logisland Documentation

Release 0.10.0-rc1

bailet.thomas

Jun 06, 2017

Contents

Chat with us on Gitter Download the latest release build and unzip on an edge node.

CHAPTER 1

Contents:

Introduction

you can find a quick presentation below :

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

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.

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

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

Architecture

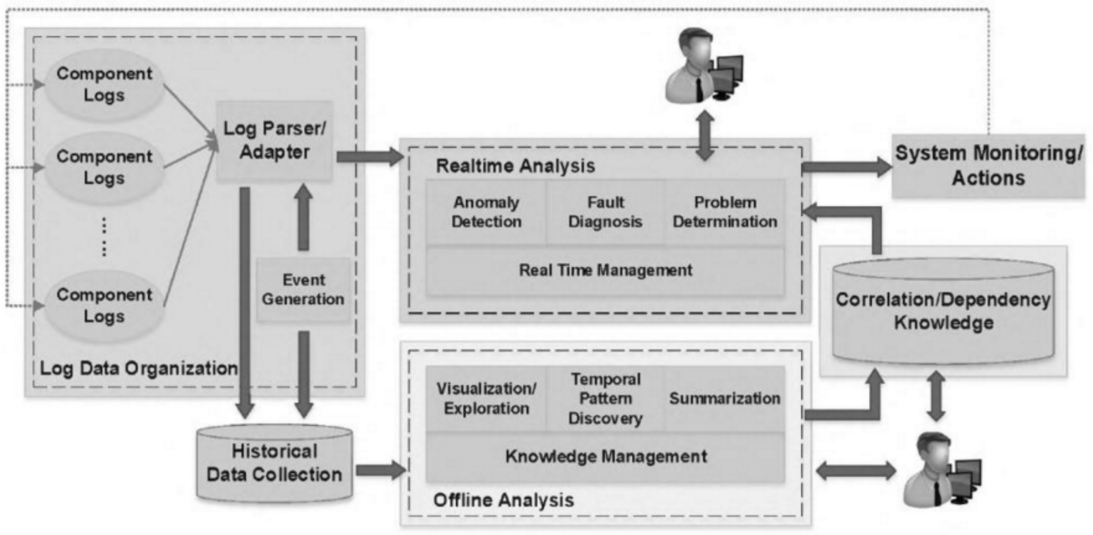
Is there something clever out there ?

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.

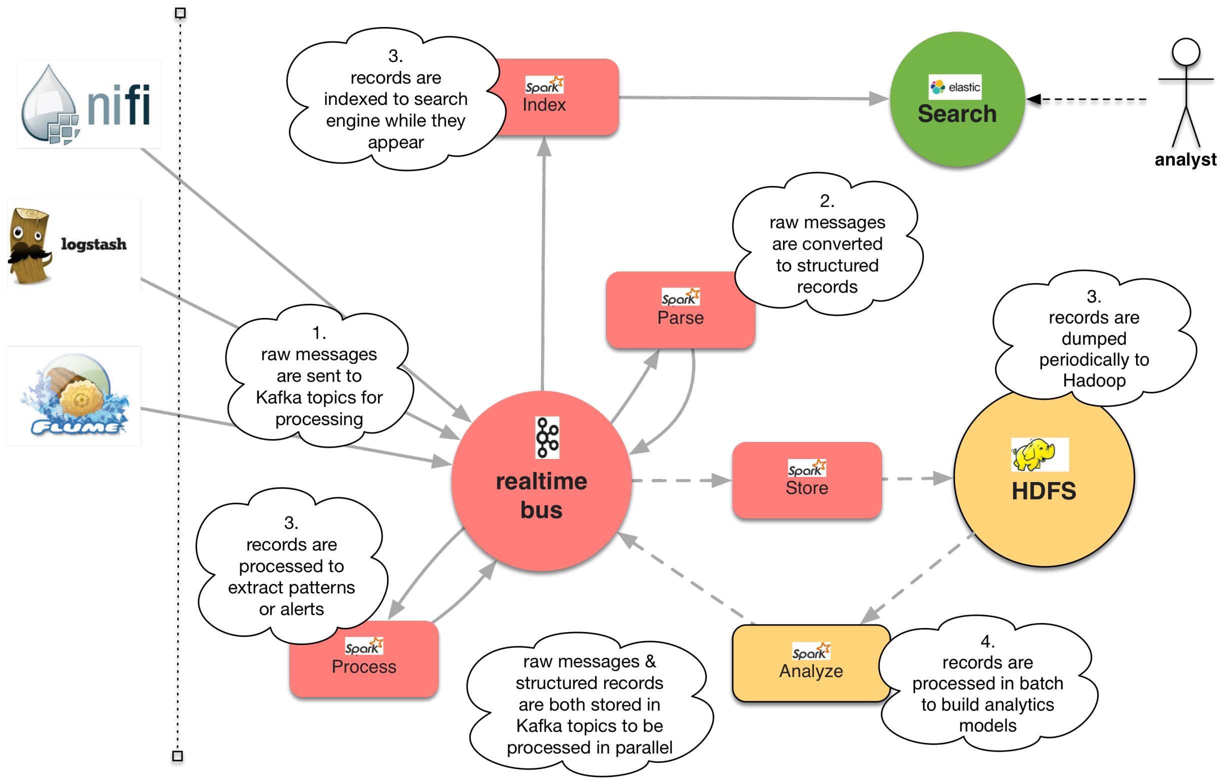
Data driven architecture



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.


nifi
logstash
FLUME
3. records are indexed to search engine while they appear
Spark Index
elastic Search
analyst
2. raw messages are converted to structured records
Spark Parse
3. records are dumped periodically to Hadoop
1. raw messages are sent to Kafka topics for processing
realtime bus
Spark Store
HDFS
3. records are processed to extract patterns or alerts
Spark Process
raw messages & structured records are both stored in Kafka topics to be processed in parallel
Spark Analyze
4. records are processed in batch to build analytics models

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.

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);

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.

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**.

Merge a pull request or patch

To pull in a merge request you should generally follow the command line instructions sent out by GitHub.

1. Go to your local copy of the [Apache git repo](https://github.com/hurence/logisland.git), switch to the *master* branch, and make sure it is up to date.

```
git checkout master
git fetch origin
git merge origin/master
```

2. Create a local branch for integrating and testing the pull request. You may want to name the branch according to the Logisland github issue ticket associated with the pull request (example: *LOGISLAND-1234*).

```
git checkout -b <local_test_branch>  # e.g. git checkout -b LOGISLAND-1234
```

3. Merge the pull request into your local test branch.

```
git pull <remote_repo_url> <remote_branch>
```

4. Assuming that the pull request merges without any conflicts: Update the top-level *changes.rst*, and add in the github issue ticket number (example: *LOGISLAND-1234*) and ticket description to the change log. Make sure that you place the github issue ticket number in the commit comments where applicable.

5. Run any sanity tests that you think are needed.

6. Once you are confident that everything is ok, you can merge your local test branch into your local *master* branch, and push the changes back to the hurence repo.

```
# Pull request looks ok, change log was updated, etc.  We are ready for␣
↪pushing.
git checkout master
git merge <local_test_branch>  # e.g. git merge LOGISLAND-1234

# At this point our local master branch is ready, so now we will push the␣
↪changes
# to the official repo.
git push origin  HEAD:refs/heads/master
```

7. The last step is updating the corresponding github issue ticket. [Go to github issue](https://hwxiot.atlassian.net) and resolve the ticket.

