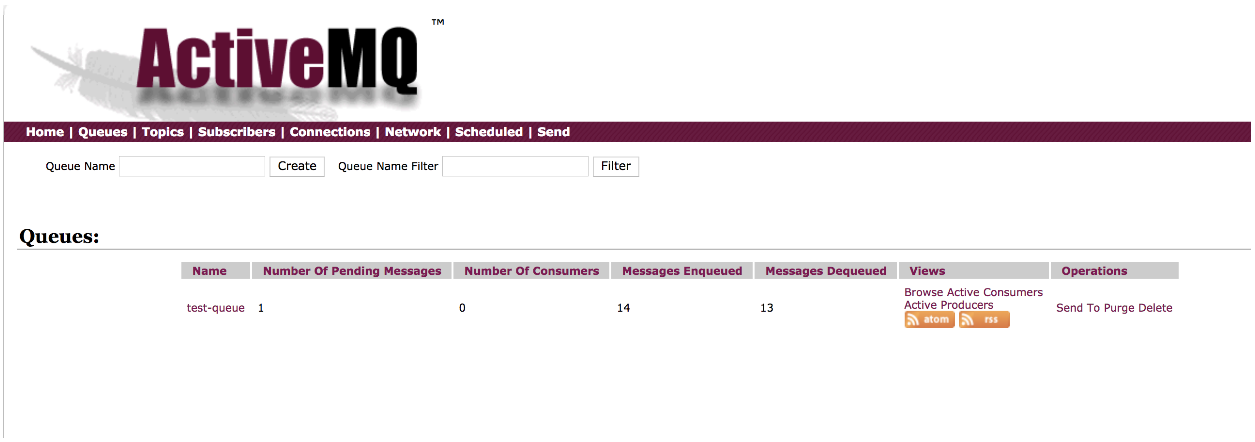
Once you downloaded the broker package just extract it in a folder and turn on your first broker by running:

```
bin/activemq start
```

You can verify if your broker is alive by connecting to the ActiveMQ console (login with admin/admin)

We are also going to create a test queue that we'll use for this tutorial.

To do that, in the just use the ActiveMQ console and in the *queue* section create a queue named *test-queue*. You should have your queue created as shown below.



As well, since JMS is actually an API, we have to provide to logisland the JMS connection factory and the client libraries. For this we can just copy the *activemq-all-5.15.5.jar* into the Logisland lib folder.

For instance, assuming you are running Logisland with the provided docker compose, you can just copy with a command like this:

..code-block:: bash

> docker cp ./activemq-all-5.15.5.jar logisland:/usr/local/logisland/lib

You can verify that activemq jar has been successfully copied inside the docker container by running

..code-block:: bash

> docker exec logisland ls lib/

2. Logisland job setup

For this tutorial please make sure to already have installed elasticsearch and JMS connector modules.

If not you can just do it through the componentes.sh command line:

```
bin/components.sh -i com.hurence.logisland:logisland-processor-elasticsearch:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-service-elasticsearch_5_4_0-client:1.1.1

bin/components.sh -i com.datamountaineer:kafka-connect-jms:1.1.1
```

The interesting part in this tutorial is how to setup the JMS stream.

Let's first focus on the stream configuration and then on its pipeline in order to extract the data in the right way.

The JMS stream

Here we are going to use a special processor (`KafkaConnectStructuredSourceProviderService`) to use the kafka connect source as input for the structured stream defined below.

Logisland ships by default a kafka connect JMS source implemented by the class *com.datamountaineer.streamreactor.connect.jms.source.JMSSourceConnector*.

You can find more information about how to configure a JMS source in the official page of the JMS Connector

Coming back to our example, we would like to read from a queue called *test-queue* hosted in our local ActiveMQ broker. For this we will connect as usual to its Openwire channel and we'll use client acknowledgement to be sure to have an exactly once delivery.

The kafka connect controller service configuration will look like this:

```
- 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.jms.source.
↪JMSSourceConnector
    kc.connector.offset.backing.store: memory
    kc.connector.properties: |
      connect.jms.url=tcp://sandbox:61616
      connect.jms.initial.context.factory=org.apache.activemq.jndi.
↪ActiveMQInitialContextFactory
      connect.jms.connection.factory=ConnectionFactory
      connect.jms.kcql=INSERT INTO topic SELECT * FROM test-queue WITHTYPE QUEUE
      connect.progress.enabled=true
```

The pipeline

Within this stream, a we need to extract the data coming from the JMS.

First of all a `FlatMap` processor takes out the value and key (required when using *StructuredStream* as source of records)

```
processorConfigurations:
 - processor: flatten
   component: com.hurence.logisland.processor.FlatMap
   type: processor
   documentation: "Takes out data from record_value"
   configuration:
     keep.root.record: false
```

Then, since our JMS messages will carry text data, we need to extract this data from the raw message bytes:

```
- processor: add_fields
  component: com.hurence.logisland.processor.AddFields
  type: processor
  documentation: "Extract the message as a text"
  configuration:
  conflict.resolution.policy: overwrite_existing
  message_text: ${new String(bytes_payload)}
```

Now we will as well set the record time as the time when the message has been created (and not received). This thanks to a NormalizeFields processor:

```yaml
- processor: rename_fields
  component: com.hurence.logisland.processor.NormalizeFields
  type: processor
  documentation: "Change the record time according to message_timestamp field"
  configuration:
  conflict.resolution.policy: overwrite_existing
  record_time: message_timestamp
```

Last but not least, a `BulkAddElasticsearch` takes care of indexing a `Record` sending it to elasticsearch.

```yaml
-  processor: es_publisher
   component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch
   type: processor
   documentation: a processor that indexes processed events in elasticsearch
   configuration:
     elasticsearch.client.service: elasticsearch_service
     default.index: logisland
     default.type: event
     timebased.index: yesterday
     es.index.field: search_index
     es.type.field: record_type
```

In details, this processor makes use of a `Elasticsearch_5_4_0_ClientService` controller service to interact with our Elasticsearch 5.X backend running locally (and started as part of the docker compose configuration we mentioned above).

Here below its configuration:

```yaml
- controllerService: elasticsearch_service
  component: com.hurence.logisland.service.elasticsearch.Elasticsearch_5_4_0_
↪ClientService
  type: service
  documentation: elasticsearch service
  configuration:
    hosts: sandbox:9300
    cluster.name: es-logisland
    batch.size: 5000
```

3. Launch the script

Connect a shell to your logisland container to launch the following streaming jobs.

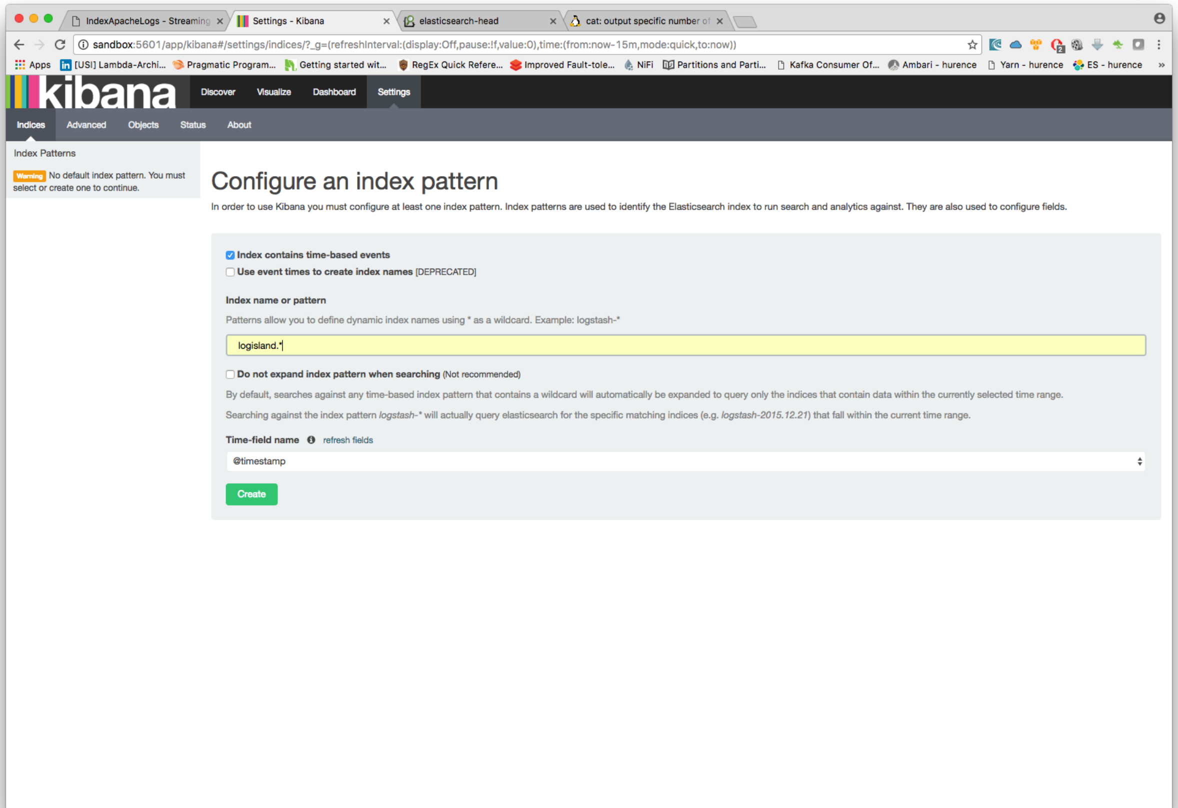
```
bin/logisland.sh --conf conf/index-jms-messages.yml
```

4. Do some insights and visualizations

With ElasticSearch, you can use Kibana.

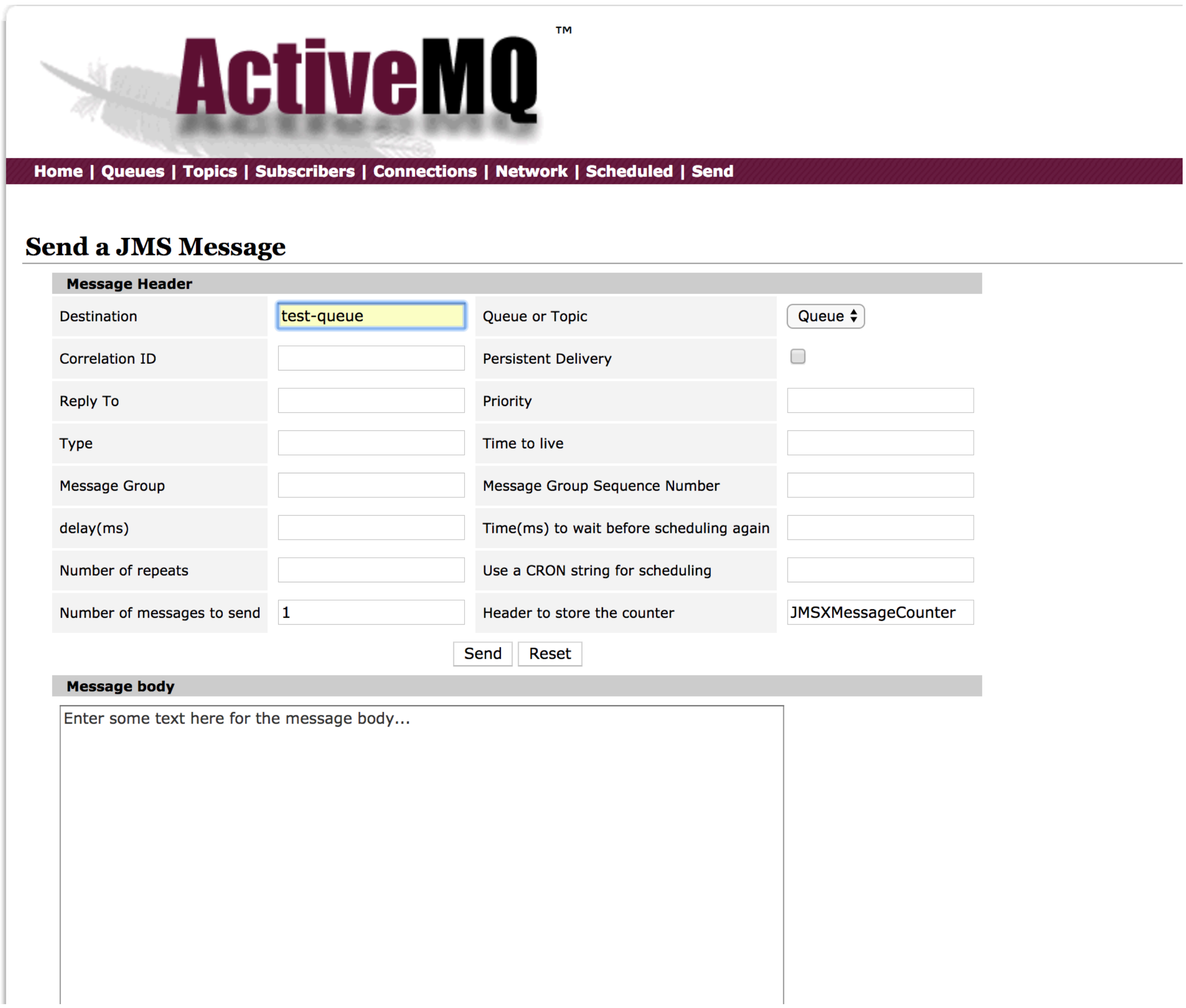
Open up your browser and go to http://sandbox:5601/app/kibana#/ and you should be able to explore the blockchain transactions.

Configure a new index pattern with `logisland.*` as the pattern name and `@timestamp` as the time value field.

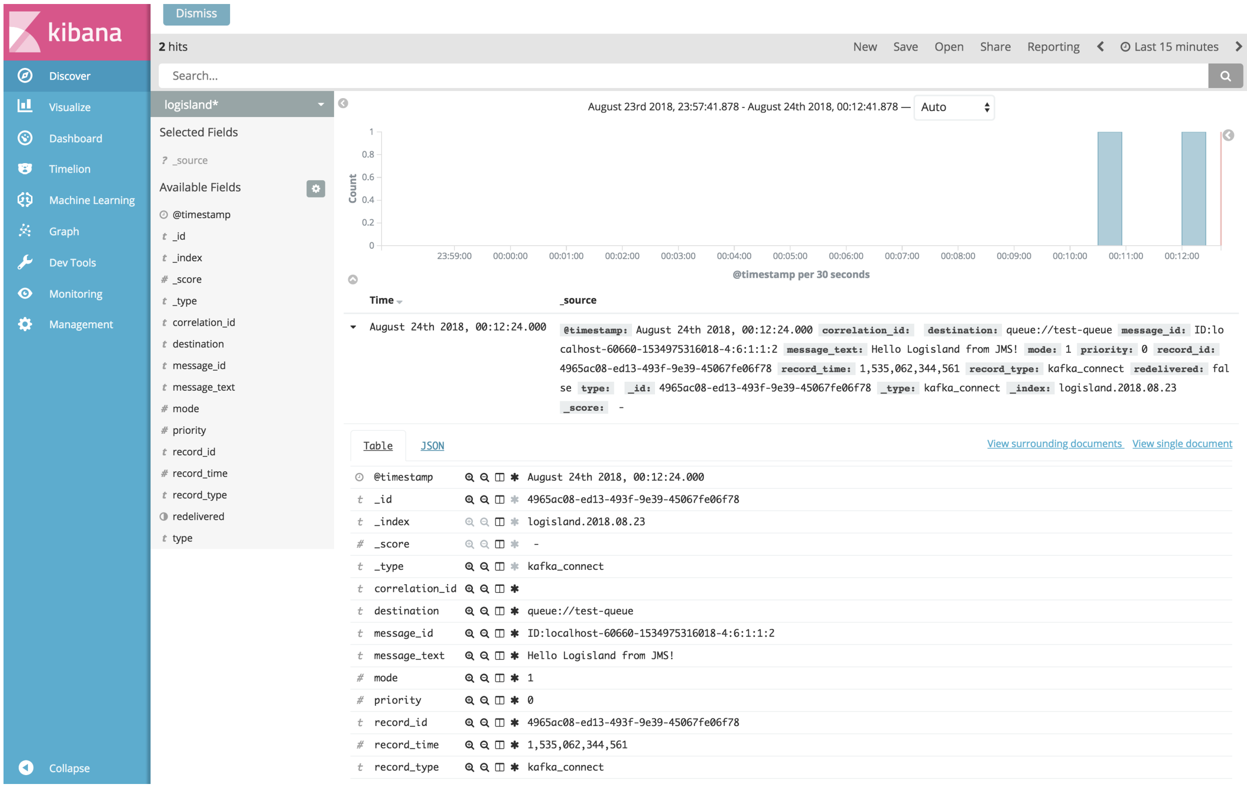


Now just send some message thanks to the ActiveMQ console.

Click on the *Send* link on the top of the console main page and specify the destination to *test-queue* and type the message you like. You should have something like this:

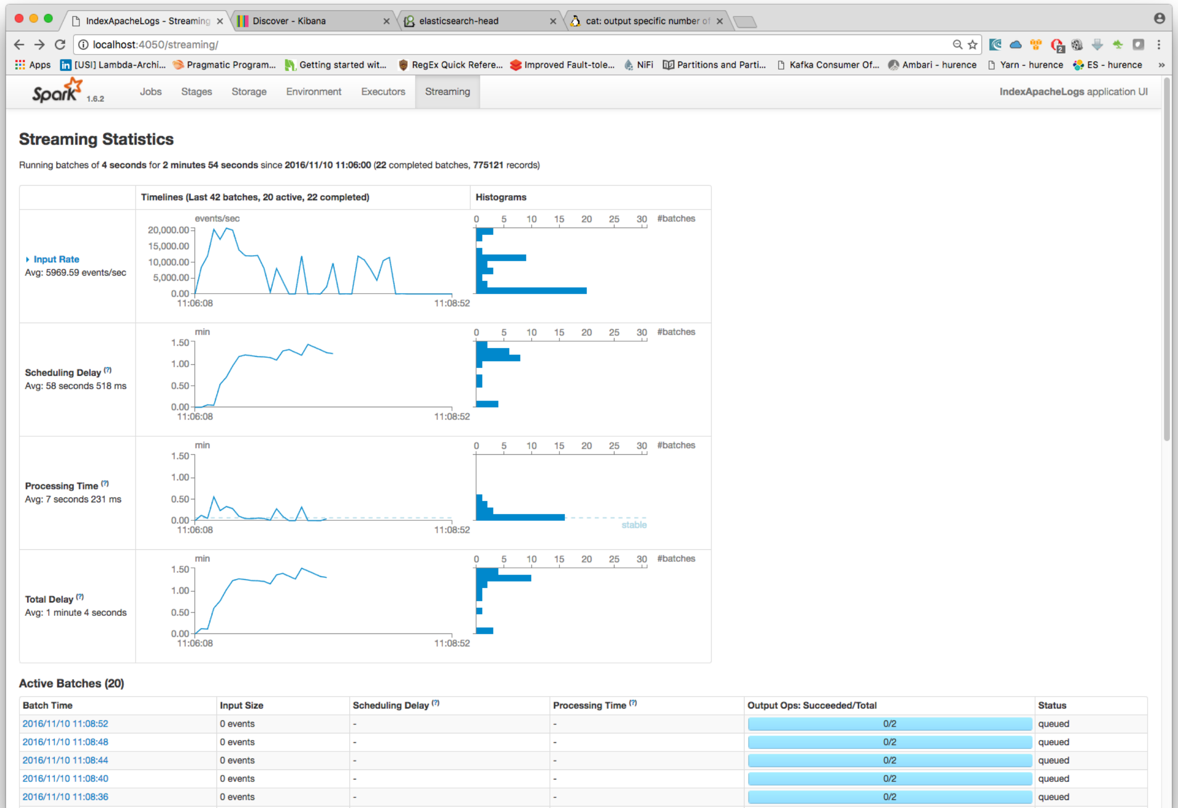


Now that the message have been consumed (you can also verify this thanks to the ActiveMQ console) you can come back to kibana and go to Explore panel for the latest 15' time window you'll only see logisland process_metrics events which give you insights about the processing bandwidth of your streams.