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 clean install -Dhdp=2.4       # or -Dhdp=2.5
```

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

```
mvn versions:set -DnewVersion=0.10.0-rc1
mvn license:format
mvn test
mvn -DperformRelease=true clean deploy
mvn versions:commit

git tag -a v0.10.0-rc1 -m "new logisland release 0.10.0-rc1"
git push origin v0.10.0-rc1
```

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

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

Publish Docker image

Building the image

```
# build logisland
mvn clean install -DskipTests -Pdocker -Dhdp=2.4

# 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

Tutorials

Chat with us on Gitter Download the latest release build and unzip on an edge node.

Contents:

Index Apache logs

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

We will start a Docker container hosting all the LogIsland services, launch two streaming processes and send some apache logs to the system in order to analyze them in a dashboard.

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. The docker container is built from a Centos 6.4 image with the following tools enabled

- Kafka
- Spark
- Elasticsearch
- Kibana
- Logstash
- Flume
- Nginx
- 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.

2. Parse the logs records

For this tutorial we will handle some apache logs with a splitText parser and send 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-apache-logs.yml
```

Setup Spark/Kafka streaming engine

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) 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: Main Logisland job entry point
  configuration:
    spark.app.name: LogislandTutorial
    spark.master: local[4]
    spark.driver.memory: 1G
    spark.driver.cores: 1
    spark.executor.memory: 3G
    spark.executor.instances: 4
    spark.executor.cores: 2
    spark.yarn.queue: default
    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 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
  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.

Stream 2 :Index the processed records to Elasticsearch

The second Kafka stream will handle `Records` pushed into `logisland_events` topic to index them into elasticsearch

```
- stream: indexing_stream
  component: com.hurence.logisland.processor.chain.KafkaRecordStream
  type: processor
  documentation: a processor that pushes 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: 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:

    # 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. 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:

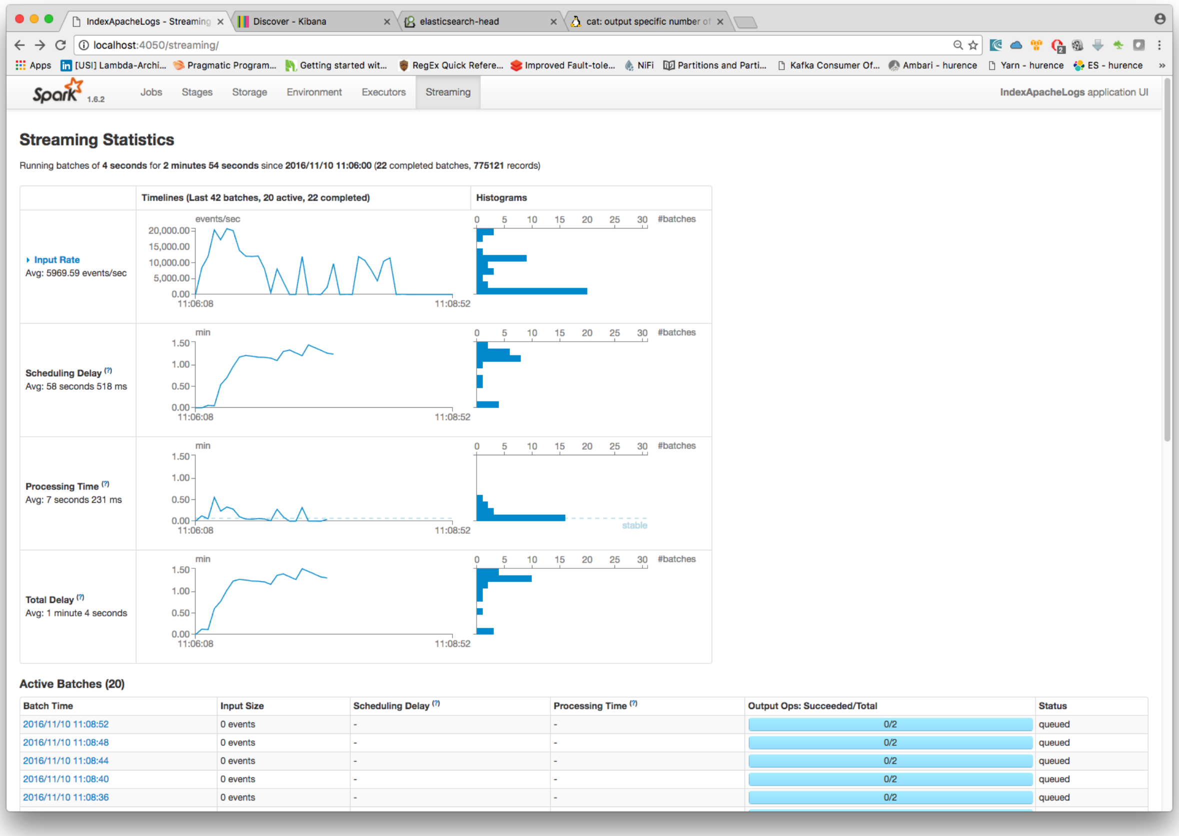
- Jul 01 to Jul 31, ASCII format, 20.7 MB gzip compressed
- Aug 04 to Aug 31, ASCII format, 21.8 MB gzip compressed

Let's send the first 500000 lines of NASA http access over July 1995 to LogIsland with kafkacat to `logisland_raw` Kafka topic

```
docker exec -ti logisland bash
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



IndexApacheLogs - Streaming
Discover - Kibana
elasticsearch-head
cat: output specific number of
localhost:4050/streaming/
Jobs
Stages
Storage
Environment
Executors
Streaming
IndexApacheLogs application UI
Timelines (Last 42 batches, 20 active, 22 completed)
Histograms
events/sec
min
#batches
11:06:08
11:08:52
stable

Streaming Statistics

Running batches of **4 seconds** for **2 minutes 54 seconds** since **2016/11/10 11:06:00** (**22** completed batches, **775121** records)

Input Rate
Avg: 5969.59 events/sec

Scheduling Delay (?)
Avg: 58 seconds 518 ms

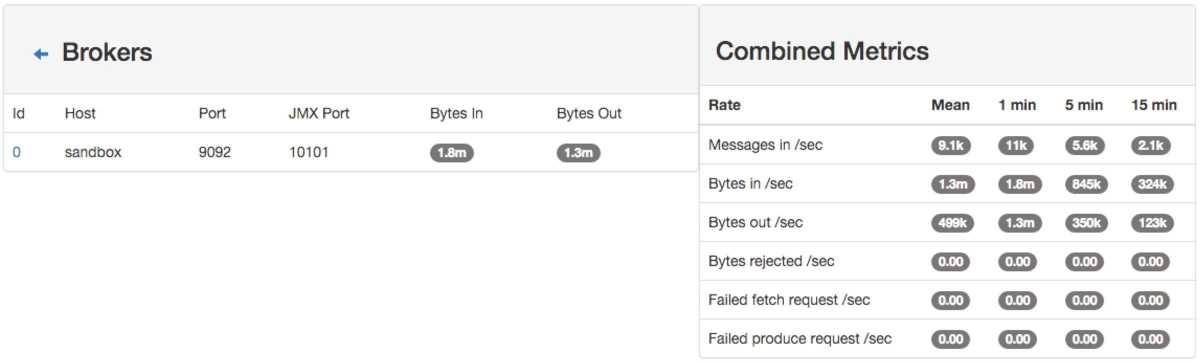
Processing Time (?)
Avg: 7 seconds 231 ms

Total Delay (?)
Avg: 1 minute 4 seconds

Active Batches (20)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Output Ops: Succeeded/Total	Status
2016/11/10 11:08:52	0 events	-	-	0/2	queued
2016/11/10 11:08:48	0 events	-	-	0/2	queued
2016/11/10 11:08:44	0 events	-	-	0/2	queued
2016/11/10 11:08:40	0 events	-	-	0/2	queued
2016/11/10 11:08:36	0 events	-	-	0/2	queued

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

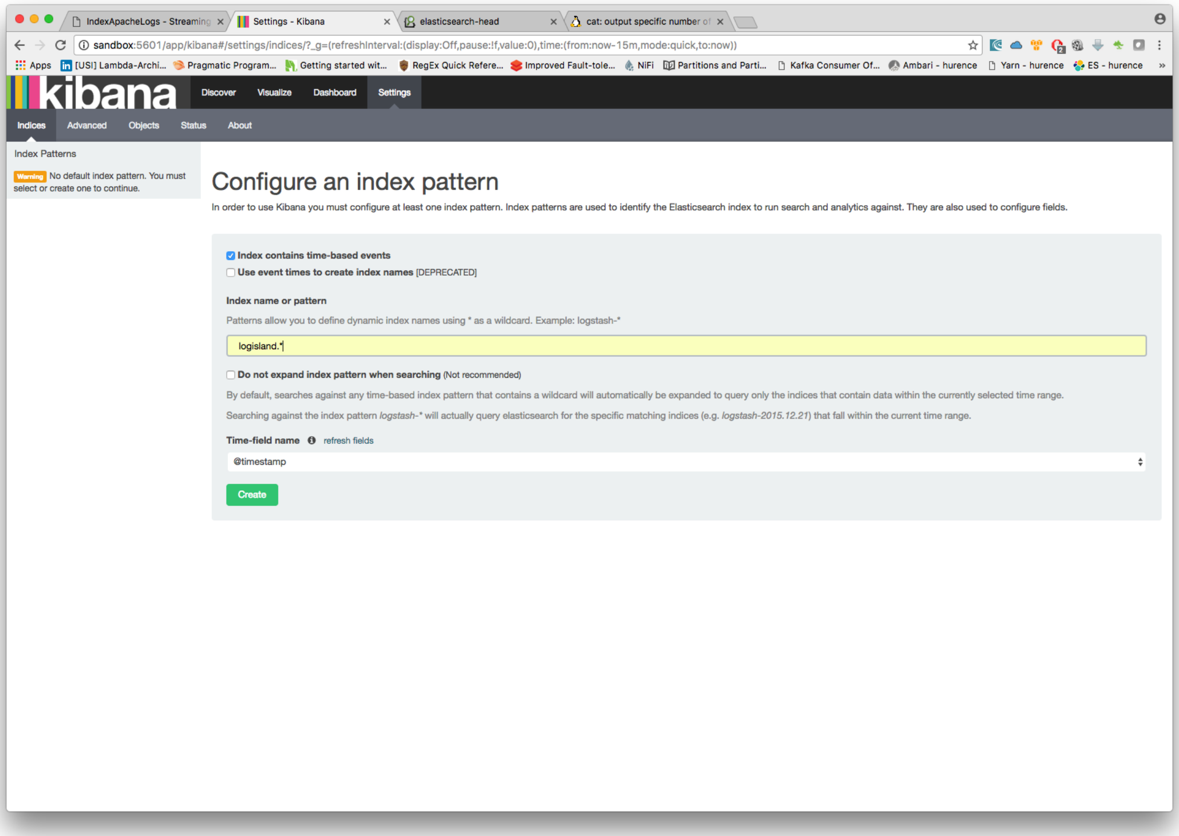
Combined Metrics

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

5. Use Kibana to inspect the logs

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.



sandbox:5601/app/kibana#/settings/indices/?_g=(refreshInterval:(display:Off,pause:!f,value:0),time:(from:now-15m,mode:quick,to:now))

Warning No default index pattern. You must select or create one to continue.

Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to identify the Elasticsearch index to run search and analytics against. They are also used to configure fields.

[✓] Index contains time-based events

[ ] Use event times to create index names [DEPRECATED]

Index name or pattern

Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*

logisland.*

[ ] Do not expand index pattern when searching (Not recommended)

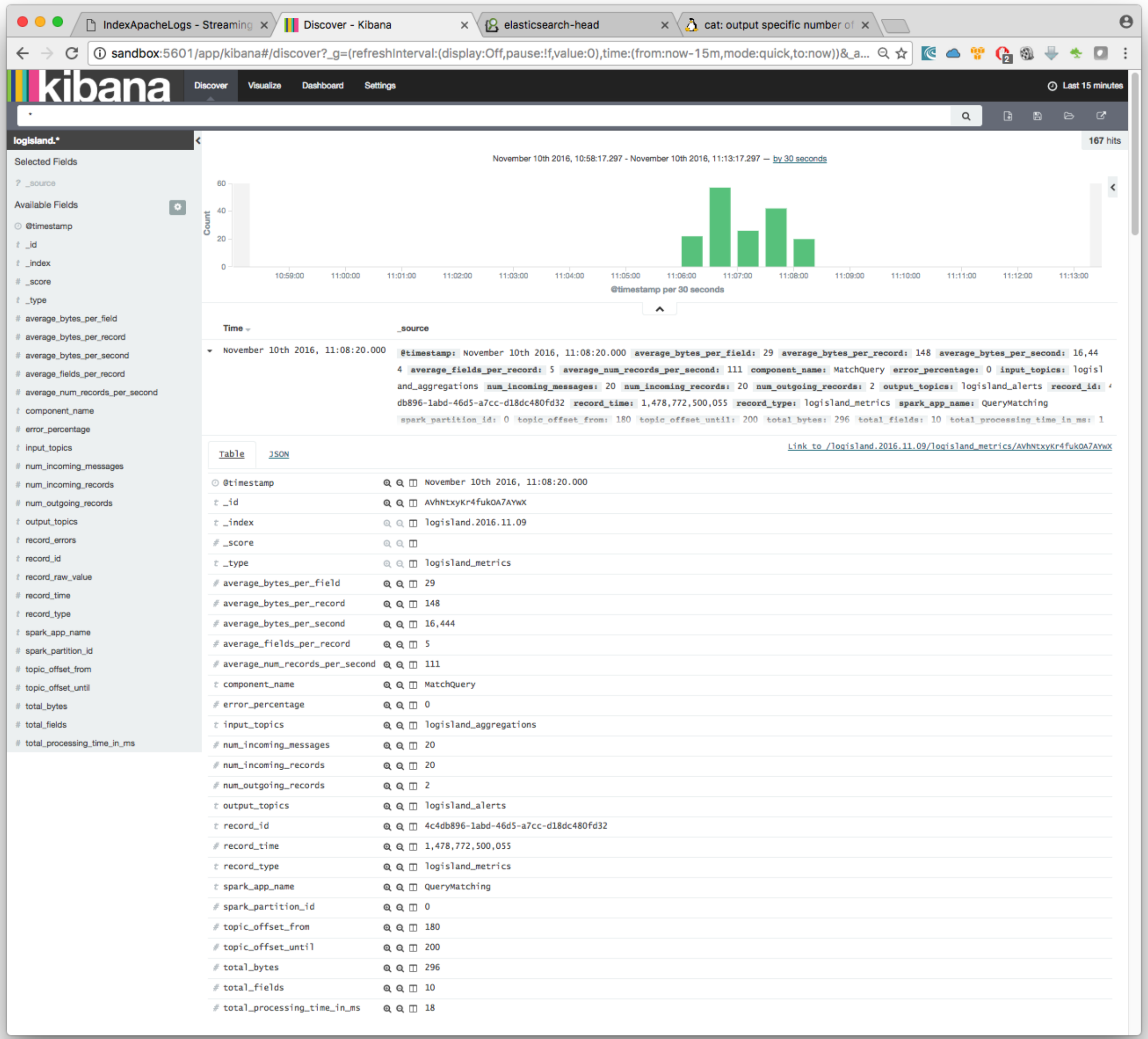
By default, searches against any time-based index pattern that contains a wildcard will automatically be expanded to query only the indices that contain data within the currently selected time range.

Searching against the index pattern *logstash-** will actually query elasticsearch for the specific matching indices (e.g. *logstash-2015.12.21*) that fall within the current time range.

Time-field name refresh fields

@timestamp

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.

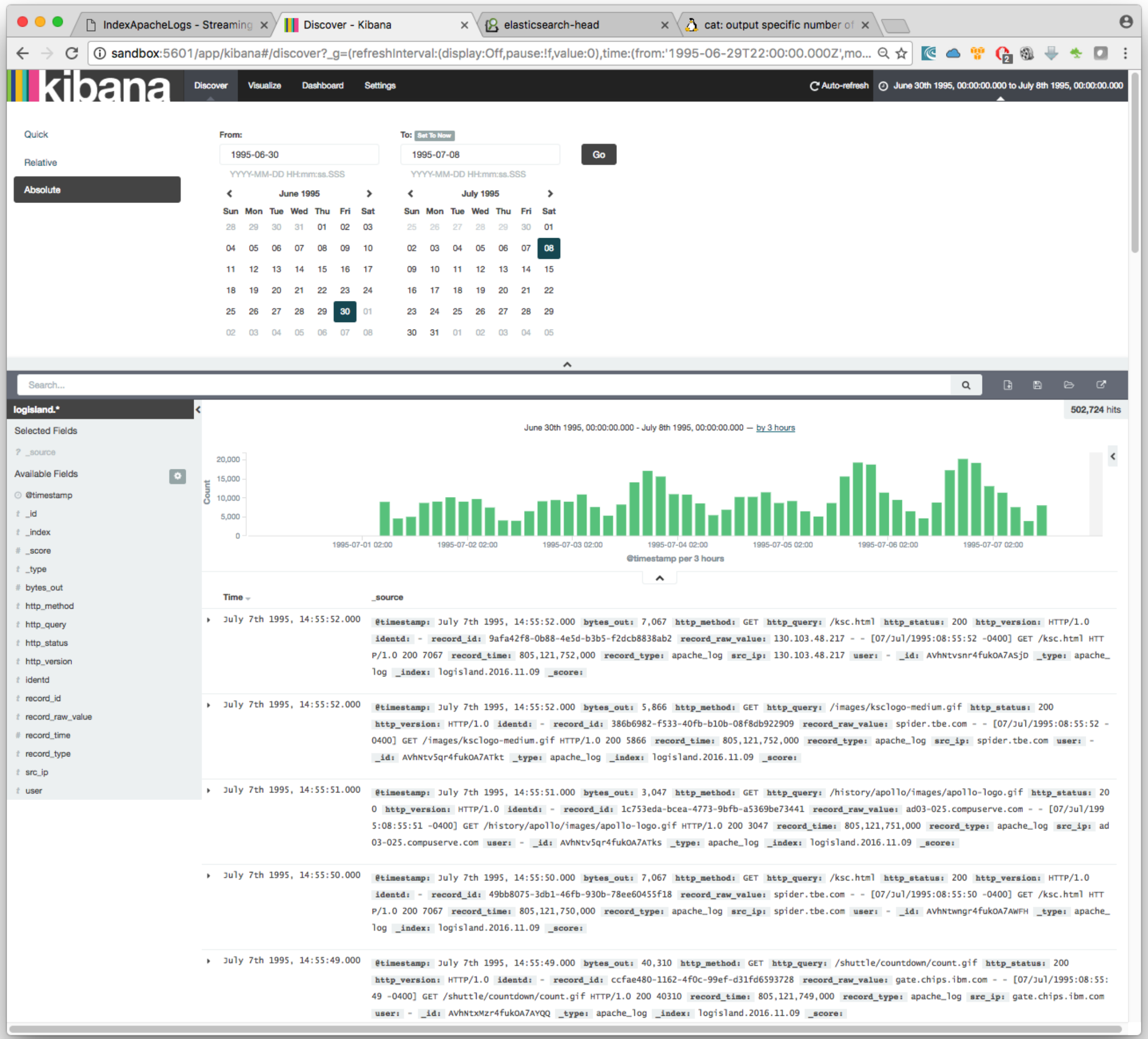