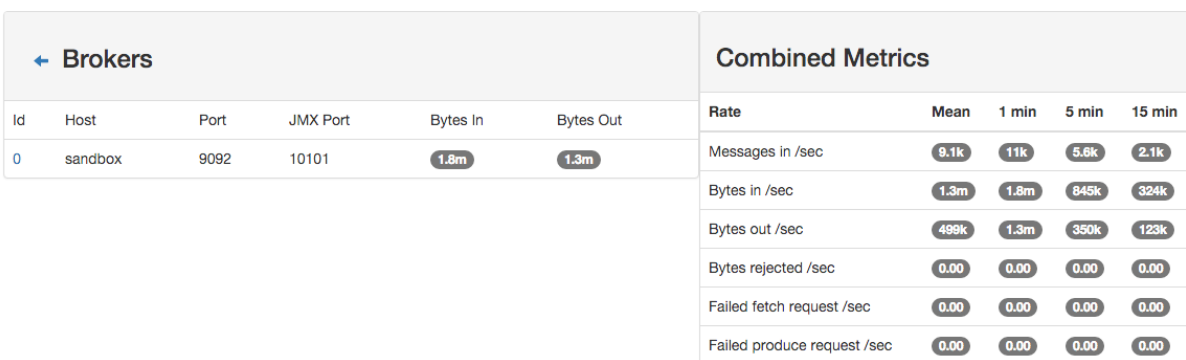


5. Monitor your spark jobs and Kafka topics

Now go to http://sandbox:4050/streaming/ to see how fast Spark can process your data



Another tool can help you to tweak and monitor your processing http://sandbox:9000/



1.8.17 Index blockchain transactions

In the following getting started tutorial, we'll explain you how to leverage logisland connectors flexibility in order process in real time every transaction emitted by the bitcoin blockchain platform and index each record into an elasticsearch platform.

This will allow us to run some dashboarding and visual data analysis as well.

Note: Be sure to know of to launch a logisland Docker environment by reading the prerequisites section

For kafka connect related information please follow as well the connectors section.

---

1. Logisland job setup

Install the blockchain connector if not already done.

```
bin/components.sh -i com.datamountaineer:kafka-connect-blockchain:1.1.1
```

The logisland job for this tutorial is already packaged in the tar.gz assembly and you can find it here for ElasticSearch :

```
vim conf/index-blockchain-transactions.yml
```

We will start by explaining each part of the config file.

The engine

The first section configures the Spark engine (we will use a KafkaStreamProcessingEngine) to run in local mode.

```
engine:
   component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine
   type: engine
   documentation: Index some blockchain transactions with logisland
   configuration:
     spark.app.name: BlockchainTest
     spark.master: local[*]
     spark.driver.memory: 512M
     spark.driver.cores: 1
     spark.executor.memory: 512M
     spark.executor.instances: 4
     spark.executor.cores: 2
     spark.yarn.queue: default
     spark.yarn.maxAppAttempts: 4
     spark.yarn.am.attemptFailuresValidityInterval: 1h
     spark.yarn.max.executor.failures: 20
     spark.yarn.executor.failuresValidityInterval: 1h
     spark.task.maxFailures: 8
     spark.serializer: org.apache.spark.serializer.KryoSerializer
     spark.streaming.batchDuration: 2000
     spark.streaming.backpressure.enabled: false
     spark.streaming.blockInterval: 500
     spark.streaming.kafka.maxRatePerPartition: 10000
     spark.streaming.timeout: -1
     spark.streaming.unpersist: false
     spark.streaming.kafka.maxRetries: 3
     spark.streaming.ui.retainedBatches: 200
     spark.streaming.receiver.writeAheadLog.enable: false
     spark.ui.port: 4040

 The `controllerServiceConfigurations` part is here to define all services that be␣
↪shared by processors within the whole job.

 ==================
 The parsing stream
```

(continues on next page)

(continued from previous page)

```
==================

Here we are going to use a special processor␣
↪(``KafkaConnectStructuredSourceProviderService``) to use the kafka connect source␣
↪as input for the structured stream defined below.

For this example, we are going to use the source *com.datamountaineer.streamreactor.
↪connect.blockchain.source.BlockchainSourceConnector*
that opens a secure websocket connections to the blockchain subscribing to any␣
↪transaction update stream.


.. code-block:: yaml

    ControllerServiceConfigurations:
    - 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
```

Note: Our source is providing structured value hence we convert with LogInslandRecordConverter serializing with Kryo

```
# Kafka sink configuration
- controllerService: kafka_out_service
  component: com.hurence.logisland.stream.spark.structured.provider.
↪KafkaStructuredStreamProviderService
  configuration:
    kafka.output.topics: logisland_raw
    kafka.error.topics: logisland_errors
    kafka.input.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
```

So that, we can now define the *parsing stream* using those source and sink

```
######### parsing stream ##############
- stream: parsing_stream_source
  component: com.hurence.logisland.stream.spark.structured.StructuredStream
  documentation: "Takes records from the kafka source and distributes related␣
↪partitions over a kafka topic. Records are then handed off to the indexing stream"
  configuration:
    read.topics: /a/in
    read.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    read.topics.key.serializer: com.hurence.logisland.serializer.KryoSerializer
    read.topics.client.service: kc_source_service
    write.topics: logisland_raw
    write.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    write.topics.key.serializer: com.hurence.logisland.serializer.KryoSerializer
    write.topics.client.service: kafka_out_service
```

Within this stream, a `FlatMap` processor takes out the value and key (required when using *StructuredStream* as source of records)

```
processorConfigurations:
  - processor: flatten
    component: com.hurence.logisland.processor.FlatMap
    type: processor
    documentation: "Takes out data from record_value"
    configuration:
      keep.root.record: false
      copy.root.record.fields: true
```

The indexing stream

Inside this engine, you will run a Kafka stream of processing, so we set up input/output topics and Kafka/Zookeeper hosts. Here the stream will read all the logs sent in `logisland_raw` topic and push the processing output into `logisland_events` topic.

Note: We want to specify an Avro output schema to validate our output records (and force their types accordingly). It's really for other streams to rely on a schema when processing records from a topic.

We can define some serializers to marshall all records from and to a topic.

```
- stream: parsing_stream_source
  component: com.hurence.logisland.stream.spark.structured.StructuredStream
  documentation: "Takes records from the kafka source and distributes related␣
↪partitions over a kafka topic. Records are then handed off to the indexing stream"
  configuration:
    read.topics: /a/in
    read.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    read.topics.key.serializer: com.hurence.logisland.serializer.KryoSerializer
    read.topics.client.service: kc_source_service
    write.topics: logisland_raw
    write.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    write.topics.key.serializer: com.hurence.logisland.serializer.KryoSerializer
    write.topics.client.service: kafka_out_service
```

Within this stream, a `BulkAddElasticsearch` takes care of indexing a `Record` sending it to elasticsearch.

```
- processor: es_publisher
  component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch
  type: processor
  documentation: a processor that indexes processed events in elasticsearch
  configuration:
    elasticsearch.client.service: elasticsearch_service
    default.index: logisland
    default.type: event
    timebased.index: yesterday
    es.index.field: search_index
    es.type.field: record_type
```

In details, this processor makes use of a `Elasticsearch_5_4_0_ClientService` controller service to interact with our Elasticsearch 5.X backend running locally (and started as part of the docker compose configuration we mentioned above).

Here below its configuration:

```
- controllerService: elasticsearch_service
  component: com.hurence.logisland.service.elasticsearch.Elasticsearch_5_4_0_
↪ClientService
  type: service
  documentation: elasticsearch service
  configuration:
    hosts: sandbox:9300
    cluster.name: es-logisland
    batch.size: 5000
```

2. Launch the script

Connect a shell to your logisland container to launch the following streaming jobs.

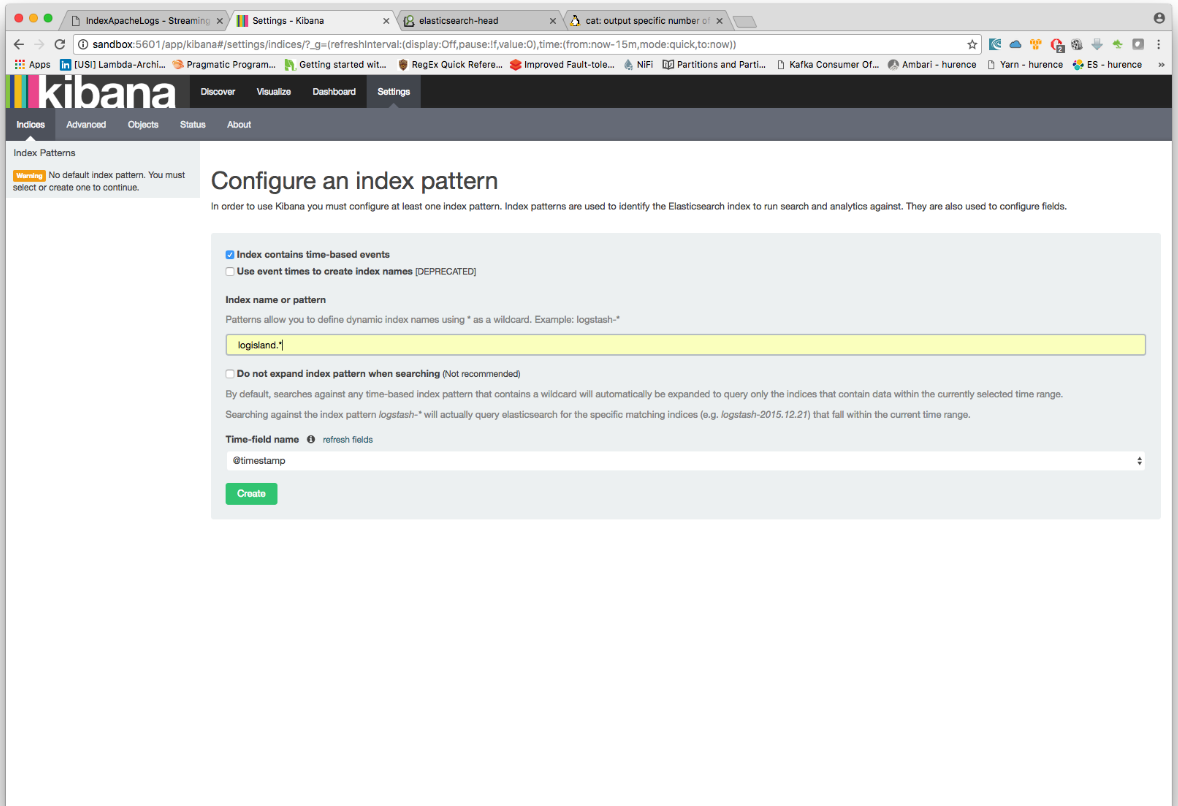
```
bin/logisland.sh --conf conf/index-blockchain-transactions.yml
```

3. Do some insights and visualizations

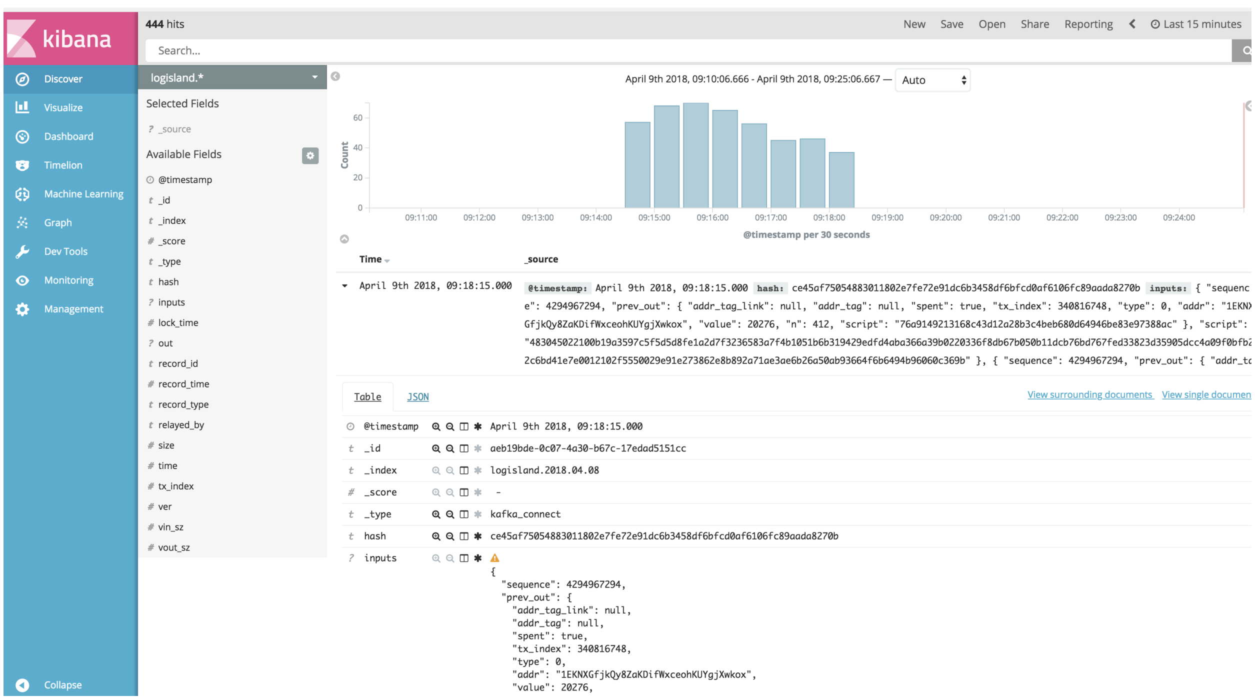
With ElasticSearch, you can use Kibana.

Open up your browser and go to http://sandbox:5601/app/kibana#/ and you should be able to explore the blockchain transactions.

Configure a new index pattern with `logisland.*` as the pattern name and `@timestamp` as the time value field.

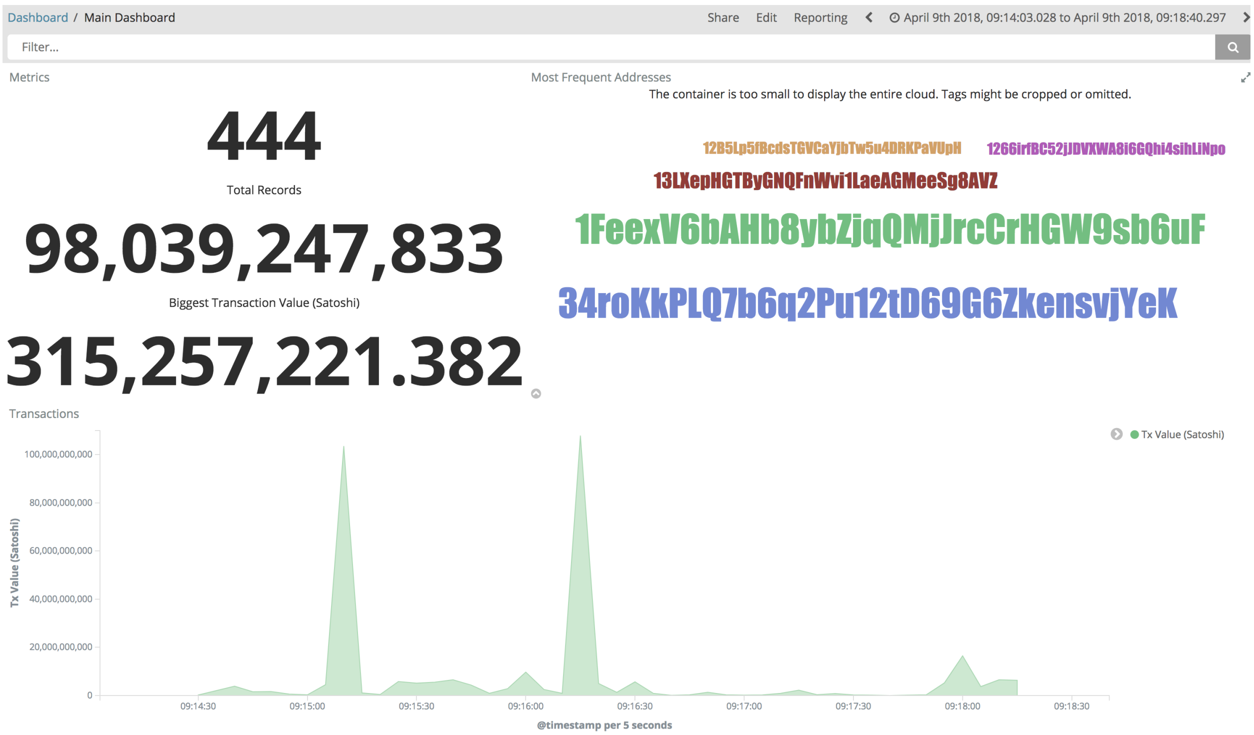


Then if you go to Explore panel for the latest 15' time window you'll only see logisland process_metrics events which give you insights about the processing bandwidth of your streams.



You can try as well to create some basic visualization in order to draw the total satoshi transacted amount (aggregating sums of `out.value` field).

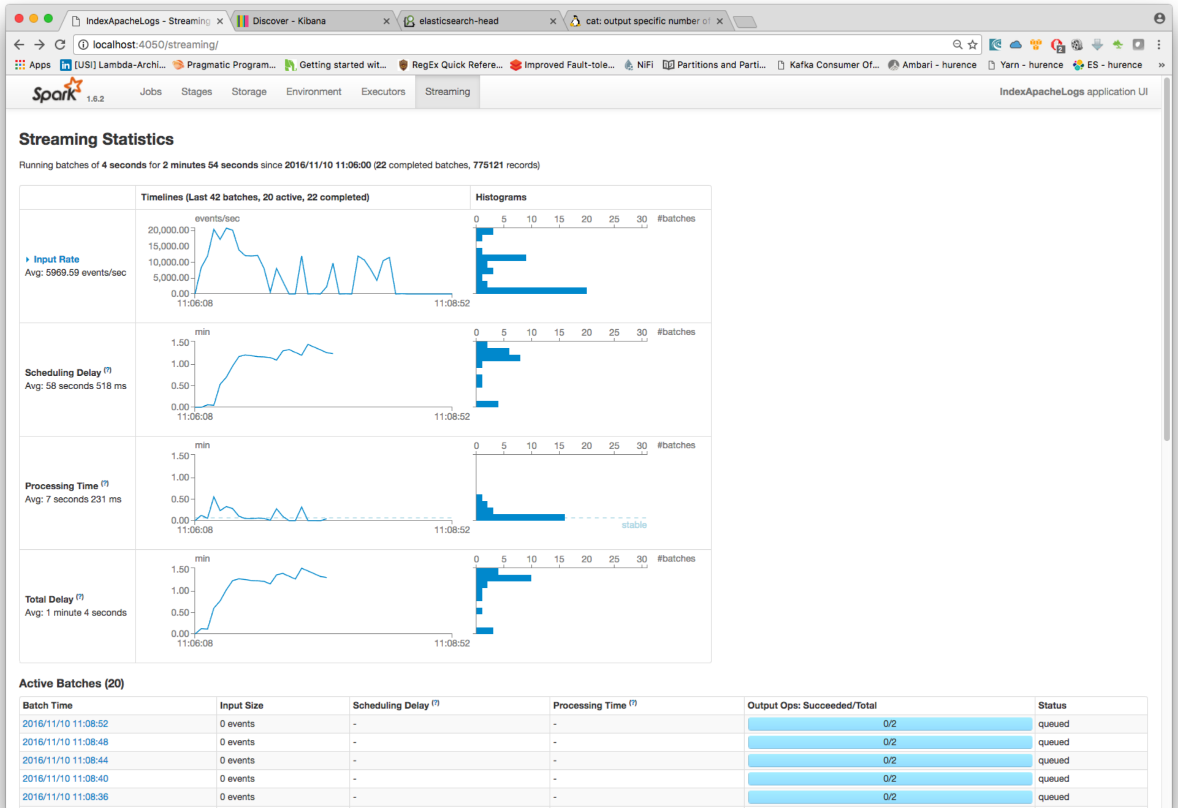
Below a nice example:



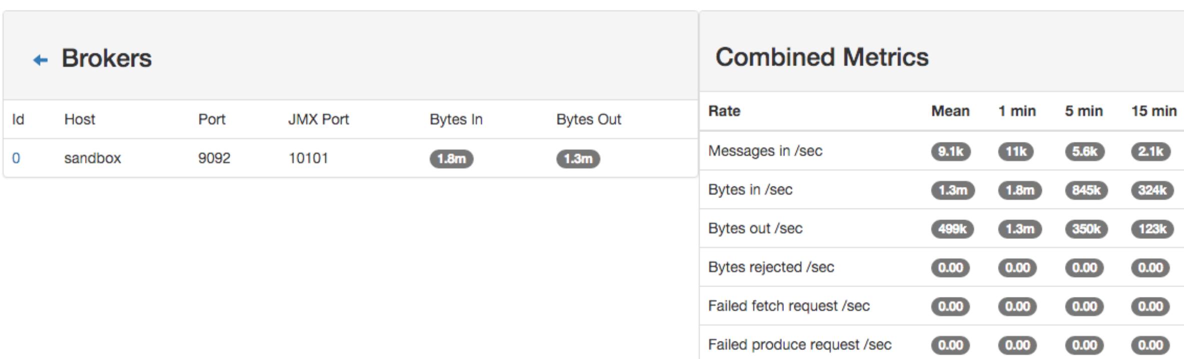
Ready to discover which addresses received most of the money? Give it a try ;-)

4. Monitor your spark jobs and Kafka topics

Now go to http://sandbox:4050/streaming/ to see how fast Spark can process your data



Another tool can help you to tweak and monitor your processing http://sandbox:9000/



1.8.18 Extract Records from Excel File

In the following getting started tutorial we'll drive you through the process of extracting data from any Excel file with LogIsland platform.

Both XLSX and old XLS file format are supported.

Note: Be sure to know of to launch a logisland Docker environment by reading the prerequisites section

Note, it is possible to store data in different datastores. In this tutorial, we will see the case of ElasticSearch only.

1.Install required components

For this tutorial please make sure to already have installed elasticsearch and excel modules. If not you can just do it through the componentes.sh command line:

```
bin/components.sh -i com.hurence.logisland:logisland-processor-elasticsearch:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-service-elasticsearch_5_4_0-
↪client:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-processor-excel:1.1.1
```

2. Logisland job setup

The logisland job for this tutorial is already packaged in the tar.gz assembly and you can find it here for ElasticSearch :

```
docker exec -i -t logisland vim conf/index-excel-spreadsheet.yml
```

We will start by explaining each part of the config file.

An Engine is needed to handle the stream processing. This `conf/extract-excel-data.yml` configuration file defines a stream processing job setup. The first section configures the Spark engine (we will use a KafkaStreamProcessingEngine) to run in local mode with 2 cpu cores and 2G of RAM.

```
  engine:
component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine
type: engine
documentation: Index records of an excel file with LogIsland
configuration:
  spark.app.name: IndexExcelDemo
  spark.master: local[4]
  spark.driver.memory: 1G
  spark.driver.cores: 1
  spark.executor.memory: 2G
  spark.executor.instances: 4
  spark.executor.cores: 2
  spark.yarn.queue: default
  spark.yarn.maxAppAttempts: 4
  spark.yarn.am.attemptFailuresValidityInterval: 1h
  spark.yarn.max.executor.failures: 20
  spark.yarn.executor.failuresValidityInterval: 1h
  spark.task.maxFailures: 8
  spark.serializer: org.apache.spark.serializer.KryoSerializer
  spark.streaming.batchDuration: 1000
  spark.streaming.backpressure.enabled: false
  spark.streaming.unpersist: false
  spark.streaming.blockInterval: 500
  spark.streaming.kafka.maxRatePerPartition: 3000
  spark.streaming.timeout: -1
  spark.streaming.unpersist: false
  spark.streaming.kafka.maxRetries: 3
  spark.streaming.ui.retainedBatches: 200
```

(continues on next page)

(continued from previous page)

```
  spark.streaming.receiver.writeAheadLog.enable: false
  spark.ui.port: 4050
```

The *controllerServiceConfigurations* part is here to define all services that be shared by processors within the whole job, here an Elasticsearch service that will be used later in the `BulkAddElasticsearch` processor.

```
- controllerService: elasticsearch_service
  component: com.hurence.logisland.service.elasticsearch.Elasticsearch_5_4_0_
↪ClientService
  type: service
  documentation: elasticsearch service
  configuration:
    hosts: sandbox:9300
    cluster.name: es-logisland
    batch.size: 5000
```

Inside this engine you will run a Kafka stream of processing, so we setup input/output topics and Kafka/Zookeeper hosts. Here the stream will read all the logs sent in `logisland_raw` topic and push the processing output into `logisland_events` topic.

We can define some serializers to marshall all records from and to a topic. We assume that the stream will be serializing the input file as a byte array in a single record. Reason why we will use a ByteArraySerialiser in the configuration below.

```
# main processing stream
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that converts raw excel file content into structured log␣
↪records
  configuration:
    kafka.input.topics: logisland_raw
    kafka.output.topics: logisland_events
    kafka.error.topics: logisland_errors
    kafka.input.topics.serializer: com.hurence.logisland.serializer.
↪BytesArraySerializer
    kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
```

Within this stream, an `ExcelExtract` processor takes a byte array excel file content and computes a list of `Record`.

```
# parse excel cells into records
    - processor: excel_parser
      component: com.hurence.logisland.processor.excel.ExcelExtract
      type: parser
      documentation: a parser that produce events from an excel file
      configuration:
        record.type: excel_record
        skip.rows: 1
        field.names: segment,country,product,discount_band,units_sold,manufacturing,
↪sale_price,gross_sales,discounts,sales,cogs,profit,record_time,month_number,month_
↪name,year
```

This stream will process log entries as soon as they will be queued into *logisland_raw* Kafka topics, each log will be parsed as an event which will be pushed back to Kafka in the `logisland_events` topic.

Note: Please note that we are mapping the excel column *Date* to be the timestamp of the produced record (*record_time* field) in order to use this as time reference in elasticsearch/kibana (see below).

The second processor will handle `Records` produced by the `ExcelExtract` to index them into elasticsearch

```
# add to elasticsearch
- processor: es_publisher
  component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch
  type: processor
  documentation: a processor that trace the processed events
  configuration:
    elasticsearch.client.service: elasticsearch_service
    default.index: logisland
    default.type: event
    timebased.index: yesterday
    es.index.field: search_index
    es.type.field: record_type
```

3. Launch the script

For this tutorial we will handle an excel file. We will process it with an ExcelExtract that will produce a bunch of Records and we'll send them to Elastiscearch Connect a shell to your logisland container to launch the following streaming jobs.

For ElasticSearch :

```
docker exec -i -t logisland bin/logisland.sh --conf conf/index-excel-spreadsheet.yml
```

4. Inject an excel file into the system

Now we're going to send a file to `logisland_raw` Kafka topic.

For testing purposes, we will use kafkacat, a *generic command line non-JVM Apache Kafka producer and consumer* which can be easily installed.

Note: Sending raw files through kafka is not recommended for production use since kafka is designed for high throughput and not big message size.

The configuration above is suited to work with the example file *Financial Sample.xlsx*.

Let's send this file in a single message to LogIsland with kafkacat to `logisland_raw` Kafka topic

```
kafkacat -P -t logisland_raw -v -b sandbox:9092 ./Financial\ Sample.xlsx
```

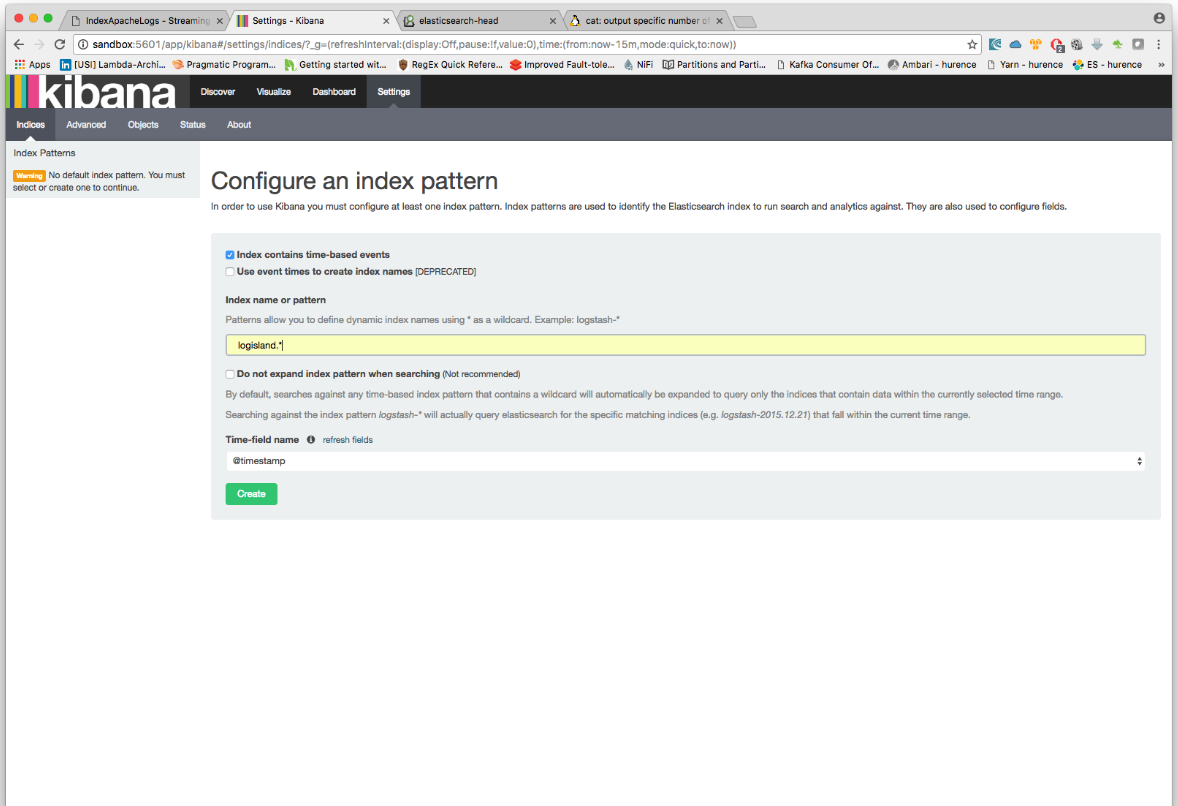
5. Inspect the logs

Kibana

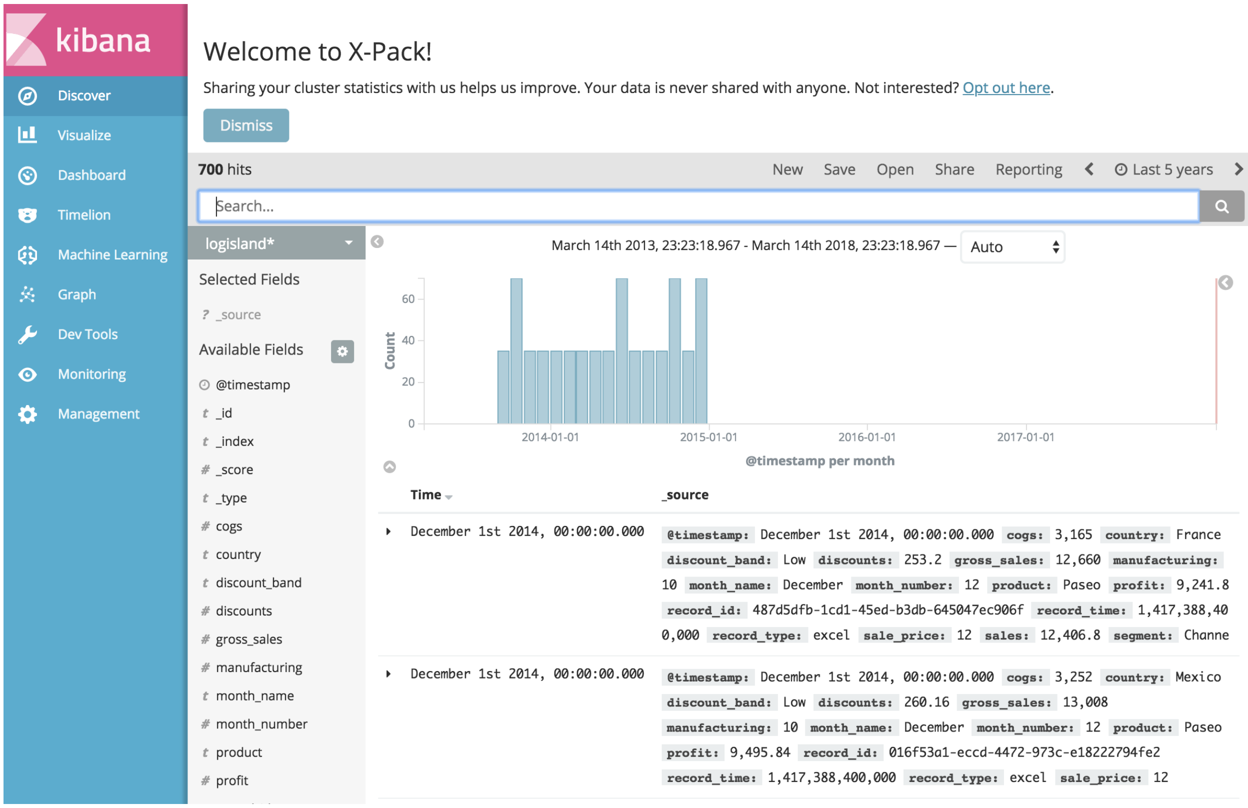
With ElasticSearch, you can use Kibana.

Open up your browser and go to http://sandbox:5601/ and you should be able to explore your excel records.

Configure a new index pattern with `logisland.*` as the pattern name and `@timestamp` as the time value field.



Then if you go to Explore panel for the latest 5 years time window. You are now able to play with the indexed data.



Thanks logisland! :-)

1.8.19 IIoT with MQTT and Logisland Data-Historian

In the following getting tutorial we'll drive you through the process of IIoT enablement with LogIsland platform.

Note: Be sure to know of to launch a logisland Docker environment by reading the prerequisites section

docker run -td –name kapua-sql -p 8181:8181 -p 3306:3306 kapua/kapua-sql:0.3.2 docker run -td –name kapua-elasticsearch -p 9200:9200 -p 9300:9300 elasticsearch:5.4.0 -Ecluster.name=kapua-datastore -Ediscovery.type=single-node -Etransport.host=_site_ -Etransport.ping_schedule=-1 -Etransport.tcp.connect_timeout=30s docker run -td –name kapua-broker –link kapua-sql:db –link kapua-elasticsearch:es –env commons.db.schema.update=true -p 1883:1883 -p 61614:61614 kapua/kapua-broker:0.3.2 docker run -td –name kapua-console –link kapua-sql:db –link kapua-broker:broker –link kapua-elasticsearch:es –env commons.db.schema.update=true -p 8080:8080 kapua/kapua-console:0.3.2 docker run -td –name kapua-api –link kapua-sql:db –link kapua-broker:broker –link kapua-elasticsearch:es –env commons.db.schema.update=true -p 8081:8080 kapua/kapua-api:0.3.2

docker run -td –name logisland-historian -p 8983:8983 hurence/chronix:latest

docker run -it –env MQTT_BROKER_URL=tcp://10.20.20.87:1883 –env SOLR_CONNECTION=http://10.20.20.87:8983/solr –name kapua-logisland hurence/logisland:0.12.0 bin/logisland.sh –conf conf/mqtt-to-historian.yml

Note, it is possible to store data in different datastores. In this tutorial, we will see the case of ElasticSearch and Solr.

1. Logisland job setup

The logisland job for this tutorial is already packaged in the tar.gz assembly and you can find it here for ElasticSearch :

```
docker exec -i -t logisland vim conf/index-apache-logs.yml
```

And here for Solr :

```
docker exec -i -t logisland vim conf/index-apache-logs-solr.yml
```

We will start by explaining each part of the config file.

An Engine is needed to handle the stream processing. This `conf/index-apache-logs.yml` configuration file defines a stream processing job setup. The first section configures the Spark engine (we will use a KafkaStreamProcessingEngine) to run in local mode with 2 cpu cores and 2G of RAM.

```
engine:
  component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine
  type: engine
  documentation: Index some apache logs with logisland
  configuration:
    spark.app.name: IndexApacheLogsDemo
    spark.master: local[2]
    spark.driver.memory: 1G
    spark.driver.cores: 1
    spark.executor.memory: 2G
    spark.executor.instances: 4
    spark.executor.cores: 2
    spark.yarn.queue: default
    spark.yarn.maxAppAttempts: 4
    spark.yarn.am.attemptFailuresValidityInterval: 1h
    spark.yarn.max.executor.failures: 20
    spark.yarn.executor.failuresValidityInterval: 1h
    spark.task.maxFailures: 8
    spark.serializer: org.apache.spark.serializer.KryoSerializer
    spark.streaming.batchDuration: 1000
    spark.streaming.backpressure.enabled: false
    spark.streaming.unpersist: false
    spark.streaming.blockInterval: 500
    spark.streaming.kafka.maxRatePerPartition: 3000
    spark.streaming.timeout: -1
    spark.streaming.unpersist: false
    spark.streaming.kafka.maxRetries: 3
    spark.streaming.ui.retainedBatches: 200
    spark.streaming.receiver.writeAheadLog.enable: false
    spark.ui.port: 4050
```

The *controllerServiceConfigurations* part is here to define all services that be shared by processors within the whole job, here an Elasticsearch service that will be used later in the `BulkAddElasticsearch` processor.

```
- controllerService: elasticsearch_service
  component: com.hurence.logisland.service.elasticsearch.Elasticsearch_5_4_0_
↪ClientService
  type: service
  documentation: elasticsearch service
  configuration:
    hosts: sandbox:9300
```

(continues on next page)

(continued from previous page)

```
    cluster.name: es-logisland
    batch.size: 5000
```

Inside this engine you will run a Kafka stream of processing, so we setup input/output topics and Kafka/Zookeeper hosts. Here the stream will read all the logs sent in `logisland_raw` topic and push the processing output into `logisland_events` topic.