```
November 10th 2016, 11:08:20.000   @timestamp: November 10th 2016, 11:08:20.000 average_bytes_per_field: 29 average_bytes_per_record: 148 average_bytes_per_second: 16,44
4 average_fields_per_record: 5 average_num_records_per_second: 111 component_name: MatchQuery error_percentage: 0 input_topics: logisl
and_aggregations num_incoming_messages: 20 num_incoming_records: 20 num_outgoing_records: 2 output_topics: logisland_alerts record_id: 4
db896-1abd-46d5-a7cc-d18dc480fd32 record_time: 1,478,772,500,055 record_type: logisland_metrics spark_app_name: QueryMatching
spark_partition_id: 0 topic_offset_from: 180 topic_offset_until: 200 total_bytes: 296 total_fields: 10 total_processing_time_in_ms: 1
```

Table JSON Link to /logisland.2016.11.09/logisland_metrics/AVhNtxyKr4fukOA7AYwX

@timestamp	November 10th 2016, 11:08:20.000
_id	AVhNtxyKr4fukOA7AYwX
_index	logisland.2016.11.09
_score	
_type	logisland_metrics
average_bytes_per_field	29
average_bytes_per_record	148
average_bytes_per_second	16,444
average_fields_per_record	5
average_num_records_per_second	111
component_name	MatchQuery
error_percentage	0
input_topics	logisland_aggregations
num_incoming_messages	20
num_incoming_records	20
num_outgoing_records	2
output_topics	logisland_alerts
record_id	4c4db896-1abd-46d5-a7cc-d18dc480fd32
record_time	1,478,772,500,055
record_type	logisland_metrics
spark_app_name	QueryMatching
spark_partition_id	0
topic_offset_from	180
topic_offset_until	200
total_bytes	296
total_fields	10
total_processing_time_in_ms	18

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.



```
Time            _source
July 7th 1995, 14:55:52.000  @timestamp: July 7th 1995, 14:55:52.000 bytes_out: 7,067 http_method: GET http_query: /ksc.html http_status: 200 http_version: HTTP/1.0 identd: - record_id: 9afa42f8-0b88-4e5d-b3b5-f2dcb8838ab2 record_raw_value: 130.103.48.217 - - [07/Jul/1995:08:55:52 -0400] GET /ksc.html HTTP/1.0 200 7067 record_time: 805,121,752,000 record_type: apache_log src_ip: 130.103.48.217 user: - _id: AVhNtvsnr4fukOA7ASjD _type: apache_log _index: logisland.2016.11.09 _score:
July 7th 1995, 14:55:52.000  @timestamp: July 7th 1995, 14:55:52.000 bytes_out: 5,866 http_method: GET http_query: /images/ksclogo-medium.gif http_status: 200 http_version: HTTP/1.0 identd: - record_id: 386b6982-f533-40fb-b10b-08f8db922909 record_raw_value: spider.tbe.com - - [07/Jul/1995:08:55:52 -0400] GET /images/ksclogo-medium.gif HTTP/1.0 200 5866 record_time: 805,121,752,000 record_type: apache_log src_ip: spider.tbe.com user: - _id: AVhNtv5qr4fukOA7ATkt _type: apache_log _index: logisland.2016.11.09 _score:
July 7th 1995, 14:55:51.000  @timestamp: July 7th 1995, 14:55:51.000 bytes_out: 3,047 http_method: GET http_query: /history/apollo/images/apollo-logo.gif http_status: 200 http_version: HTTP/1.0 identd: - record_id: 1c753eda-bcea-4773-9bfb-a5369be73441 record_raw_value: ad03-025.compuserve.com - - [07/Jul/1995:08:55:51 -0400] GET /history/apollo/images/apollo-logo.gif HTTP/1.0 200 3047 record_time: 805,121,751,000 record_type: apache_log src_ip: ad03-025.compuserve.com user: - _id: AVhNtv5qr4fukOA7ATks _type: apache_log _index: logisland.2016.11.09 _score:
July 7th 1995, 14:55:50.000  @timestamp: July 7th 1995, 14:55:50.000 bytes_out: 7,067 http_method: GET http_query: /ksc.html http_status: 200 http_version: HTTP/1.0 identd: - record_id: 49bb8075-3db1-46fb-930b-78ee60455f18 record_raw_value: spider.tbe.com - - [07/Jul/1995:08:55:50 -0400] GET /ksc.html HTTP/1.0 200 7067 record_time: 805,121,750,000 record_type: apache_log src_ip: spider.tbe.com user: - _id: AVhNtwngr4fukOA7AWFH _type: apache_log _index: logisland.2016.11.09 _score:
July 7th 1995, 14:55:49.000  @timestamp: July 7th 1995, 14:55:49.000 bytes_out: 40,310 http_method: GET http_query: /shuttle/countdown/count.gif http_status: 200 http_version: HTTP/1.0 identd: - record_id: ccfae480-1162-4f0c-99ef-d31fd6593728 record_raw_value: gate.chips.ibm.com - - [07/Jul/1995:08:55:49 -0400] GET /shuttle/countdown/count.gif HTTP/1.0 200 40310 record_time: 805,121,749,000 record_type: apache_log src_ip: gate.chips.ibm.com user: - _id: AVhNtxMzr4fukOA7AYQQ _type: apache_log _index: logisland.2016.11.09 _score:
```

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 step 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).

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.

We will reuse the Docker container hosting all the LogIsland services from the previous tutorial, and add the User-Agent 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 User-Agent processor)

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/user-agent-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 the processing of the User-Agent string after the parsing of the logs
```

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

```
# 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"] } ,
           { "name": "http_user_agent",   "type": ["string","null"] },
           { "name": "http_referer",      "type": ["string","null"] },
           { "name": "DeviceClass",  "type": ["string","null"] },
           { "name": "DeviceName",  "type": ["string","null"] },
           { "name": "DeviceBrand",  "type": ["string","null"] },
           { "name": "DeviceCpu",  "type": ["string","null"] },
           { "name": "DeviceFirmwareVersion",  "type": ["string","null"] },
           { "name": "DeviceVersion",  "type": ["string","null"] },
           { "name": "OperatingSystemClass",  "type": ["string","null"] },
           { "name": "OperatingSystemName",  "type": ["string","null"] },
           { "name": "OperatingSystemVersion",  "type": ["string","null"] },
           { "name": "OperatingSystemNameVersion",  "type": ["string","null"] },
           { "name": "OperatingSystemVersionBuild",  "type": ["string","null"] },
           { "name": "LayoutEngineClass",  "type": ["string","null"] },
           { "name": "LayoutEngineName",  "type": ["string","null"] },
           { "name": "LayoutEngineVersion",  "type": ["string","null"] },
           { "name": "LayoutEngineVersionMajor",  "type": ["string","null"] },
           { "name": "LayoutEngineNameVersion",  "type": ["string","null"] },
           { "name": "LayoutEngineNameVersionMajor",  "type": ["string","null"] },
           { "name": "LayoutEngineBuild",  "type": ["string","null"] },
           { "name": "AgentClass",  "type": ["string","null"] },
           { "name": "AgentName",  "type": ["string","null"] },
           { "name": "AgentVersion",  "type": ["string","null"] },
           { "name": "AgentVersionMajor",  "type": ["string","null"] },
           { "name": "AgentNameVersion",  "type": ["string","null"] },
           { "name": "AgentNameVersionMajor",  "type": ["string","null"] },
           { "name": "AgentBuild",  "type": ["string","null"] },
           { "name": "AgentLanguage",  "type": ["string","null"] },
           { "name": "AgentLanguageCode",  "type": ["string","null"] },
           { "name": "AgentInformationEmail",  "type": ["string","null"] },
           { "name": "AgentInformationUrl",  "type": ["string","null"] },
           { "name": "AgentSecurity",  "type": ["string","null"] },
           { "name": "AgentUuid",  "type": ["string","null"] },
           { "name": "FacebookCarrier",  "type": ["string","null"] },
           { "name": "FacebookDeviceClass",  "type": ["string","null"] },
           { "name": "FacebookDeviceName",  "type": ["string","null"] },
           { "name": "FacebookDeviceVersion",  "type": ["string","null"] },
           { "name": "FacebookFBOP",  "type": ["string","null"] },
           { "name": "FacebookFBSS",  "type": ["string","null"] },
           { "name": "FacebookOperatingSystemName",  "type": ["string","null"] },
           { "name": "FacebookOperatingSystemVersion",  "type": ["string","null"] },
           { "name": "Anonymized",  "type": ["string","null"] },
           { "name": "HackerAttackVector",  "type": ["string","null"] },
           { "name": "HackerToolkit",  "type": ["string","null"] },
           { "name": "KoboAffiliate",  "type": ["string","null"] },
           { "name": "KoboPlatformId",  "type": ["string","null"] },
           { "name": "IECompatibilityVersion",  "type": ["string","null"] },
           { "name": "IECompatibilityVersionMajor",  "type": ["string","null"] },
           { "name": "IECompatibilityNameVersion",  "type": ["string","null"] },
           { "name": "IECompatibilityNameVersionMajor",  "type": ["string","null"] },
           { "name": "Carrier",  "type": ["string","null"] },
           { "name": "GSAInstallationID",  "type": ["string","null"] },
          { "name": "WebviewAppName",  "type": ["string","null"] },
          { "name": "WebviewAppNameVersionMajor",  "type": ["string","null"] },
          { "name": "WebviewAppVersion",  "type": ["string","null"] },
          { "name": "WebviewAppVersionMajor",  "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:

    # 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:
        record.type: apache_log
        # Previous regex
        #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
        # Updated regex
        value.regex: (\S+)\s+(\S+)\s+(\S+)\s+\[([\w:\/]+\s[+\-]\d{4})\]\s+
↪"(\S+)\s+(\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,http_referer,http_user_agent

    - processor: user_agent_analyzer
      component: com.hurence.logisland.processor.useragent.ParseUserAgent
      type: processor
      documentation: decompose the user_agent field into meaningful attributes
      configuration:
        useragent.field: http_user_agent
        fields: DeviceClass,DeviceName,DeviceBrand,DeviceCpu,DeviceFirmwareVersion,
↪DeviceVersion,OperatingSystemClass,OperatingSystemName,OperatingSystemVersion,
↪OperatingSystemNameVersion,OperatingSystemVersionBuild,LayoutEngineClass,
↪LayoutEngineName,LayoutEngineVersion,LayoutEngineVersionMajor,
↪LayoutEngineNameVersion,LayoutEngineNameVersionMajor,LayoutEngineBuild,AgentClass,
↪AgentName,AgentVersion,AgentVersionMajor,AgentNameVersion,AgentNameVersionMajor,
↪AgentBuild,AgentLanguage,AgentLanguageCode,AgentInformationEmail,
↪AgentInformationUrl,AgentSecurity,AgentUuid,FacebookCarrier,FacebookDeviceClass,
↪FacebookDeviceName,FacebookDeviceVersion,FacebookFBOP,FacebookFBSS,
↪FacebookOperatingSystemName,FacebookOperatingSystemVersion,Anonymized,
↪HackerAttackVector,HackerToolkit,KoboAffiliate,KoboPlatformId,
↪IECompatibilityVersion,IECompatibilityVersionMajor,IECompatibilityNameVersion,
↪IECompatibilityNameVersionMajor,GSAInstallationID,WebviewAppName,
↪WebviewAppNameVersionMajor,WebviewAppVersion,WebviewAppVersionMajor
```