Note: We want to specify an Avro output schema to validate our ouput records (and force their types accordingly). It's really for other streams to rely on a schema when processing records from a topic.

We can define some serializers to marshall all records from and to a topic.

```
- stream: parsing_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that converts raw apache logs into structured log records
  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
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
```

Within this stream a `SplitText` processor takes a log line as a String and computes a `Record` as a sequence of fields.

```
# parse apache logs
- processor: apache_parser
  component: com.hurence.logisland.processor.SplitText
  type: parser
  documentation: a parser that produce events from an apache log REGEX
  configuration:
    value.regex: (\S+)\s+(\S+)\s+(\S+)\s+\[([\w:\/]+\s[+\-]\d{4})\]\s+
↪"(\S+)\s+(\S+)\s*(\S*)"\s+(\S+)\s+(\S+)
    value.fields: src_ip,identd,user,record_time,http_method,http_query,http_version,
↪http_status,bytes_out
```

This stream will process log entries as soon as they will be queued into *logisland_raw* Kafka topics, each log will be parsed as an event which will be pushed back to Kafka in the `logisland_events` topic.

The second processor will handle `Records` produced by the `SplitText` to index them into elasticsearch

```
# add to elasticsearch
- processor: es_publisher
  component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch
  type: processor
  documentation: a processor that trace the processed events
  configuration:
    elasticsearch.client.service: elasticsearch_service
```

(continues on next page)

(continued from previous page)

```
    default.index: logisland
    default.type: event
    timebased.index: yesterday
    es.index.field: search_index
    es.type.field: record_type
```

Solr

In the case of Solr, we have to declare another service :

```
# Datastore service using Solr 6.6.2 - 5.5.5 also available
- controllerService: datastore_service
  component: com.hurence.logisland.service.solr.Solr_6_6_2_ClientService
  type: service
  documentation: "SolR 6.6.2 service"
  configuration:
    solr.cloud: false
    solr.connection.string: http://sandbox:8983/solr
    solr.collection: solr-apache-logs
    solr.concurrent.requests: 4
    flush.interval: 2000
    batch.size: 1000
```

With this configuration, Solr is used in standalone mode but you can also use the cloud mode by changing the corresponding config.

Note: You have to create the core/collection manually with the following fields : `src_ip`, `identd`, `user`, `bytes_out`, `http_method`, `http_version`, `http_query`, `http_status`

Then, the second processor have to send data to Solr :

```
# all the parsed records are added to solr by bulk
- processor: solr_publisher
  component: com.hurence.logisland.processor.datastore.BulkPut
  type: processor
  documentation: "indexes processed events in SolR"
  configuration:
    datastore.client.service: datastore_service
```

2. Launch the script

For this tutorial we will handle some apache logs with a splitText parser and send them to Elastiscearch Connect a shell to your logisland container to launch the following streaming jobs.

For ElasticSearch :

```
docker exec -i -t logisland bin/logisland.sh --conf conf/index-apache-logs.yml
```

For Solr :

```
docker exec -i -t logisland bin/logisland.sh --conf conf/index-apache-logs-solr.yml
```

3. Inject some Apache logs into the system

Now we're going to send some logs to `logisland_raw` Kafka topic.

We could setup a logstash or flume agent to load some apache logs into a kafka topic but there's a super useful tool in the Kafka ecosystem : kafkacat, a *generic command line non-JVM Apache Kafka producer and consumer* which can be easily installed.

If you don't have your own httpd logs available, you can use some freely available log files from NASA-HTTP web site access:

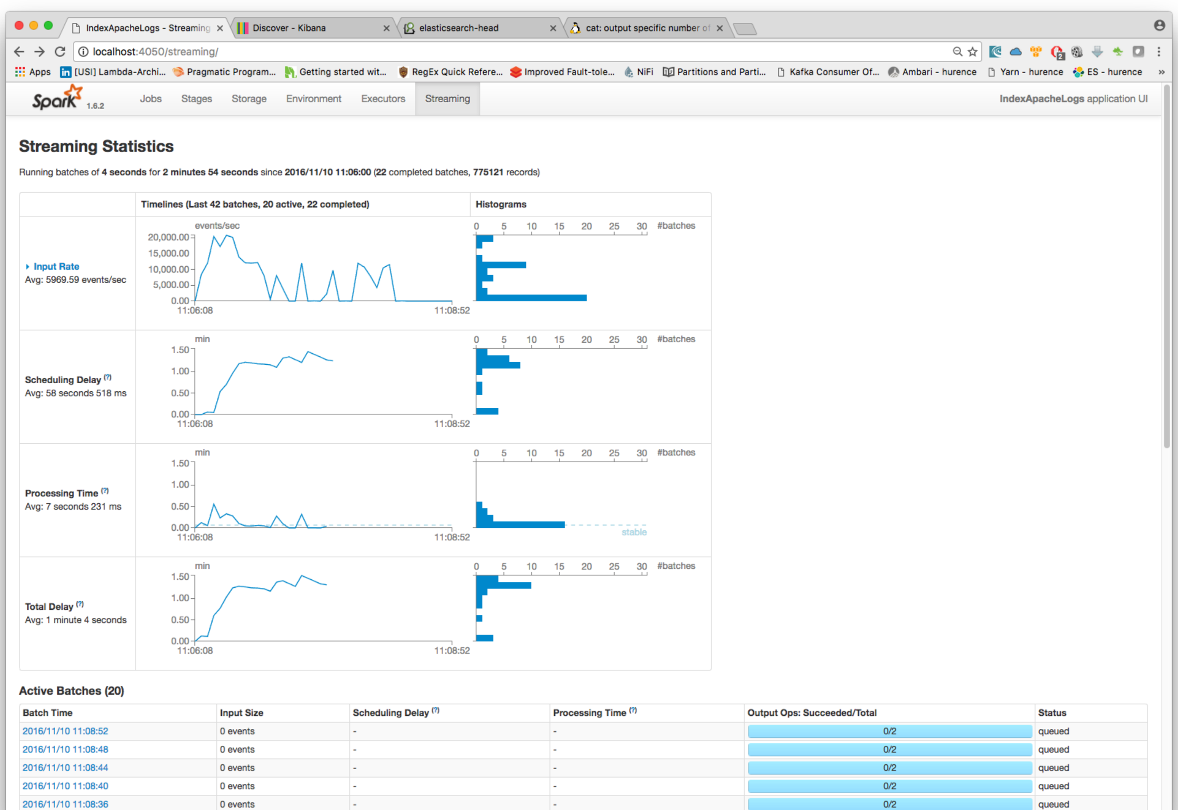
- Jul 01 to Jul 31, ASCII format, 20.7 MB gzip compressed
- Aug 04 to Aug 31, ASCII format, 21.8 MB gzip compressed

Let's send the first 500000 lines of NASA http access over July 1995 to LogIsland with kafkacat to `logisland_raw` Kafka topic

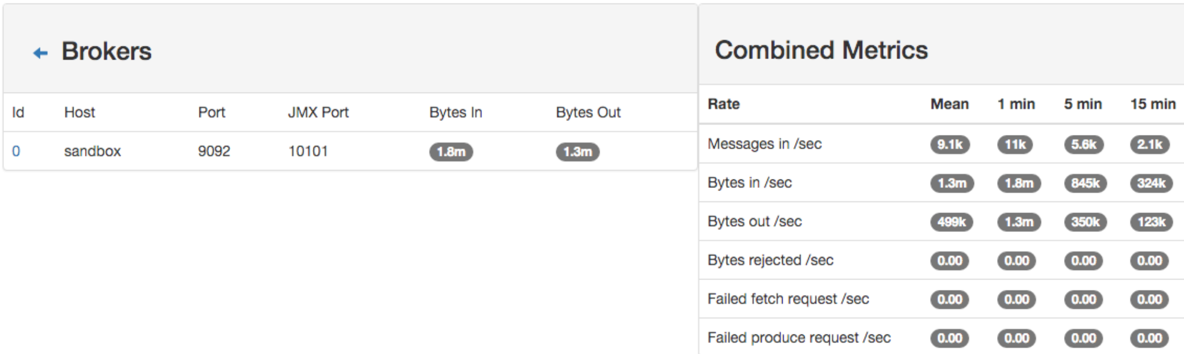
```
cd /tmp
wget ftp://ita.ee.lbl.gov/traces/NASA_access_log_Jul95.gz
gunzip NASA_access_log_Jul95.gz
head -500000 NASA_access_log_Jul95 | kafkacat -b sandbox:9092 -t logisland_raw
```

4. Monitor your spark jobs and Kafka topics

Now go to http://sandbox:4050/streaming/ to see how fast Spark can process your data



Another tool can help you to tweak and monitor your processing http://sandbox:9000/



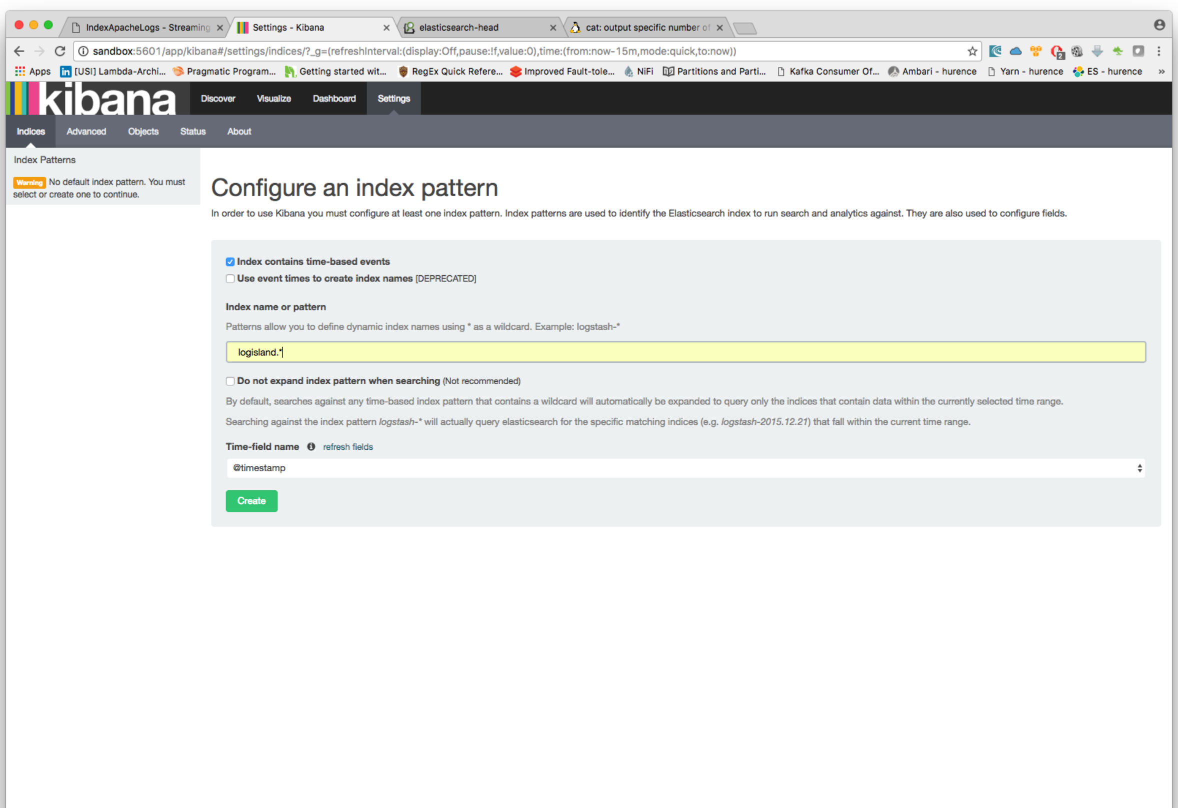
5. Inspect the logs

Kibana

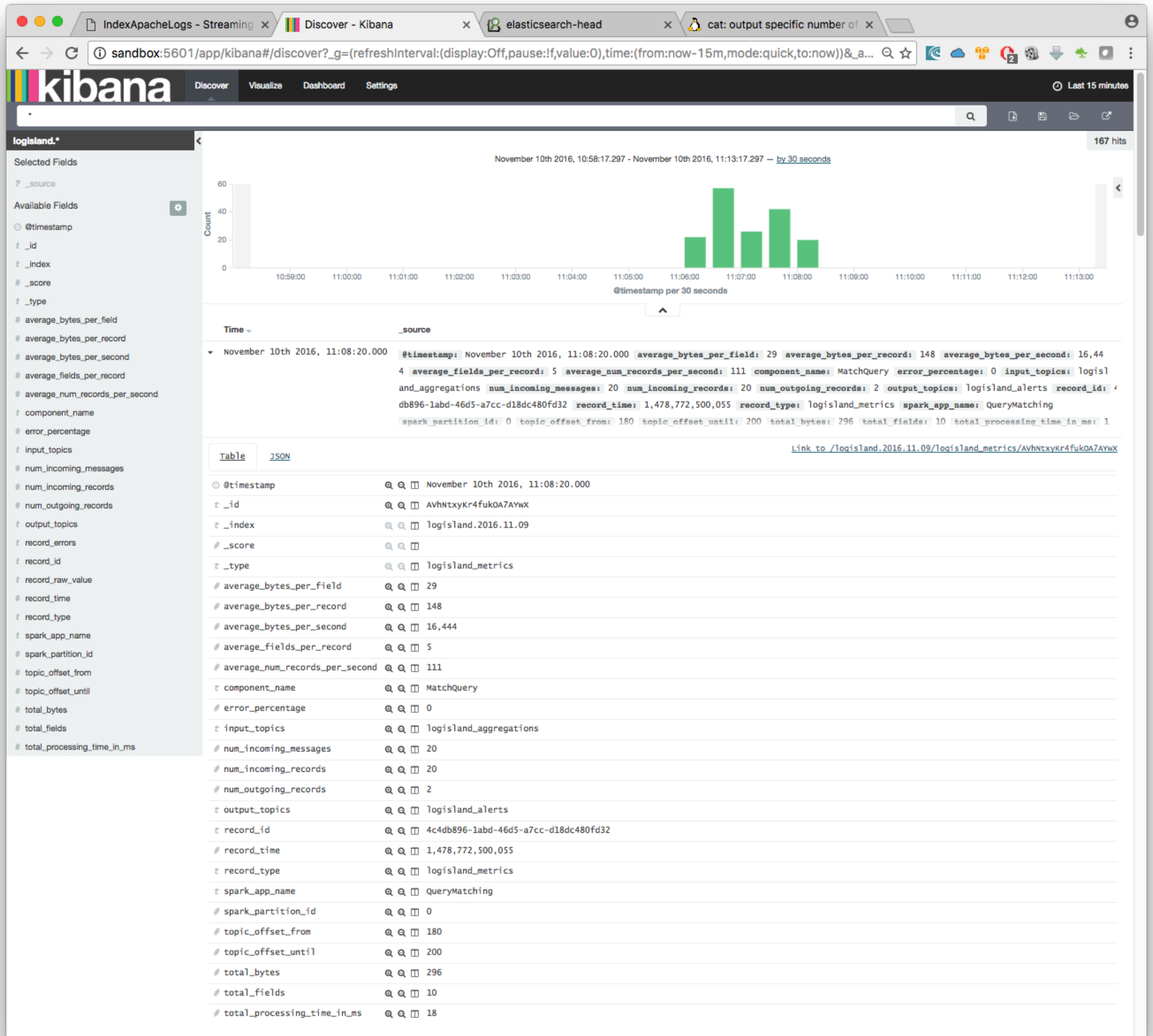
With ElasticSearch, you can use Kibana.

Open up your browser and go to http://sandbox:5601/ and you should be able to explore your apache logs.

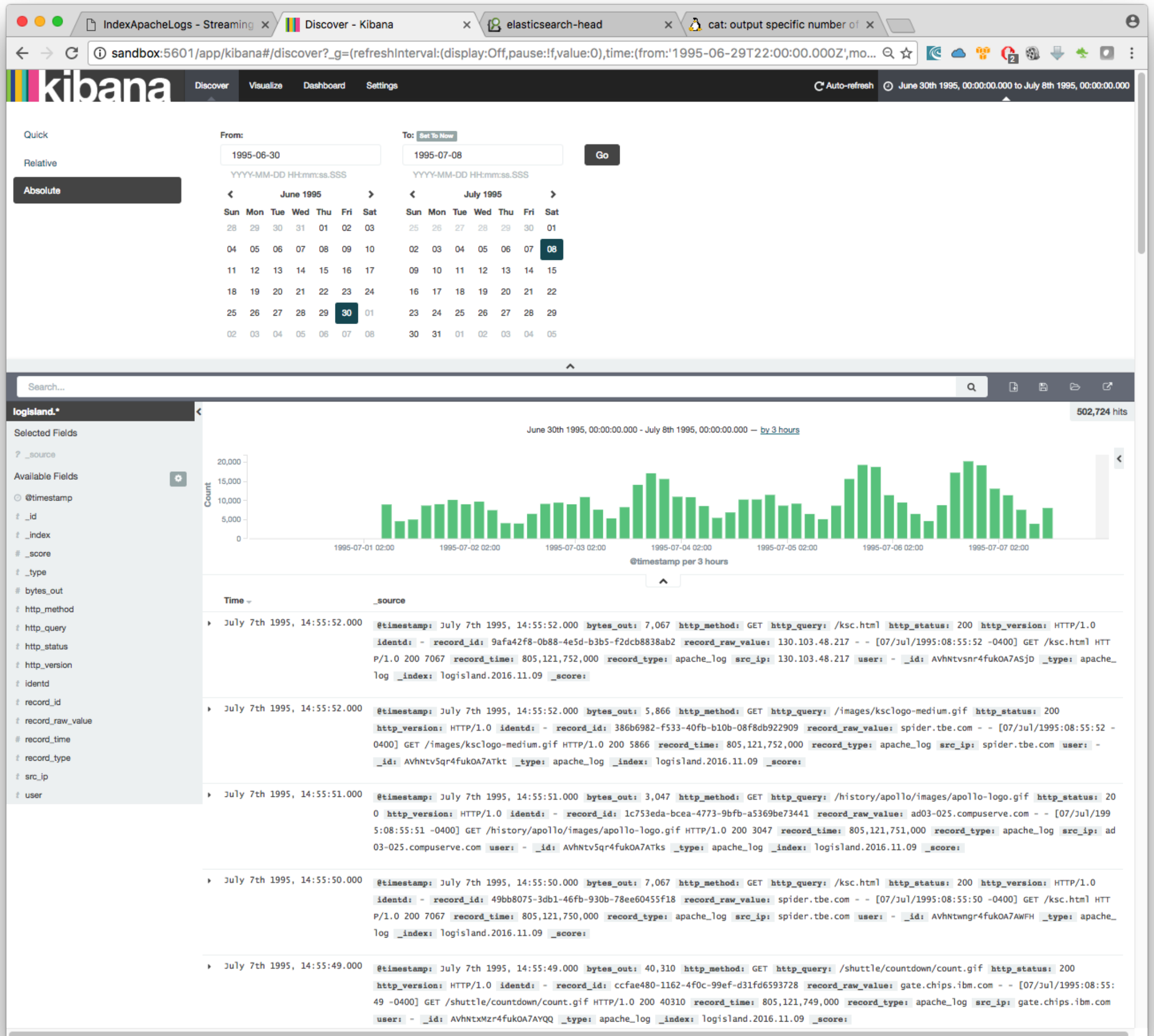
Configure a new index pattern with `logisland.*` as the pattern name and `@timestamp` as the time value field.