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).

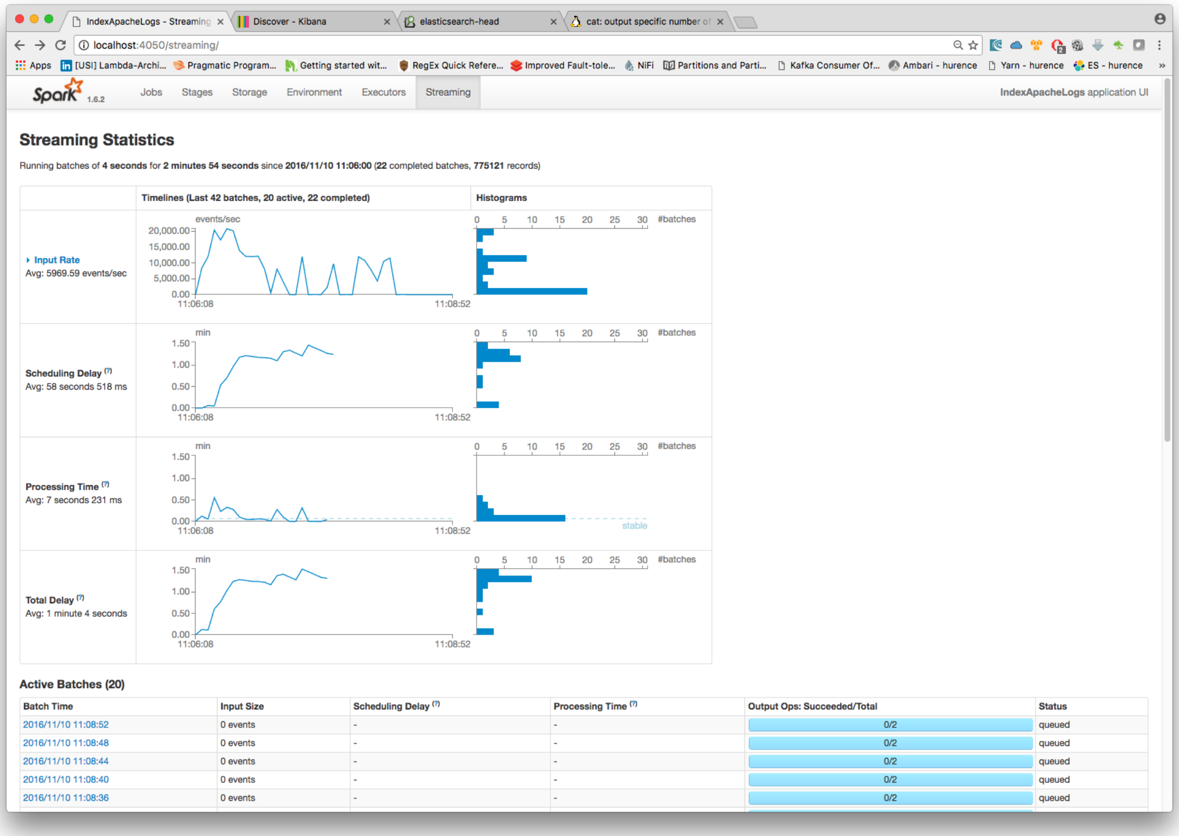
If you don't have your own httpd logs available, you can use some freely available log files from Elastic web site

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
```

3. Monitor your spark jobs and Kafka topics

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


IndexApacheLogs - Streaming
Discover - Kibana
elasticsearch-head
cat: output specific number of
localhost:4050/streaming/
Spark 1.6.2
Jobs
Stages
Storage
Environment
Executors
Streaming
IndexApacheLogs application UI

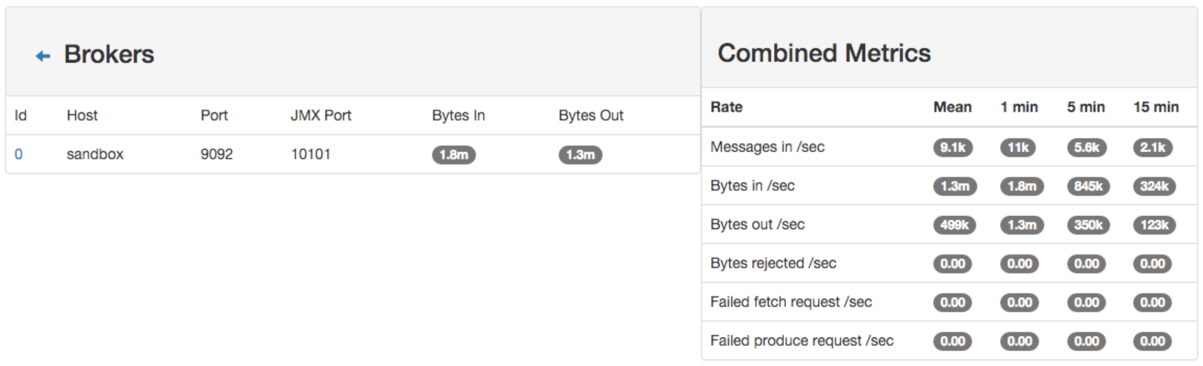
Streaming Statistics

Running batches of **4 seconds** for **2 minutes 54 seconds** since **2016/11/10 11:06:00** (**22** completed batches, **775121** records)

Active Batches (20)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Output Ops: Succeeded/Total	Status
2016/11/10 11:08:52	0 events	-	-	0/2	queued
2016/11/10 11:08:48	0 events	-	-	0/2	queued
2016/11/10 11:08:44	0 events	-	-	0/2	queued
2016/11/10 11:08:40	0 events	-	-	0/2	queued
2016/11/10 11:08:36	0 events	-	-	0/2	queued

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 elastic-Search, 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",
        "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

Alerts & Query Matching

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: 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 SQL Aggregation Stream

Our application will be composed of 2 streams, the first 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

```
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
    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.

2. Setup Query matching Stream on log Records

The second stream makes use of the KafkaRecordStreamParallelProcessing Stream with a 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.

We'll use 2 streams for query matching because we will handle 2 kind of Records. The first one will send an alert when a particular host (src_ip:199.0.2.27) will make a connection and anywhen someone from *.edu domain makes a connection (src_ip:*.edu).

```
# match threshold queries
- stream: query_matching_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that match query in parrallel
  configuration:
    kafka.input.topics: logisland_events
    kafka.output.topics: logisland_alerts
    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: match_query
      component: com.hurence.logisland.processor.MatchQuery
      type: processor
      documentation: a parser that produce events from an apache log REGEX
      configuration:
        blacklisted_host: src_ip:slip-5.io.com
        edu_host: src_ip:edu
        output.record.type: connection_alert
```

3. Setup Query matching Stream

The third one will match numeric fields on sql aggregates computed in the very first stream in this tutorial.

```
# match threshold queries
- stream: query_matching_stream
  component: com.hurence.logisland.stream.spark.KafkaRecordStreamParallelProcessing
  type: stream
  documentation: a processor that match query in parrallel
  configuration:
    kafka.input.topics: logisland_aggregations
    kafka.output.topics: logisland_alerts
    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: match_query
      component: com.hurence.logisland.processor.MatchQuery
      type: processor
      documentation: a parser that produce events from an apache log REGEX
      configuration:
        numeric.fields: bytes_out,connections_count
        too_much_bandwidth: average_bytes:[100 TO 50000]
        too_many_connections: connections_count:[500 TO 1000000]
        output.record.type: threshold_alert
```

4. Start logisland application

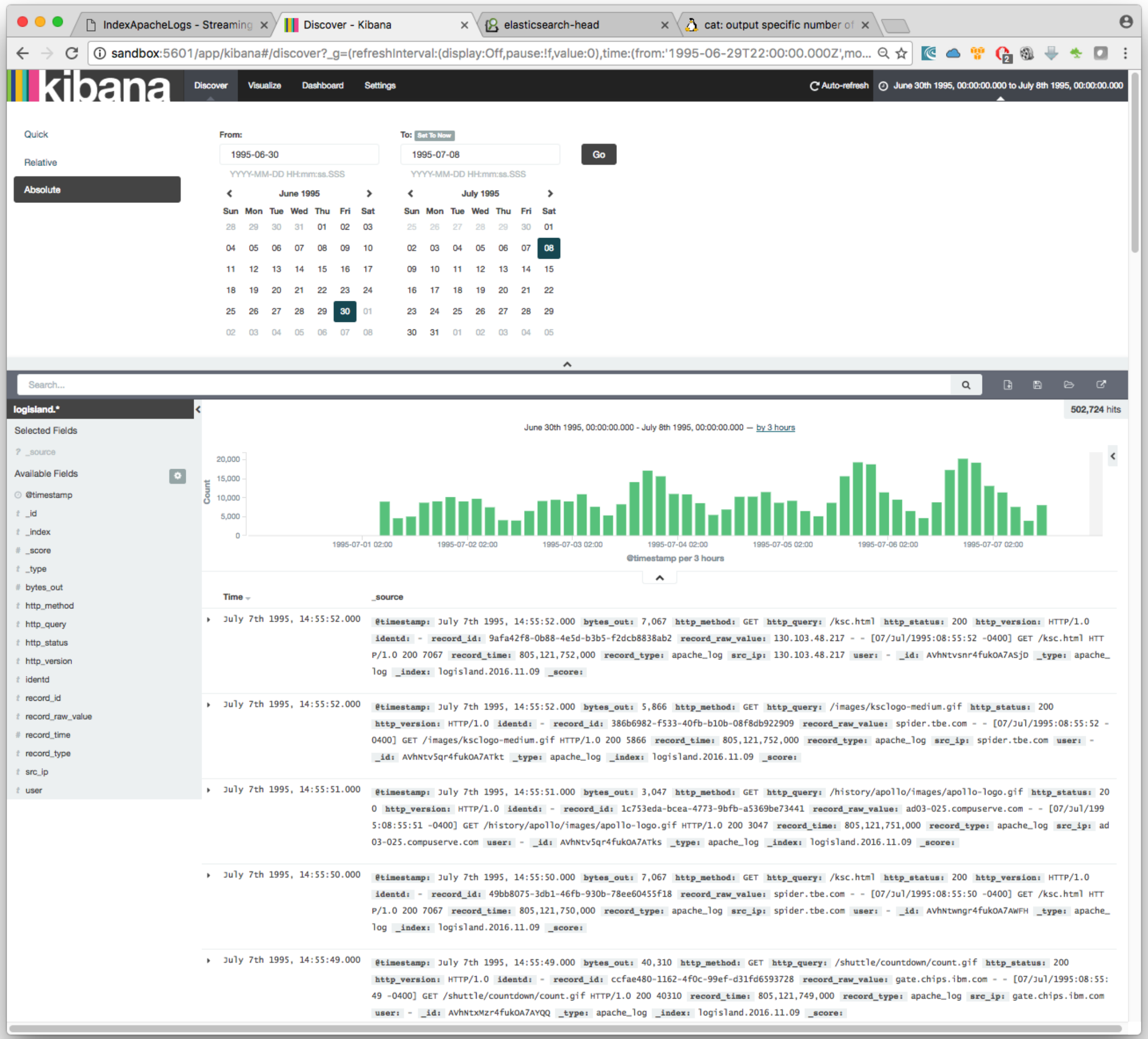
Connect a shell to your logisland container to launch the following stream processing job previously defined.

```
docker exec -ti logisland bash

#launch logisland streams
cd $LOGISLAND_HOME
bin/logisland.sh --conf conf/index-apache-logs.yml
bin/logisland.sh --conf conf/query-matching.yml

# send logs to kafka
head 500000 NASA_access_log_Jul95 | kafkacat -b sandbox:9092 -t logisland_raw
```