Then if you go to Explore panel for the latest 15' time window you'll only see logisland process_metrics events which give you insights about the processing bandwidth of your streams.



As we explore data logs from july 1995 we'll have to select an absolute time filter from 1995-06-30 to 1995-07-08 to see the events.

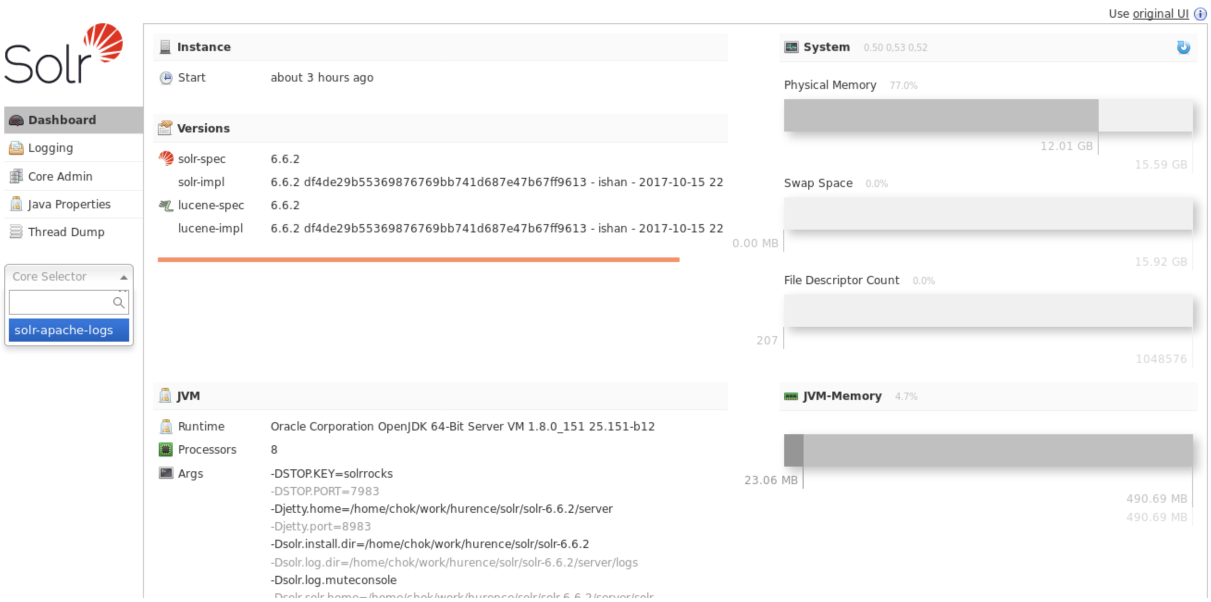


Solr

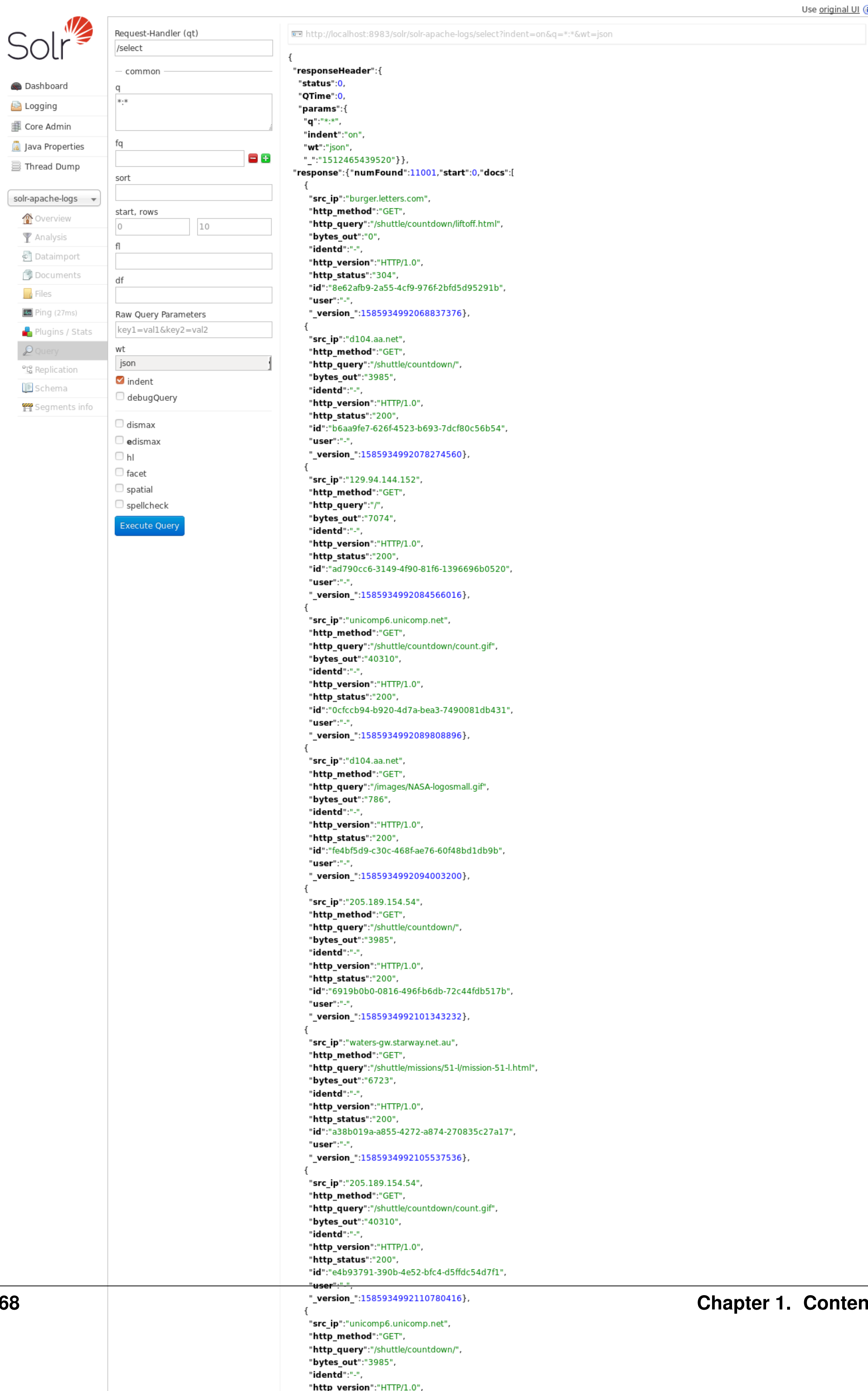
With Solr, you can directly use the solr web ui.

Open up your browser and go to http://sandbox:8983/solr and you should be able to view your apache logs.

In non cloud mode, use the core selector, to select the core `` `solr-apache-logs` `` :



Then, go to query and by clicking to Execute Query, you will see some data from your Apache logs :


Use original UI
Solr
Dashboard
Logging
Core Admin
Java Properties
Thread Dump
solr-apache-logs
Overview
Analysis
Dataimport
Documents
Files
Ping (27ms)
Plugins / Stats
Query
Replication
Schema
Segments info
Request-Handler (qt)
/select
common
q
:
fq
sort
start, rows
0
10
fl
df
Raw Query Parameters
key1=val1&key2=val2
wt
json
indent
debugQuery
dismax
edismax
hl
facet
spatial
spellcheck
Execute Query
http://localhost:8983/solr/solr-apache-logs/select?indent=on&q=*:*&wt=json
{
"responseHeader":{
"status":0,
"QTime":0,
"params":{
"q":"*:*",
"indent":"on",
"wt":"json",
"_":"1512465439520"}},
"response":{"numFound":11001,"start":0,"docs":[
{
"src_ip":"burger.letters.com",
"http_method":"GET",
"http_query":"/shuttle/countdown/liftoff.html",
"bytes_out":"0",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"304",
"id":"8e62afb9-2a55-4cf9-976f-2bfd5d95291b",
"user":"-",
"_version_":1585934992068837376},
{
"src_ip":"d104.aa.net",
"http_method":"GET",
"http_query":"/shuttle/countdown/",
"bytes_out":"3985",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"b6aa9fe7-626f-4523-b693-7dcf80c56b54",
"user":"-",
"_version_":1585934992078274560},
{
"src_ip":"129.94.144.152",
"http_method":"GET",
"http_query":"/",
"bytes_out":"7074",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"ad790cc6-3149-4f90-81f6-1396696b0520",
"user":"-",
"_version_":1585934992084566016},
{
"src_ip":"unicomp6.unicomp.net",
"http_method":"GET",
"http_query":"/shuttle/countdown/count.gif",
"bytes_out":"40310",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"0cfccb94-b920-4d7a-bea3-7490081db431",
"user":"-",
"_version_":1585934992089808896},
{
"src_ip":"d104.aa.net",
"http_method":"GET",
"http_query":"/images/NASA-logosmall.gif",
"bytes_out":"786",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"fe4bf5d9-c30c-468f-ae76-60f48bd1db9b",
"user":"-",
"_version_":1585934992094003200},
{
"src_ip":"205.189.154.54",
"http_method":"GET",
"http_query":"/shuttle/countdown/",
"bytes_out":"3985",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"6919b0b0-0816-496f-b6db-72c44fdb517b",
"user":"-",
"_version_":1585934992101343232},
{
"src_ip":"waters-gw.starway.net.au",
"http_method":"GET",
"http_query":"/shuttle/missions/51-l/mission-51-l.html",
"bytes_out":"6723",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"a38b019a-a855-4272-a874-270835c27a17",
"user":"-",
"_version_":1585934992105537536},
{
"src_ip":"205.189.154.54",
"http_method":"GET",
"http_query":"/shuttle/countdown/count.gif",
"bytes_out":"40310",
"identd":"-",
"http_version":"HTTP/1.0",
"http_status":"200",
"id":"e4b93791-390b-4e52-bfc4-d5ffdc54d7f1",
"user":"-",
"_version_":1585934992110780416},
{
"src_ip":"unicomp6.unicomp.net",
"http_method":"GET",
"http_query":"/shuttle/countdown/",
"bytes_out":"3985",
"identd":"-",
"http_version":"HTTP/1.0",

1.8.20 IIoT with OPC and Logisland

In this tutorial we'll show you how to ingest IIoT data from an OPC-UA server and process it with Logisland, storing everything into an elasticsearch database.

In particular, we'll use the Prosys OPC-UA simulation server you can download for free here

Note: You will need to have a logisland Docker environment. Please follow the prerequisites section for more information.

Please also remember to always turn on the simulation server before running the logisland job.

1.Install required components

For this tutorial please make sure to already have installed elasticsearch and OPC modules. If not you can just do it through the componentes.sh command line:

```
bin/components.sh -i com.hurence.logisland:logisland-processor-elasticsearch:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-service-elasticsearch_5_4_0-
↪client:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-connector-opc:1.1.1
```

2. Logisland job setup

The logisland job for this tutorial is already packaged in the tar.gz assembly and you can find it here for ElasticSearch :

```
docker exec -i -t logisland vim conf/opc-iiot.yml
```

We will start by explaining each part of the config file.

The first section configures the Spark engine (we will use a KafkaStreamProcessingEngine) to run in local mode with 1 cpu cores and 512M of RAM.

```
engine:
  component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine
  type: engine
  documentation: Index some OPC-UA tagw with Logisland
  configuration:
  spark.app.name: OpcUaLogisland
  spark.master: local[2]
  spark.driver.memory: 512M
  spark.driver.cores: 1
  spark.executor.memory: 512M
  spark.executor.instances: 4
  spark.executor.cores: 1
  spark.yarn.queue: default
  spark.yarn.maxAppAttempts: 4
  spark.yarn.am.attemptFailuresValidityInterval: 1h
  spark.yarn.max.executor.failures: 20
  spark.yarn.executor.failuresValidityInterval: 1h
  spark.task.maxFailures: 8
```

(continues on next page)

(continued from previous page)

```
spark.serializer: org.apache.spark.serializer.KryoSerializer
spark.streaming.batchDuration: 3000
spark.streaming.backpressure.enabled: false
spark.streaming.blockInterval: 500
spark.streaming.kafka.maxRatePerPartition: 10000
spark.streaming.timeout: -1
spark.streaming.unpersist: false
spark.streaming.kafka.maxRetries: 3
spark.streaming.ui.retainedBatches: 200
spark.streaming.receiver.writeAheadLog.enable: false
spark.ui.port: 4040
```

The *controllerServiceConfigurations* part is here to define all services that be shared by processors within the whole job.

Here we have the OPC-UA source with all the connection parameters.

```
- controllerService: kc_source_service
  component: com.hurence.logisland.stream.spark.provider.
↪KafkaConnectStructuredSourceProviderService
   documentation: Kafka connect OPC-UA source service
   type: service
   configuration:
     kc.connector.class: com.hurence.logisland.connect.opc.ua.OpcUaSourceConnector
     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.offset.backing.store: memory
     kc.connector.properties: |
       session.publicationRate=PT1S
       connection.socketTimeoutMillis=10000
       server.uri=opc.tcp://localhost:53530/OPCUA/SimulationServer
       tags.id=ns=5;s=Sawtooth1
       tags.sampling.rate=PT0.5S
       tags.stream.mode=SUBSCRIBE
```

In particular, we have

- A tag to be read: *"ns=5;s=Sawtooth1"*
- The tag will be subscribed and sampled each 0.5s
- The data will be published by the opc server each second (*session.publicationRate*)
- Please use your own opc server uri, in our case opc.tcp://localhost:53530/OPCUA/SimulationServer

Full connector documentation is on javadoc of class `com.hurence.logisland.connect.opc.ua.OpcUaSourceConnector`

Then we also define her Elasticsearch service that will be used later in the `BulkAddElasticsearch` processor.

```
- controllerService: elasticsearch_service
  component: com.hurence.logisland.service.elasticsearch.Elasticsearch_5_4_0_
↪ClientService
```

(continues on next page)

(continued from previous page)

```
  type: service
  documentation: elasticsearch service
  configuration:
    hosts: ${ES_HOSTS}
    cluster.name: ${ES_CLUSTER_NAME}
    batch.size: 5000
```

Inside this engine you will run a spark structured stream, taking records from the previously defined source and letting data flow through the processing pipeline till the console output.

```
- stream: ingest_stream
    component: com.hurence.logisland.stream.spark.structured.StructuredStream
    configuration:
      read.topics: /a/in
      read.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
      read.topics.key.serializer: com.hurence.logisland.serializer.StringSerializer
      read.topics.client.service: kc_source_service
      write.topics: /a/out
      write.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
      write.topics.key.serializer: com.hurence.logisland.serializer.StringSerializer
      write.topics.client.service: console_service
```

And now it's time to describe the parsing pipeline.

First, we need to extract the record thanks to a `FlatMap` processor

```
- processor: flatten
  component: com.hurence.logisland.processor.FlatMap
  type: processor
  documentation: "extract from root record"
  configuration:
    keep.root.record: false
    copy.root.record.fields: true
```

Now that the record is well-formed, we want to set the record time to be the same of the one given by the source (and stored on the field *tag_sampled_timestamp*).

For this, we use a `NormalizeFields` processor.

```
- processor: rename_fields
  component: com.hurence.logisland.processor.NormalizeFields
  type: processor
  documentation: "set record time to tag server generation time"
  configuration:
    conflict.resolution.policy: overwrite_existing
    record_time: tag_sampled_timestamp
```

Then, the last processor will index our records into elasticsearch

```
# add to elasticsearch
- processor: es_publisher
  component: com.hurence.logisland.processor.elasticsearch.BulkAddElasticsearch
  type: processor
  documentation: a processor that trace the processed events
  configuration:
    elasticsearch.client.service: elasticsearch_service
    default.index: logisland
```

(continues on next page)

(continued from previous page)

```
    default.type: event
    timebased.index: yesterday
    es.index.field: search_index
    es.type.field: record_type
```

3. Launch the script

Just ensure the Prosys OPC-UA server is up and running and that on the *Simulation* tab the simulation is ticked.

Then you can use the docker-compose file **docker-compose-opc-iiot.yml** available in the tar gz assembly in conf directory.

Note: If your simulation server is hosted on local and the hostname is different from 'localhost'. For example if your server uri is 'opc.tcp://${hostname}:53530/OPCUA/SimulationServer'. You can add it to logisland container add a extra_hosts properties to logisland container in docker-compose file so that it is accessible from the container.

```
logisland:
  network_mode: host
  image: hurence/logisland:1.1.1
  command: tail -f bin/logisland.sh
  environment:
    ZK_QUORUM: localhost:2181
    ES_HOSTS: localhost:9300
    ES_CLUSTER_NAME: es-logisland
  extra_hosts:
    - "${hostname}:127.0.0.1"
```

Then you can execute:

```
docker exec -i -t logisland bin/logisland.sh --conf conf/opc-iiot.yml
```

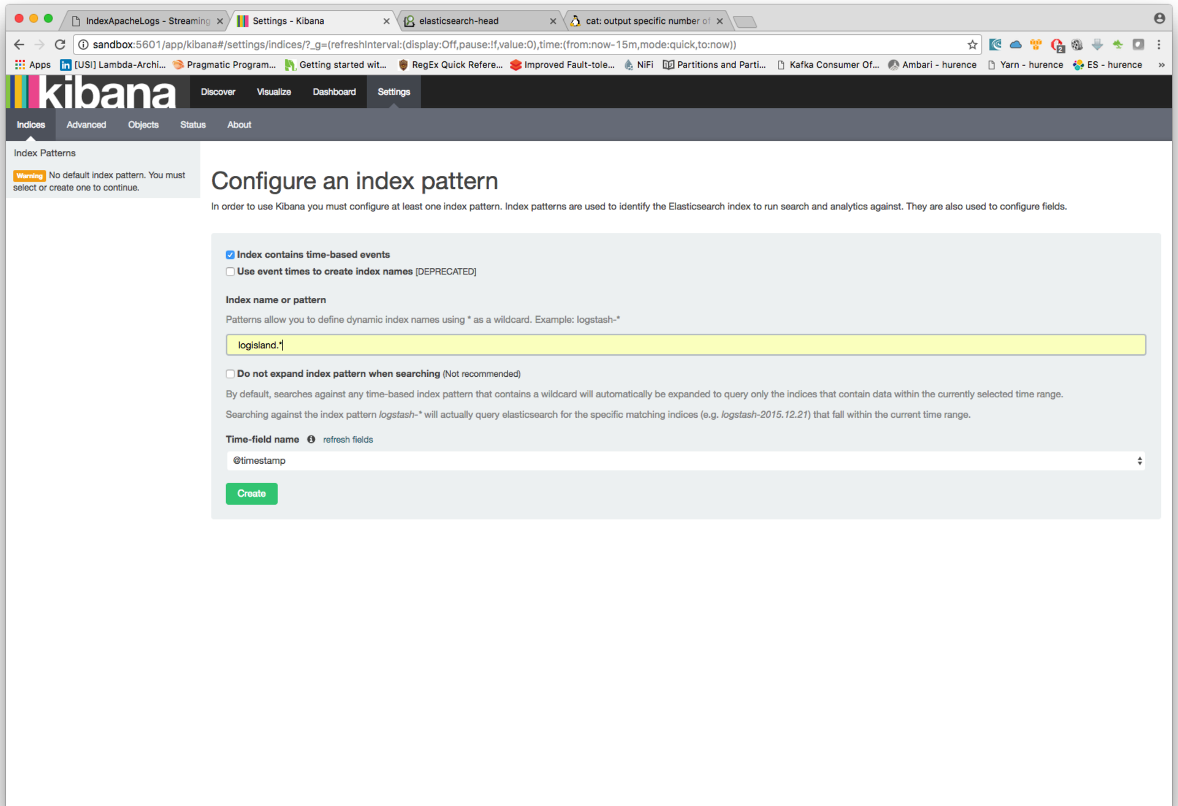
Note: Be sure to have added your server uri in conf/opc-iiot.yml file.

4. Inspect the records

With ElasticSearch, you can use Kibana.

Open up your browser and go to http://localhost:5601/ and you should be able to explore your apache logs.

Configure a new index pattern with `logisland.*` as the pattern name and `@timestamp` as the time value field.



Then if you go to Explore panel for the latest 15' time window you'll only see logisland process_metrics events which give you insights about the processing bandwidth of your streams.

1.8.21 Integrate Kafka Connect Sources & Sinks

In the following getting started tutorial, we'll focus on how to seamlessly integrate Kafka connect sources and sinks in logisland.

We can call this functionality *Logisland connect*.

Note: Be sure to know of to launch a logisland Docker environment by reading the prerequisites section

1. Logisland job setup

For this tutorial please make sure to already have installed elasticsearch and excel modules.

If not you can just do it through the components.sh command line:

```
bin/components.sh -i com.hurence.logisland:logisland-processor-elasticsearch:1.1.1

bin/components.sh -i com.hurence.logisland:logisland-service-elasticsearch_5_4_0-
↪client:1.1.1
```

(continues on next page)

(continued from previous page)

```
bin/components.sh -i com.github.jcustenborder.kafka.connect:kafka-connect-simulator:0.
↪1.118
```

The logisland job for this tutorial is already packaged in the tar.gz assembly and you can find it here for ElasticSearch :

```
docker exec -i -t logisland vim conf/logisland-kafka-connect.yml
```

We will start by explaining each part of the config file.

The engine

The first section configures the Spark engine (we will use a KafkaStreamProcessingEngine) to run in local mode.

```
engine:
  component: com.hurence.logisland.engine.spark.KafkaStreamProcessingEngine
  type: engine
  documentation: Use Kafka connectors with logisland
  configuration:
    spark.app.name: LogislandConnect
    spark.master: local[2]
    spark.driver.memory: 1G
    spark.driver.cores: 1
    spark.executor.memory: 2G
    spark.executor.instances: 4
    spark.executor.cores: 2
    spark.yarn.queue: default
    spark.yarn.maxAppAttempts: 4
    spark.yarn.am.attemptFailuresValidityInterval: 1h
    spark.yarn.max.executor.failures: 20
    spark.yarn.executor.failuresValidityInterval: 1h
    spark.task.maxFailures: 8
    spark.serializer: org.apache.spark.serializer.KryoSerializer
    spark.streaming.batchDuration: 1000
    spark.streaming.backpressure.enabled: false
    spark.streaming.unpersist: false
    spark.streaming.blockInterval: 500
    spark.streaming.kafka.maxRatePerPartition: 3000
    spark.streaming.timeout: -1
    spark.streaming.unpersist: false
    spark.streaming.kafka.maxRetries: 3
    spark.streaming.ui.retainedBatches: 200
    spark.streaming.receiver.writeAheadLog.enable: false
    spark.ui.port: 4050
```

The *controllerServiceConfigurations* part is here to define all services that be shared by processors within the whole job.

The parsing stream

Here we are going to use a special processor (`KafkaConnectStructuredSourceProviderService`) to use the kafka connect source as input for the structured stream defined below.

For this example, we are going to use the source *com.github.jcustenborder.kafka.connect.simulator.SimulatorSourceConnector* that generates records containing fake personal data at rate of 100 messages/s.

```
# Our source service
- controllerService: kc_source_service
  component: com.hurence.logisland.stream.spark.provider.
↪KafkaConnectStructuredSourceProviderService
  documentation: A kafka source connector provider reading from its own source and␣
↪providing structured streaming to the underlying layer
  configuration:
    # We will use the logisland record converter for both key and value
    kc.data.value.converter: com.hurence.logisland.connect.converter.
↪LogIslandRecordConverter
    # Use kryo to serialize the inner data
    kc.data.value.converter.properties: |
      record.serializer=com.hurence.logisland.serializer.KryoSerializer

    kc.data.key.converter: com.hurence.logisland.connect.converter.
↪LogIslandRecordConverter
    # Use kryo to serialize the inner data
    kc.data.key.converter.properties: |
      record.serializer=com.hurence.logisland.serializer.KryoSerializer
    # Only one task to handle source input (unique)
    kc.worker.tasks.max: 1
    # The kafka source connector to wrap (here we're using a simulator source)
    kc.connector.class: com.github.jcustenborder.kafka.connect.simulator.
↪SimulatorSourceConnector
    # The properties for the connector (as per connector documentation)
    kc.connector.properties: |
      key.schema.fields=email
      topic=simulator
      value.schema.fields=email,firstName,middleName,lastName,telephoneNumber,
↪dateOfBirth
    # We are using a standalone source for testing. We can store processed offsets in␣
↪memory
    kc.connector.offset.backing.store: memory
```

Note: The parameter **kc.connector.properties** contains the connector properties as you would have defined if you were using vanilla kafka connect.

As well, we are using a *memory* offset backing store. In a distributed scenario, you may have chosen a *kafka* topic based one.

Since each stream can be read and written, we are going to define as well a Kafka topic sink (`KafkaStructuredStreamProviderService`) that will be used as output for the structured stream defined below.

```
# Kafka sink configuration
- controllerService: kafka_out_service
  component: com.hurence.logisland.stream.spark.structured.provider.
↪KafkaStructuredStreamProviderService
  configuration:
    kafka.output.topics: logisland_raw
    kafka.error.topics: logisland_errors
    kafka.input.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    kafka.output.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
```

(continues on next page)

(continued from previous page)

```
    kafka.error.topics.serializer: com.hurence.logisland.serializer.JsonSerializer
    kafka.metadata.broker.list: sandbox:9092
    kafka.zookeeper.quorum: sandbox:2181
    kafka.topic.autoCreate: true
    kafka.topic.default.partitions: 4
    kafka.topic.default.replicationFactor: 1
```

So that, we can now define the *parsing stream* using those source and sink

```
######### parsing stream ##############
- stream: parsing_stream_source
  component: com.hurence.logisland.stream.spark.structured.StructuredStream
  documentation: "Takes records from the kafka source and distributes related␣
↪partitions over a kafka topic. Records are then handed off to the indexing stream"
  configuration:
    read.topics: /a/in
    read.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    read.topics.key.serializer: com.hurence.logisland.serializer.KryoSerializer
    read.topics.client.service: kc_source_service
    write.topics: logisland_raw
    write.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    write.topics.key.serializer: com.hurence.logisland.serializer.KryoSerializer
    write.topics.client.service: kafka_out_service
```

Within this stream, a `FlatMap` processor takes out the value and key (required when using *StructuredStream* as source of records)

```
processorConfigurations:
 - processor: flatten
   component: com.hurence.logisland.processor.FlatMap
   type: processor
   documentation: "Takes out data from record_value"
   configuration:
     keep.root.record: false
     copy.root.record.fields: true
```

The indexing stream

Inside this engine, you will run a Kafka stream of processing, so we set up input/output topics and Kafka/Zookeeper hosts. Here the stream will read all the logs sent in `logisland_raw` topic and push the processing output into `logisland_events` topic.

Note: We want to specify an Avro output schema to validate our output records (and force their types accordingly). It's really for other streams to rely on a schema when processing records from a topic.

We can define some serializers to marshall all records from and to a topic.

```
- stream: parsing_stream_source
  component: com.hurence.logisland.stream.spark.structured.StructuredStream
  documentation: "Takes records from the kafka source and distributes related␣
↪partitions over a kafka topic. Records are then handed off to the indexing stream"
  configuration:
    read.topics: /a/in
```

(continues on next page)

(continued from previous page)

```
    read.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    read.topics.key.serializer: com.hurence.logisland.serializer.KryoSerializer
    read.topics.client.service: kc_source_service
    write.topics: logisland_raw
    write.topics.serializer: com.hurence.logisland.serializer.KryoSerializer
    write.topics.key.serializer: com.hurence.logisland.serializer.KryoSerializer
    write.topics.client.service: kafka_out_service
```

Within this stream, a `DebugStream` processor takes a log line as a String and computes a `Record` as a sequence of fields.

```
processorConfigurations:
 # We just print the received records (but you may do something more interesting!)
 - processor: stream_debugger
   component: com.hurence.logisland.processor.DebugStream
   type: processor
   documentation: debug records
   configuration:
     event.serializer: json
```

This stream will process log entries as soon as they will be queued into *logisland_raw* Kafka topics, each log will be printed in the console and pushed back to Kafka in the `logisland_events` topic.

2. Launch the script

Connect a shell to your logisland container to launch the following streaming jobs.

```
docker exec -i -t logisland bin/logisland.sh --conf conf/logisland-kafka-connect.yml
```

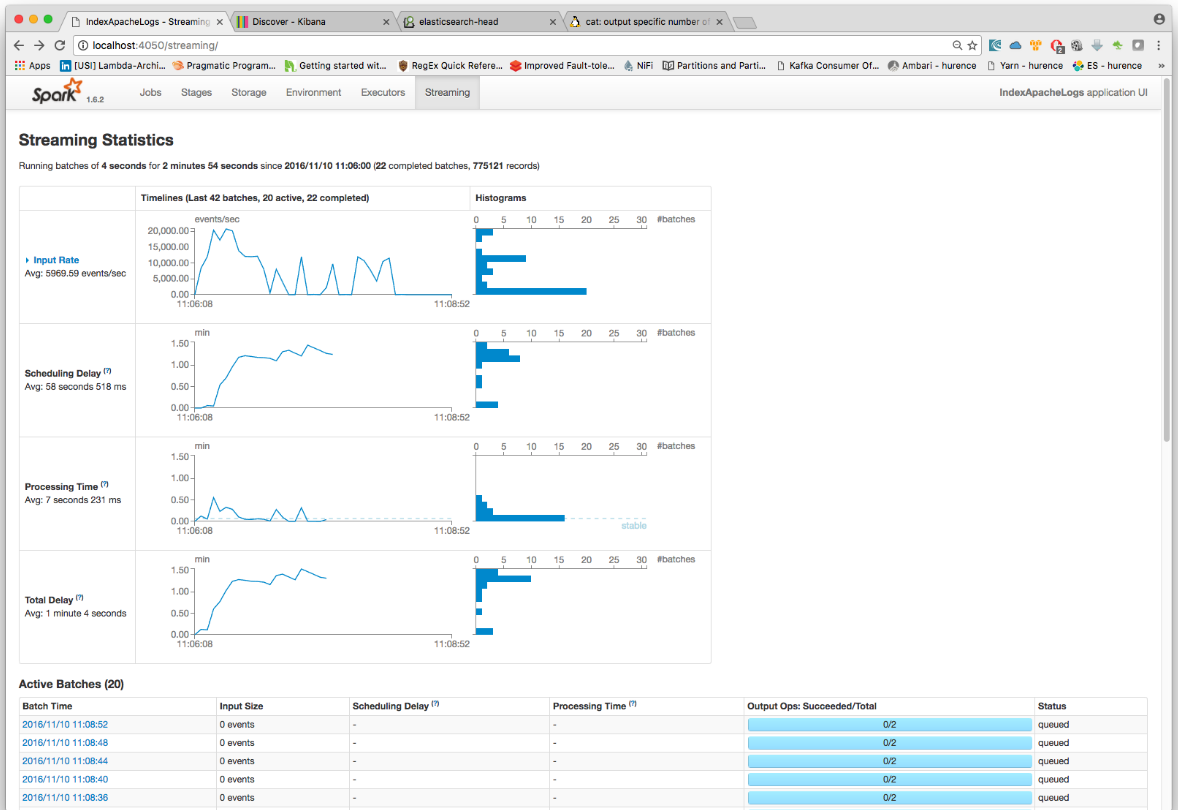
3. Examine your console output

Since we put a *DebugStream* processor, messages produced by our source connectors are then output to the console in json.

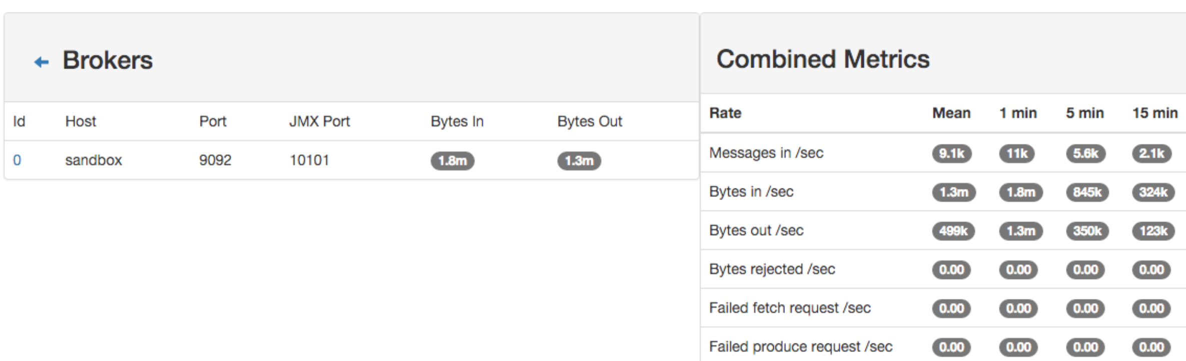
```
18/04/06 11:17:06 INFO DebugStream: {
 "id" : "9b17a9ac-97c4-44ef-9168-d298e8c53d42",
 "type" : "kafka_connect",
 "creationDate" : 1523006216376,
 "fields" : {
   "record_id" : "9b17a9ac-97c4-44ef-9168-d298e8c53d42",
   "firstName" : "London",
   "lastName" : "Marks",
   "telephoneNumber" : "005-694-4540",
   "record_key" : {
     "email" : "londonmarks@fake.com"
   },
   "middleName" : "Anna",
   "dateOfBirth" : 836179200000,
   "record_time" : 1523006216376,
   "record_type" : "kafka_connect",
   "email" : "londonmarks@fake.com"
 }
}
```

4. Monitor your spark jobs and Kafka topics

Now go to http://sandbox:4050/streaming/ to see how fast Spark can process your data



Another tool can help you to tweak and monitor your processing http://sandbox:9000/



1.8.22 Index JDBC messages

In the following getting started tutorial, we'll explain you how to read messages from a JDBC table.

The JDBC data will leverage the JDBC connector available as part of logisland connect.

Note: Be sure to know of to launch a logisland Docker environment by reading the prerequisites section

For kafka connect related information please follow as well the connectors section.

1.Install required components

For this tutorial please make sure to already have installed the kafka connect jdbc connector.

If not you can just do it through the componentes.sh command line:

```
bin/components.sh -r com.hurence.logisland.repackaged:kafka-connect-jdbc:5.0.0
```

2. Installing H2 datatabase

In this tutorial we'll use H2 Database.

H2 is a Java relational database

- Very fast database engine
- Open source
- Written in Java
- Supports standard SQL, JDBC API
- Embedded and Server mode, Clustering support
- Strong security features
- The PostgreSQL ODBC driver can be used
- Multi version concurrency

first wee need an sql engine. Let's use an **'H2 Java database<http://h2database.com/html/main.html>'_**. You can get the jar from their website and copy it to logisland lib folder inside Docker container. Then run the server on 9999 port

```
docker cp ./h2-1.4.197.jar logisland:/opt/logisland-1.1.1/lib
docker exec logisland java -jar lib/h2-1.4.197.jar   -webAllowOthers -tcpAllowOthers -
↪tcpPort 9999
```

You can manage your database through the web ui at http://sandbox:8082

With the URL JDBC parameter set to *jdbc:h2:tcp://sandbox:9999/~/test* you should be able to connect and create the following table

```
CREATE SCHEMA IF NOT EXISTS logisland;
USE logisland;

DROP TABLE IF EXISTS apache;

CREATE TABLE apache (record_id int auto_increment primary key, bytes_out integer,␣
↪http_method varchar(20), http_query varchar(200), http_status varchar(10), http_
↪version varchar(10), record_time timestamp, src_ip varchar(50), user varchar(20));
```

3. Logisland job setup

The interesting part in this tutorial is how to setup the JDBC stream.

Let's first focus on the stream configuration and then on its pipeline in order to extract the data in the right way.

Here we are going to use a special processor (`KafkaConnectStructuredSourceProviderService`) to use the kafka connect source as input for the structured stream defined below.

Logisland ships by default a kafka connect JDBC source implemented by the class *io.confluent.connect.jdbc.JdbcSourceConnector*.

You can find more information about how to configure a JDBC source in the official page of the JDBC Connector

Coming back to our example, we would like to read from a table called *logisland.apache* hosted in our local H2 database. The kafka connect controller service configuration will look like this:

```
- controllerService: kc_jdbc_source
  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:
    kc.data.key.converter: org.apache.kafka.connect.storage.StringConverter
    kc.worker.tasks.max: 1
    kc.partitions.max: 4
    kc.connector.class: io.confluent.connect.jdbc.JdbcSourceConnector
    kc.connector.offset.backing.store: memory
    kc.connector.properties: |
      connection.url=jdbc:h2:tcp://sandbox:9999/~/test
      connection.user=sa
      connection.password=
      mode=incrementing
      incrementing.column.name=RECORD_ID
      query=SELECT * FROM LOGISLAND.APACHE
      topic.prefix=test-jdbc-
```

Within this stream, a we need to extract the data coming from the JDBC.

First of all a `FlatMap` processor takes out the value and key (required when using *StructuredStream* as source of records)

```
processorConfigurations:
  - processor: flatten
    component: com.hurence.logisland.processor.FlatMap
    type: processor
    documentation: "Takes out data from record_value"
    configuration:
      keep.root.record: false
```

4. Launch the script

Now run the logisland job that will poll updates of new records inserted into *logisland.apache* table

```
docker exec logisland bin/logisland.sh --conf conf/index-jdbc-messages.yml
```

try to insert a few rows and have a look at the console output

```
INSERT into apache values (default, 46888, 'GET', '/shuttle/missions/sts-71/images/
↪KSC-95EC-0918.jpg', '200', 'HTTP/1.0', '2010-01-01 10:00:00' , 'net-1-141.eden.com',
↪ '-');
INSERT into apache values (default, 110,'GET','/cgi-bin/imagemap/countdown?99,176',
↪'302' ,'HTTP/1.0 ', '1995-07-01 04:01:06' ,'205.189.154.54', '-');
INSERT into apache values (default,12040,'GET','/shuttle/missions/sts-71/mission-sts-
↪71.html','200','HTTP/1.0', '1995-07-01 04:04:38','pme607.onramp.awinc.com', '-');
INSERT into apache values (default, 40310,'GET','/shuttle/countdown/count.gif','200' ,
↪'HTTP/1.0 ', '1995-07-01 04:05:18' ,'199.166.39.14', '-');
INSERT into apache values (default, 1.1.18,'GET','/images/dual-pad.gif','200' ,'HTTP/
↪1.0 ', '1995-07-01 04:04:10' ,'isdn6-34.dnai.com', '-');
INSERT into apache values (default,  9867,'GET','/software/winvn/winvn.html','200' ,
↪'HTTP/1.0 ', '1995-07-01 04:02:39' ,'dynip42.efn.org', '-');
INSERT into apache values (default, 1204,'GET','/images/KSC-logosmall.gif','200' ,
↪'HTTP/1.0 ', '1995-07-01 04:04:34' ,'netport-27.iu.net', '-');
```

it should be something like the following

```
...
18/09/04 12:47:33 INFO DebugStream: {
  "id" : "f7690b71-f339-4a84-8bd9-a0beb9ba5f92",
  "type" : "kafka_connect",
  "creationDate" : 1536065253831,
  "fields" : {
    "record_id" : "f7690b71-f339-4a84-8bd9-a0beb9ba5f92",
    "RECORD_TIME" : 0,
    "HTTP_STATUS" : "200",
    "SRC_IP" : "netport-27.iu.net",
    "RECORD_ID" : 7,
    "HTTP_QUERY" : "/images/KSC-logosmall.gif",
    "HTTP_VERSION" : "HTTP/1.0 ",
    "USER" : "-",
    "record_time" : 1536065253831,
    "record_type" : "kafka_connect",
    "HTTP_METHOD" : "GET",
    "BYTES_OUT" : 1204
  }
}
```

1.9 API design

logisland is a framework that you can extend through its API, you can use it to build your own `Processors` or to build data processing apps over it.

1.9.1 Java API

You can extend logisland with the Java low-level API as described below.

The primary material : Records

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`.

You can instanciate a Record like in the following code snipet:

```
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);
assertEquals(record.getType(), record.getField(FieldDictionary.RECORD_TYPE).
↪asString());
assertEquals(record.getType(), record.getField(FieldDictionary.RECORD_TYPE).
↪getRawValue());
```

And the other 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"));

assertFalse(record.hasField("unkown_field"));
assertTrue(record.hasField("method"));
assertEquals(record.getField("method").asString(), "GET");
assertTrue(record.getField("response_size").asInteger() - 452 == 0);
assertTrue(record.getField("is_outside_office_hours").asBoolean());
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.INT, 45L);
assertFalse(record.isValid());
record.setField("request_size", FieldType.LONG, 45L);
assertTrue(record.isValid());
record.setField("request_size", FieldType.DOUBLE, 45.5d);
assertTrue(record.isValid());
record.setField("request_size", FieldType.DOUBLE, 45.5);
assertTrue(record.isValid());
record.setField("request_size", FieldType.DOUBLE, 45L);
```

(continues on next page)

(continued from previous page)

```
assertFalse(record.isValid());
record.setField("request_size", FieldType.FLOAT, 45.5f);
assertTrue(record.isValid());
record.setField("request_size", FieldType.STRING, 45L);
assertFalse(record.isValid());
record.setField("request_size", FieldType.FLOAT, 45.5d);
assertFalse(record.isValid());
```

The tools to handle processing : Processor

logisland is designed as a component centric framework, so there's a layer of abstraction to build configurable components. Basically a component can be Configurable and Configured.

The most common component you'll use is the `Processor`

Let's explain the code of a basic `MockProcessor`, that doesn't acheive a really useful work but which is really self-explanatory we first need to extend `AbstractProcessor` class (or to implement `Processor` interface).

```
public class MockProcessor extends AbstractProcessor {

    private static Logger logger = LoggerFactory.getLogger(MockProcessor.class);
    private static String EVENT_TYPE_NAME = "mock";
```

Then we have to define a list of supported `PropertyDescriptor`. All theses properties and validation stuff are handled by `Configurable` interface.

```
public static final PropertyDescriptor FAKE_MESSAGE
    = new PropertyDescriptor.Builder()
        .name("fake.message")
        .description("a fake message")
        .required(true)
        .addValidator(StandardPropertyValidators.NON_EMPTY_VALIDATOR)
        .defaultValue("yoyo")
        .build();

@Override
public final List<PropertyDescriptor> getSupportedPropertyDescriptors() {
    final List<PropertyDescriptor> descriptors = new ArrayList<>();
    descriptors.add(FAKE_MESSAGE);

    return Collections.unmodifiableList(descriptors);
}
```

then comes the initialization bloc of the component given a `ComponentContext` (more on this later)

```
@Override
public void init(final ProcessContext context) {
    logger.info("init MockProcessor");
}
```

And now the real business part with the `process` method which handles all the work on the record's collection.

```
@Override
public Collection<Record> process(final ProcessContext context, final Collection
↪<Record> collection) {
```

(continues on next page)

(continued from previous page)

```
    final String message = context.getPropertyValue(FAKE_MESSAGE).asString();
    final List<Record> outputRecords = new ArrayList<>(collection);
    outputRecords.forEach(record -> record.setStringField("message", message));

    return outputRecords;
}
```

The Processor can then be configured through yaml config files

```
- processor: mock_processor
  component: com.hurence.logisland.util.runner.MockProcessor
  type: parser
  documentation: a parser that produce events for nothing
  configuration:
     fake.message: the super message
```

Transverse service injection : ControllerService

we often need to 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. We need to provide an interface API for this service :

```
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);
}
```

And an implementation of the cache contract :

```
public class LRUKeyValueCacheService<K,V>  extends AbstractControllerService␣
↪implements CacheService<K,V> {

    private volatile Cache<K,V> cache;

    @Override
    public V get(K k) {
        return cache.get(k);
    }

    @Override
    public void set(K k, V v) {
        cache.set(k, v);
    }
```

(continues on next page)

(continued from previous page)

```
    @Override
    @OnEnabled
    public void init(ControllerServiceInitializationContext context) throws␣
↪InitializationException {
        try {
            this.cache = createCache(context);
        }catch (Exception e){
            throw new InitializationException(e);
        }
    }

    @Override
    public List<PropertyDescriptor> getSupportedPropertyDescriptors() {
        List<PropertyDescriptor> props = new ArrayList<>();
        props.add(CACHE_SIZE);
        return Collections.unmodifiableList(props);
    }

    protected Cache<K,V> createCache(final ControllerServiceInitializationContext␣
↪context) throws IOException, InterruptedException {
        final int capacity = context.getPropertyValue(CACHE_SIZE).asInteger();
        return new LRUCache<K,V>(capacity);
    }
}
```

You can then use this service in a custom processor :

```
public class TestProcessor extends AbstractProcessor {

    static final PropertyDescriptor CACHE_SERVICE = new PropertyDescriptor.Builder()
            .name("cache.service")
            .description("CacheService")
            .identifiesControllerService(CacheService.class)
            .required(true)
            .build();

    @Override
    public boolean hasControllerService() {
        return true;
    }

    @Override
    public List<PropertyDescriptor> getSupportedPropertyDescriptors() {
        List<PropertyDescriptor> propDescs = new ArrayList<>();
        propDescs.add(CACHE_SERVICE);
        return propDescs;
    }

    @Override
    public Collection<Record> process(ProcessContext context, Collection<Record>␣
↪records) {
        return Collections.emptyList();
    }
}
```

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
    type: service
    documentation: cache service
    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
        type: parser
        documentation: a parser that produce events for nothing
        configuration:
           cache.service: lru_cache
```

Chaining processors in a stream : RecordStream

Warning: @todo

Running the processor's flow : Engine

Warning: @todo

Testing your processors : TestRunner

When you have coded your processor, pretty sure you want to test it with unit test. The framework provides you with the `TestRunner` tool for that. All you need is to instantiate a Testrunner with your Processor and its properties.

```
final String APACHE_LOG_SCHEMA = "/schemas/apache_log.avsc";
final String APACHE_LOG = "/data/localhost_access.log";
final String APACHE_LOG_FIELDS =
    "src_ip,identd,user,record_time,http_method,http_query,http_version,http_status,
↪bytes_out";
final String APACHE_LOG_REGEX =
    "(\\S+)\\s+(\\S+)\\s+(\\S+)\\s+\\[([\\w:/]+\\s[+\\-]\\d{4})\\]\\s+\
↪"(\\S+)\\s+(\\S+)\\s+(\\S+)\"\\s+(\\S+)\\s+(\\S+)";

final TestRunner testRunner = TestRunners.newTestRunner(new SplitText());
testRunner.setProperty(SplitText.VALUE_REGEX, APACHE_LOG_REGEX);
testRunner.setProperty(SplitText.VALUE_FIELDS, APACHE_LOG_FIELDS);
```

(continues on next page)

(continued from previous page)

```
// check if config is valid
testRunner.assertValid();
```

Now enqueue some messages as if they were sent to input Kafka topics

```
testRunner.clearQueues();
testRunner.enqueue(SplitTextTest.class.getResourceAsStream(APACHE_LOG));
```

Now run the process method and check that every `Record` has been correctly processed.

```
testRunner.run();
testRunner.assertAllInputRecordsProcessed();
testRunner.assertOutputRecordsCount(200);
testRunner.assertOutputErrorCount(0);
```

You can validate that all output records are validated against an avro schema

```
final RecordValidator avroValidator = new AvroRecordValidator(SplitTextTest.class.
→getResourceAsStream
testRunner.assertAllRecords(avroValidator);
```

And check if your output records behave as expected.

```
MockRecord out = testRunner.getOutputRecords().get(0);
out.assertFieldExists("src_ip");
out.assertFieldNotExists("src_ip2");
out.assertFieldEquals("src_ip", "10.3.10.134");
out.assertRecordSizeEquals(9);
out.assertFieldEquals(FieldDictionary.RECORD_TYPE, "apache_log");
out.assertFieldEquals(FieldDictionary.RECORD_TIME, 1469342728000L);
```

1.10 Logisland REST API

The Logisland REST API for third party applications.

maxdepth 3

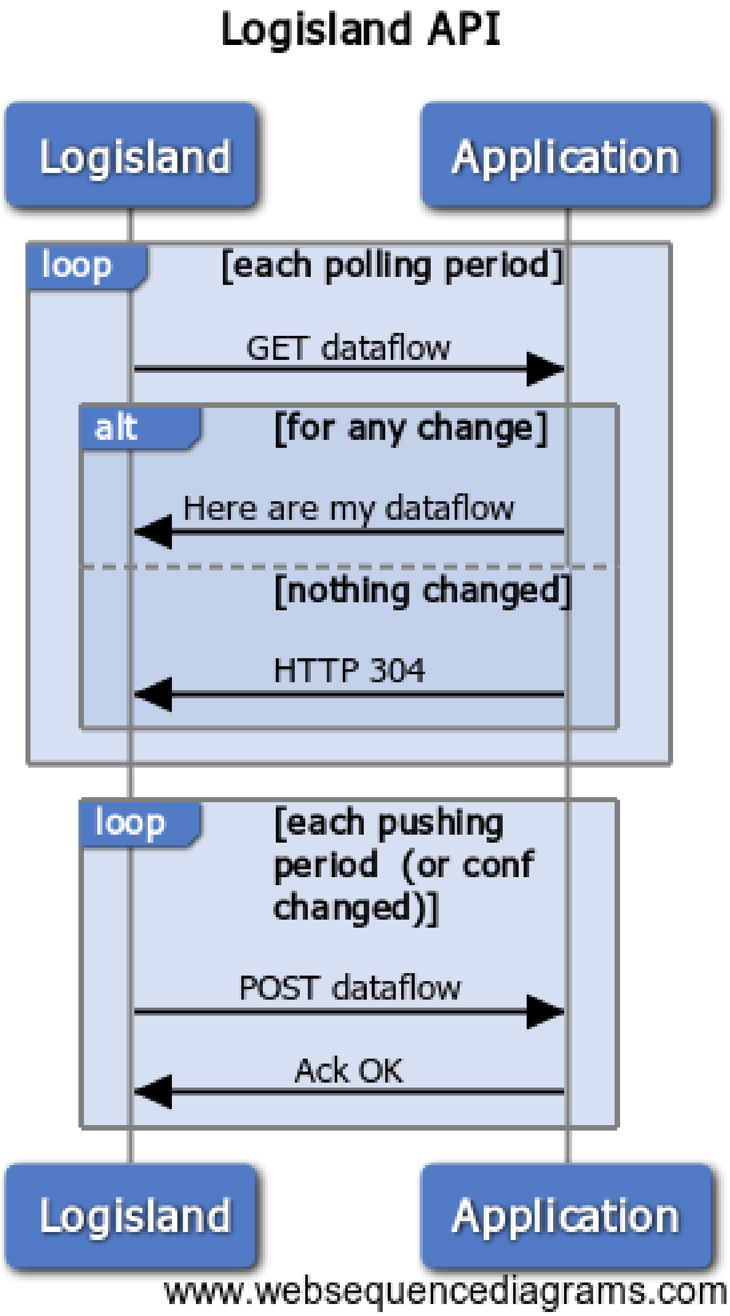
1.10.1 Introduction

Logisland makes available a standard RESTful API definition to interoperate with any third party application implementing it.

The API should be implemented by a third party application and logisland will regularly poll this endpoint in order to:

- Ask for configuration changes to be triggered.
- Report the latest configuration applied (to ease up resynchronization and business continuity).

Both flows can hence be resumed by the following sequence diagram:



1.10.2 Usage

In terms of API, two degrees of freedom are possible:

- **Dataflow**:

  A dataflow is a set of services and streams allowing a data flowing from one or more sources, being transformed and reach one or more destinations (sinks).

  Act at dataflow level if you want to:

   - Add/Remove any streaming endpoint
   - Change any active stream configuration (e.g. kafka topic)
   - Create/Remote/Modify any service

- **Pipeline**:

  A pipeline is a processing chain acting on a data flowing point-to-point.

  The api gives you the possibility to have a finer-grained control of what is going of any stream pipeline without perturbing the stream itself. This means that the processor chain will be dynamically reconfigured without the need of stopping the stream and reconfigure the whole dataflow.

  Act at pipeline level if you want to:

   - Add/Remove processors in the pipeline
   - Change any processor configuration

Hint: As a general rule, the changes will be triggered if the *lastUpdated* field of the object you are going to modify is fresher than the one known by logisland.

1.10.3 API Specification

This section resumes the Rest API specification. More details are available on the swagger spec.

Operations

GET `/dataflows/{dataflowName}`

Summary

Retrieves the configuration for a specified dataflow

Description

Logisland will call this endpoint to know which configuration should be run.

> This endpoint also supports HTTP caching (Last-Updated, If-Modified-Since) as per RFC 7232, section 3.3

Parameters

delim

header "Name", "Located in", "Required", "Type", "Format", "Properties", "Description" :widths: 20, 15, 10, 10, 10, 20, 30

dataflowName | path | Yes | string | | | the dataflow name (aka the logisland job name)

Request

Headers

```
If-Modified-Since: Timestamp of last response
```

Responses

200

Return the dataflow configuration. On logisland side, the following will happen: - At dataflow level:

- Fully reconfigure a dataflow (stop and then start) if nothing is running (initial state) or if lastUpdated is fresher than the one of the already running dataflow.

  In this case be aware that old stream and services will be destroyed and new ones will be created.

- Do nothing otherwise (keep running the active dataflow)
- At pipeline level:
  - The processor chain will be fully reconfigured if and only if the pipeline lastUpdated is fresher than the lastUpdated known by the system.

  In any case the stream is never stopped.

Type: *Versioned* extended inline

Example:


```
{
    "lastModified": "2015-01-01T15:00:00.000Z",
    "modificationReason": "somestring",
    "services": [
        {
            "component": "somestring",
            "config": [
                {
                    "key": "somestring",
                    "type": "string",
                    "value": "somestring"
                },
                {
                    "key": "somestring",
                    "type": "string",
                    "value": "somestring"
                }
            ],
            "documentation": "somestring",
            "name": "somestring"
        },
        {
            "component": "somestring",
            "config": [
                {
                    "key": "somestring",
                    "type": "string",
```


(continues on next page)

(continued from previous page)


```
                    "value": "somestring"
                },
                {
                    "key": "somestring",
                    "type": "string",
                    "value": "somestring"
                }
            ],
            "documentation": "somestring",
            "name": "somestring"
        }
    ],
    "streams": [
        {
            "component": "somestring",
            "config": [
                {
                    "key": "somestring",
                    "type": "string",
                    "value": "somestring"
                },
                {
                    "key": "somestring",
                    "type": "string",
                    "value": "somestring"
                }
            ],
            "documentation": "somestring",
            "name": "somestring",
            "pipeline": {
                "lastModified": "2015-01-01T15:00:00.000Z",
                "modificationReason": "somestring",
                "processors": [
                    {
                        "component": "somestring",
                        "config": [
                            {
                                "key": "somestring",
                                "type": "string",
                                "value": "somestring"
                            },
                            {
                                "key": "somestring",
                                "type": "string",
                                "value": "somestring"
                            }
                        ],
                        "documentation": "somestring",
                        "name": "somestring"
                    },
                    {
                        "component": "somestring",
                        "config": [
                            {
                                "key": "somestring",
                                "type": "string",
                                "value": "somestring"
```


(continues on next page)

(continued from previous page)


```
                        },
                        {
                            "key": "somestring",
                            "type": "string",
                            "value": "somestring"
                        }
                    ],
                    "documentation": "somestring",
                    "name": "somestring"
                }
            ]
        }
    },
    {
        "component": "somestring",
        "config": [
            {
                "key": "somestring",
                "type": "string",
                "value": "somestring"
            },
            {
                "key": "somestring",
                "type": "string",
                "value": "somestring"
            }
        ],
        "documentation": "somestring",
        "name": "somestring",
        "pipeline": {
            "lastModified": "2015-01-01T15:00:00.000Z",
            "modificationReason": "somestring",
            "processors": [
                {
                    "component": "somestring",
                    "config": [
                        {
                            "key": "somestring",
                            "type": "string",
                            "value": "somestring"
                        },
                        {
                            "key": "somestring",
                            "type": "string",
                            "value": "somestring"
                        }
                    ],
                    "documentation": "somestring",
                    "name": "somestring"
                },
                {
                    "component": "somestring",
                    "config": [
                        {
                            "key": "somestring",
                            "type": "string",
                            "value": "somestring"
```


(continues on next page)

(continued from previous page)

```
                            },
                            {
                                "key": "somestring",
                                "type": "string",
                                "value": "somestring"
                            }
                        ],
                        "documentation": "somestring",
                        "name": "somestring"
                    }
                ]
            }
        }
    ]
}
```

304

Nothing has been modified since the last call.

In this case the body content will be completely ignored (hence the server can answer with an empty body to save network and resources).

404

Not found (the server probably does not handle this dataflow)

default

Unexpected error

POST `/dataflows/{dataflowName}`

Summary

Push the configuration of running dataflows.

Description

The endpoint will be called: - On a regular basis (according to logisland configuration). - Each time the a dataflow or a pipeline configuration change has been applied.

This service can be seen as well as a liveness ping.

Parameters

delim

header “Name”, “Located in”, “Required”, “Type”, “Format”, “Properties”, “Description” :widths: 20, 15, 10, 10, 10, 20, 30

jobId | path | Yes | string | | | logisland job id (aka the engine name) dataflowName | path | Yes | string | | | the dataflow name (aka the logisland job name)

Request

Body

A streaming pipeline.

Versioned extended inline

Inline schema:

delim

header “Name”, “Required”, “Type”, “Format”, “Properties”, “Description” :widths: 20, 10, 15, 15, 30, 25

lastModified | Yes | string | date-time | | the last modified timestamp of this pipeline (used to trigger changes). modificationReason | No | string | | | Can be used to document latest changeset. services | No | array of *Component* | | | The service controllers.

streams | No | array of *Component* extended *inline* | | | The engine properties.

```
{
    "lastModified": "2015-01-01T15:00:00.000Z",
    "modificationReason": "somestring",
    "services": [
        {
            "component": "somestring",
            "config": [
                {
                    "key": "somestring",
                    "type": "string",
                    "value": "somestring"
                },
                {
                    "key": "somestring",
                    "type": "string",
                    "value": "somestring"
                }
            ],
            "documentation": "somestring",
            "name": "somestring"
        },
        {
            "component": "somestring",
            "config": [
                {
                    "key": "somestring",
                    "type": "string",
                    "value": "somestring"
```

(continues on next page)

(continued from previous page)

```
                },
                {
                    "key": "somestring",
                    "type": "string",
                    "value": "somestring"
                }
            ],
            "documentation": "somestring",
            "name": "somestring"
        }
    ],
    "streams": [
        {
            "component": "somestring",
            "config": [
                {
                    "key": "somestring",
                    "type": "string",
                    "value": "somestring"
                },
                {
                    "key": "somestring",
                    "type": "string",
                    "value": "somestring"
                }
            ],
            "documentation": "somestring",
            "name": "somestring",
            "pipeline": {
                "lastModified": "2015-01-01T15:00:00.000Z",
                "modificationReason": "somestring",
                "processors": [
                    {
                        "component": "somestring",
                        "config": [
                            {
                                "key": "somestring",
                                "type": "string",
                                "value": "somestring"
                            },
                            {
                                "key": "somestring",
                                "type": "string",
                                "value": "somestring"
                            }
                        ],
                        "documentation": "somestring",
                        "name": "somestring"
                    },
                    {
                        "component": "somestring",
                        "config": [
                            {
                                "key": "somestring",
                                "type": "string",
                                "value": "somestring"
                            },
```

(continues on next page)

(continued from previous page)

```
                        {
                            "key": "somestring",
                            "type": "string",
                            "value": "somestring"
                        }
                    ],
                    "documentation": "somestring",
                    "name": "somestring"
                }
            ]
        }
    },
    {
        "component": "somestring",
        "config": [
            {
                "key": "somestring",
                "type": "string",
                "value": "somestring"
            },
            {
                "key": "somestring",
                "type": "string",
                "value": "somestring"
            }
        ],
        "documentation": "somestring",
        "name": "somestring",
        "pipeline": {
            "lastModified": "2015-01-01T15:00:00.000Z",
            "modificationReason": "somestring",
            "processors": [
                {
                    "component": "somestring",
                    "config": [
                        {
                            "key": "somestring",
                            "type": "string",
                            "value": "somestring"
                        },
                        {
                            "key": "somestring",
                            "type": "string",
                            "value": "somestring"
                        }
                    ],
                    "documentation": "somestring",
                    "name": "somestring"
                },
                {
                    "component": "somestring",
                    "config": [
                        {
                            "key": "somestring",
                            "type": "string",
                            "value": "somestring"
                        },
```

(continues on next page)

(continued from previous page)

```
                                {
                                    "key": "somestring",
                                    "type": "string",
                                    "value": "somestring"
                                }
                            ],
                            "documentation": "somestring",
                            "name": "somestring"
                        }
                    ]
                }
            }
        ]
}
```

Responses

default

The server should return HTTP 200 OK. By the way, the response is ignored by Logisland since the operation has a *fire and forget* nature.

Data Structures

Component Model Structure

delim

header “Name”, “Required”, “Type”, “Format”, “Properties”, “Description” :widths: 20, 10, 15, 15, 30, 25

component | Yes | string | | | config | No | array of *Property* | | |

documentation | No | string | | | name | Yes | string | | |

DataFlow Model Structure

A streaming pipeline.

Versioned extended inline

Inline schema:

delim

header “Name”, “Required”, “Type”, “Format”, “Properties”, “Description” :widths: 20, 10, 15, 15, 30, 25

lastModified | Yes | string | date-time | | the last modified timestamp of this pipeline (used to trigger changes). modificationReason | No | string | | | Can be used to document latest changeset. services | No | array of *Component* | | | The service controllers.

streams | No | array of *Component* extended *inline* | | | The engine properties.

Pipeline Model Structure

Tracks stream processing pipeline configuration

Versioned extended *inline*

Inline schema:

delim

header "Name", "Required", "Type", "Format", "Properties", "Description" :widths: 20, 10, 15, 15, 30, 25

lastModified | Yes | string | date-time | | the last modified timestamp of this pipeline (used to trigger changes). modificationReason | No | string | | | Can be used to document latest changeset. processors | No | array of *Component* | | |

Processor Model Structure

A logisland 'processor'.

Component

Property Model Structure

delim

header "Name", "Required", "Type", "Format", "Properties", "Description" :widths: 20, 10, 15, 15, 30, 25

key | Yes | string | | | type | No | string | | { 'default': 'string' } | value | Yes | string | | |

Service Model Structure

A logisland 'controller service'.

Component

Stream Model Structure

Component extended *inline*

Inline schema:

delim

header "Name", "Required", "Type", "Format", "Properties", "Description" :widths: 20, 10, 15, 15, 30, 25

component | Yes | string | | | config | No | array of *Property* | | |

documentation | No | string | | | name | Yes | string | | | pipeline | No | *Versioned* extended *inline* | | |

Versioned Model Structure

a versioned component

delim

header "Name", "Required", "Type", "Format", "Properties", "Description" :widths: 20, 10, 15, 15, 30, 25

lastModified | Yes | string | date-time | | the last modified timestamp of this pipeline (used to trigger changes). modificationReason | No | string | | | Can be used to document latest changeset.

1.11 What's new in logisland ?

1.11.1 v1.1.1

- add a clock service
- improve monitoring
- improve Cassandra support

1.11.2 v1.0.0

- add support for JMS kafka connect source
- add support for JDBC kafka connect source
- add Cassandra datastore service
- support all Kafka connect sinks
- add KafkaStreams engine
- update documentation
- fix test framework (runner)
- added vanilla java engine

1.11.3 v0.14.0

- add support for SOLR
- add support for Chronix timeseries
- review Datastore API
- fix matchquery update field policy issue
- remove elasticsearch 2.3 support

1.11.4 v0.10.0

- add kibana pcap panel cyber-security feature gui #187
- add support for elasticsearch 2.4 feature processor
- add support for elasticsearch 5 feature processor #214
- fix pb in kafkaStreamProcessingEngine (2.1) #244
- allow to set a default profile during build #271
- add ElasticSearch Service feature framework #241
- add multiGet elastic search processor feature processor #255
- fix Pcap telemetry processor issue #180 #224
- Make build work if no profile specified (use the highest hdp one) build #210
- implement Logisland agent #201
- fix travis build randomly fails on travis CI (spark-engine module tests) bug framework #159
- support maven profiles to handle dépendencies (hdp 2.4 & hdp 2.5) #116
- add a RESTful API for components live update agent feature framework #42
- add a logisland agent agent enhancement feature framework #117
- add a Topic metadata view feature gui #101
- add scheduler view feature framework gui #103
- add job configuration view feature gui #94
- add a global logisland.properties agent feature #71
- add a Topic metadata registry feature framework
- integrate BRO files & notification through a BroProcessor feature processor security #93
- add Support for SMTP/Mailer Processor feature processor security #138
- add a Release/deployment documentation #108
- Ensure source files have a licence header
- add HBase service to get and scan records
- add Multiget elasticsearch enricher processor
- add sessionization processor
- improve topic management in web ui gui #222

- Docker images shall be builded automatically framework #200
- fix classpath issue bug framework #247
- add Netflow telemetry Processor cyber-security feature processor #181
- add an "How to contribute page" documentation #183
- fix PutElasticsearch throws UnsupportedOperationException when duplicate document is found bug processor #221
- Feature/maven docker#200 enhancement framework #242
- Feature/partitioner enhancement framework #238
- add PCAP telemetry Processor cyber-security feature processor #180
- Move Mailer Processor into commons plugins build #196
- Origin/webanalytics framework processor web-analytics #236
- rename Plugins to Processors in online documentation documentation #173

1.11.5 v0.9.8

- add a retry parameter to PutElasticsearch bug enhancement processor #124
- add Timezone managmt to SplitText enhancement processor #126
- add IdempotentId processor enhancement feature processor #127
- migrate to Kafka 0.9 enhancement

1.11.6 v0.9.7

- add HDFS burner feature processor #89
- add ExtractJsonPath processor #90
- check compatibility with HDP 2.5 #112
- sometimes the drivers fails with status SUCCEEDED which prevents YARN to resubmit the job automatically #105
- logisland crashes when starting with wrong offsets #111
- add type checking for SplitText component enhancement #46
- add optional regex to SplitText #106
- add record schema management with ConvertFieldsType processor #75
- add field auto extractor processor : SplitTextWithProperties #49
- add a new RemoveFields processor
- add a NormalizeFields processor #88
- Add notion of asserting the asserted fields in MockRecord

1.11.7 v0.9.6

- add a Documentation generator for plugins feature #69
- add SQL aggregator plugin feature #74
- #66 merge elasticsearch-shaded and elasticsearch-plugin enhancement
- #73 add metric aggregator processor feature
- #57 add sampling processor enhancement
- #72 integrate OutlierDetection plugin feature
- #34 integrate QueryMatcherProcessor bug

1.11.8 v0.9.5

- generify API from Event to Records
- add docker container for demo
- add topic auto-creation parameters
- add Record validators
- add processor chaining that works globally on an input/output topic and pipe in-memory contexts into sub-processors
- better error handling for SplitText
- testRunner API
- migrate LogParser to LogProcessor Interface
- reporting metrics to know where are exactly the processors on the topics
- add an HDFSBurner Engine
- yarn stability improvements
- more spark parameters handling
- driver failover through Zookeper offset checkpointing
- add raw_content to event if regex matching failed in SplitText
- integration testing with embedded Kafka/Spark
- processor chaining
-

1.12 Frequently Asked Questions.

1.12.1 I already use ELK, why would I need to use LogIsland ?

Well, at first one could say that that both stacks are overlapping, but the real purpose of the LogIsland framework is the abstraction of scalability of log aggregation.

In fact if you already have an ELK stack you'll likely want to make it scale (without pain) in both volume and features ways. LogIsland will be used for this purpose as an EOM (Event Oriented Middleware) based on Kafka & Spark, where you can plug advanced features with ease.

So you just have to route your logs from the Logstash (or Flume, or Collectd, . . .) agents to Kafka topics and launch parsers and processors.

1.12.2 Do I need Hadoop to play with LogIsland ?

No, if your goal is simply to aggregate a massive amount of logs in an Elasticsearch cluster, and to define complex event processing rules to generate new events you definitely don't need an Hadoop cluster.

Kafka topics can be used as an high throughput log buffer for sliding-windows event processing. But if you need advanced batch analytics, it's really easy to dump your logs into an hadoop cluster to build machine learning models.

1.12.3 How do I make it scale ?

LogIsland is made for scalability, it relies on Spark and Kafka which are both scalable by essence, to scale LogIsland just have to add more kafka brokers and more Spark slaves. This is the *manual* way, but we've planned in further releases to provide auto-scaling either Docker Swarn support or Mesos Marathon.

1.12.4 What's the difference between Apache NIFI and LogIsland ?

Apache NIFI is a powerful ETL very well suited to process incoming data such as logs file, process & enrich them and send them out to any datastore. You can do that as well with LogIsland but LogIsland is an event oriented framework designed to process huge amount of events in a Complex Event Processing manner not a Single Event Processing as NIFI does. **LogIsland** is not an ETL or a DataFlow, the main goal is to extract information from realtime data.

Anyway you can use Apache NIFI to process your logs and send them to Kafka in order to be processed by LogIsland

1.12.5 Error : realpath not found

If you don't have the `realpath` command on you system you may need to install it:

```
brew install coreutils
sudo apt-get install coreutils
```

1.12.6 How to deploy LogIsland as a Single node Docker container

The easy way : you start a small Docker container with all you need inside (Elasticsearch, Kibana, Kafka, Spark, LogIsland + some usefull tools)

Docker is becoming an unavoidable tool to isolate a complex service component. It's easy to manage, deploy and maintain. That's why you can start right away to play with LogIsland through the Docker image provided from Docker HUB

```
# Get the LogIsland image
docker pull hurence/logisland

# Run the container
docker run \
      -it \
      -p 80:80 \
      -p 9200-9300:9200-9300 \
      -p 5601:5601 \
```

(continues on next page)

(continued from previous page)

```
    -p 2181:2181 \
    -p 9092:9092 \
    -p 9000:9000 \
    -p 4050-4060:4050-4060 \
    --name logisland \
    -h sandbox \
    hurence/logisland:latest bash

# Connect a shell to your LogIsland container
docker exec -ti logisland bash
```

1.12.7 How to deploy LogIsland in an Hadoop cluster ?

When it comes to scale, you'll need a cluster. **logisland** is just a framework that facilitates running sparks jobs over Kafka topics so if you already have a cluster you just have to get the latest logisland binaries and unzip them to a edge node of your hadoop cluster.

For now Log-Island is fully compatible with HDP 2.4 but it should work well on any cluster running Kafka and Spark. Get the latest release and build the package.

You can download the latest release build

```
git clone git@github.com:Hurence/logisland.git
cd logisland-0.9.5
mvn clean install -DskipTests
```

This will produce a `logisland-assembly/target/logisland-0.9.5-bin.tar.gz` file that you can untar into any folder of your choice in a edge node of your cluster.

Please read this excellent article on spark long running job setup : http://mkuthan.github.io/blog/2016/09/30/spark-streaming-on-yarn/

1.12.8 How can I configure Kafka to avoid irrecoverable exceptions ?

If the message must be reliable published on Kafka cluster, Kafka producer and Kafka cluster needs to be configured with care. It needs to be done independently of chosen streaming framework.

Kafka producer buffers messages in memory before sending. When our memory buffer is exhausted, Kafka producer must either stop accepting new records (block) or throw errors. By default Kafka producer blocks and this behavior is legitimate for stream processing. The processing should be delayed if Kafka producer memory buffer is full and could not accept new messages. Ensure that block.on.buffer.full Kafka producer configuration property is set.

With default configuration, when Kafka broker (leader of the partition) receive the message, store the message in memory and immediately send acknowledgment to Kafka producer. To avoid data loss the message should be replicated to at least one replica (follower). Only when the follower acknowledges the leader, the leader acknowledges the producer.

This guarantee you will get with ack=all property in Kafka producer configuration. This guarantees that the record will not be lost as long as at least one in-sync replica remains alive.

But this is not enough. The minimum number of replicas in-sync must be defined. You should configure min.insync.replicas property for every topic. I recommend to configure at least 2 in-sync replicas (leader and one follower). If you have datacenter with two zones, I also recommend to keep leader in the first zone and 2 followers in the second zone. This configuration guarantees that every message will be stored in both zones.

We are almost done with Kafka cluster configuration. When you set min.insync.replicas=2 property, the topic should be replicated with factor 2 + N. Where N is the number of brokers which could fail, and Kafka producer will still be able to publish messages to the cluster. I recommend to configure replication factor 3 for the topic (or more).

With replication factor 3, the number of brokers in the cluster should be at least 3 + M. When one or more brokers are unavailable, you will get underreplicated partitions state of the topics. With more brokers in the cluster than replication factor, you can reassign underreplicated partitions and achieve fully replicated cluster again. I recommend to build the 4 nodes cluster at least for topics with replication factor 3.

The last important Kafka cluster configuration property is unclean.leader.election.enable. It should be disabled (by default it is enabled) to avoid unrecoverable exceptions from Kafka consumer. Consider the situation when the latest committed offset is N, but after leader failure, the latest offset on the new leader is M < N. M < N because the new leader was elected from the lagging follower (not in-sync replica). When the streaming engine ask for data from offset N using Kafka consumer, it will get an exception because the offset N does not exist yet. Someone will have to fix offsets manually.

So the minimal recommended Kafka setup for reliable message processing is:

```
4 nodes in the cluster
unclean.leader.election.enable=false in the brokers configuration
replication factor for the topics - 3
min.insync.replicas=2 property in topic configuration
ack=all property in the producer configuration
block.on.buffer.full=true property in the producer configuration
```

With the above setup your configuration should be resistant to single broker failure, and Kafka consumers will survive new leader election.

You could also take look at replica.lag.max.messages and replica.lag.time.max.ms properties for tuning when the follower is removed from ISR by the leader. But this is out of this blog post scope.

1.12.9 How to purge a Kafka queue ?

Temporarily update the retention time on the topic to one second:

```
kafka-topics.sh --zookeeper localhost:13003 --alter --topic MyTopic --config␣
↪retention.ms=1000
```

then wait for the purge to take effect (about one minute). Once purged, restore the previous retention.ms value.

You can also try to delete the topic :

add one line to server.properties file under config folder:

```
delete.topic.enable=true
```

then, you can run this command:

```
bin/kafka-topics.sh --zookeeper localhost:2181 --delete --topic test
```

CHAPTER 2

Indices and tables

- genindex
- modindex
- search