5. Check your alerts with Kibana

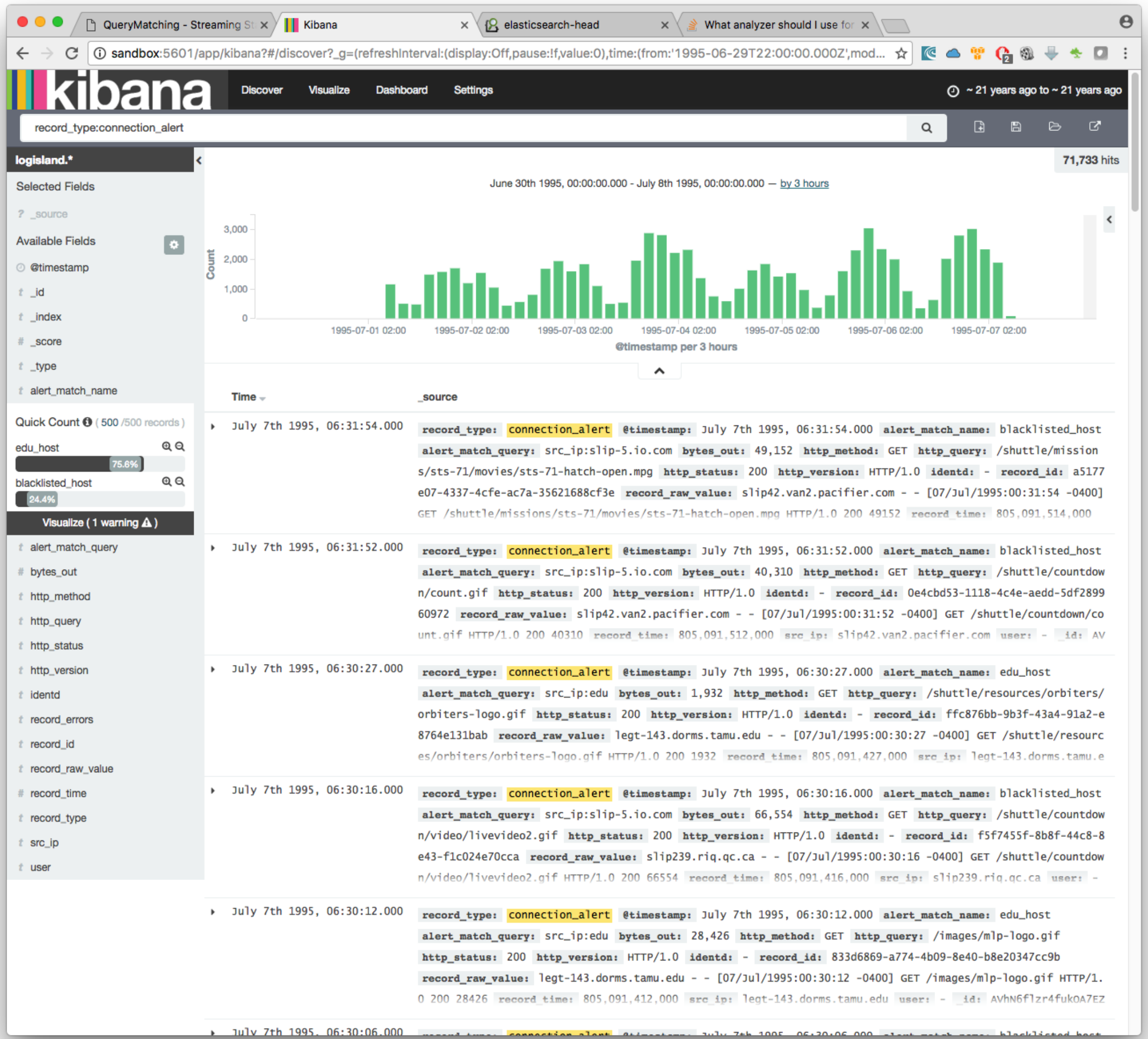
Open up your browser and go to http://sandbox:5601/ and you should be able to explore your apache logs.

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.



```
sandbox:5601/app/kibana#/discover?_g=(refreshInterval:(display:Off,pause:!f,value:0),time:(from:'1995-06-29T22:00:00.000Z',mo...
June 30th 1995, 00:00:00.000 to July 8th 1995, 00:00:00.000
From: 1995-06-30    To: 1995-07-08
logisland.*    502,724 hits
June 30th 1995, 00:00:00.000 - July 8th 1995, 00:00:00.000 — by 3 hours
Fields: _source, @timestamp, _id, _index, _score, _type, bytes_out, http_method, http_query, http_status, http_version, identd, record_id, record_raw_value, record_time, record_type, src_ip, user

Time                           _source
July 7th 1995, 14:55:52.000    @timestamp: July 7th 1995, 14:55:52.000 bytes_out: 7,067 http_method: GET http_query: /ksc.html http_status: 200 http_version: HTTP/1.0 identd: - record_id: 9afa42f8-0b88-4e5d-b3b5-f2dcb8838ab2 record_raw_value: 130.103.48.217 - - [07/Jul/1995:08:55:52 -0400] GET /ksc.html HTTP/1.0 200 7067 record_time: 805,121,752,000 record_type: apache_log src_ip: 130.103.48.217 user: - _id: AVhNtvsnr4fukOA7ASjD _type: apache_log _index: logisland.2016.11.09 _score:
July 7th 1995, 14:55:52.000    @timestamp: July 7th 1995, 14:55:52.000 bytes_out: 5,866 http_method: GET http_query: /images/ksclogo-medium.gif http_status: 200 http_version: HTTP/1.0 identd: - record_id: 386b6982-f533-40fb-b10b-08f8db922909 record_raw_value: spider.tbe.com - - [07/Jul/1995:08:55:52 -0400] GET /images/ksclogo-medium.gif HTTP/1.0 200 5866 record_time: 805,121,752,000 record_type: apache_log src_ip: spider.tbe.com user: - _id: AVhNtv5qr4fukOA7ATkt _type: apache_log _index: logisland.2016.11.09 _score:
July 7th 1995, 14:55:51.000    @timestamp: July 7th 1995, 14:55:51.000 bytes_out: 3,047 http_method: GET http_query: /history/apollo/images/apollo-logo.gif http_status: 200 http_version: HTTP/1.0 identd: - record_id: 1c753eda-bcea-4773-9bfb-a5369be73441 record_raw_value: ad03-025.compuserve.com - - [07/Jul/1995:08:55:51 -0400] GET /history/apollo/images/apollo-logo.gif HTTP/1.0 200 3047 record_time: 805,121,751,000 record_type: apache_log src_ip: ad03-025.compuserve.com user: - _id: AVhNtv5qr4fukOA7ATks _type: apache_log _index: logisland.2016.11.09 _score:
July 7th 1995, 14:55:50.000    @timestamp: July 7th 1995, 14:55:50.000 bytes_out: 7,067 http_method: GET http_query: /ksc.html http_status: 200 http_version: HTTP/1.0 identd: - record_id: 49bb8075-3db1-46fb-930b-78ee60455f18 record_raw_value: spider.tbe.com - - [07/Jul/1995:08:55:50 -0400] GET /ksc.html HTTP/1.0 200 7067 record_time: 805,121,750,000 record_type: apache_log src_ip: spider.tbe.com user: - _id: AVhNtwngr4fukOA7AWFH _type: apache_log _index: logisland.2016.11.09 _score:
July 7th 1995, 14:55:49.000    @timestamp: July 7th 1995, 14:55:49.000 bytes_out: 40,310 http_method: GET http_query: /shuttle/countdown/count.gif http_status: 200 http_version: HTTP/1.0 identd: - record_id: ccfae480-1162-4f0c-99ef-d31fd6593728 record_raw_value: gate.chips.ibm.com - - [07/Jul/1995:08:55:49 -0400] GET /shuttle/countdown/count.gif HTTP/1.0 200 40310 record_time: 805,121,749,000 record_type: apache_log src_ip: gate.chips.ibm.com user: - _id: AVhNtxMzr4fukOA7AYQQ _type: apache_log _index: logisland.2016.11.09 _score:
```

you can filter your events with `record_type:connection_alert` to get 71733 connections alerts matching your query

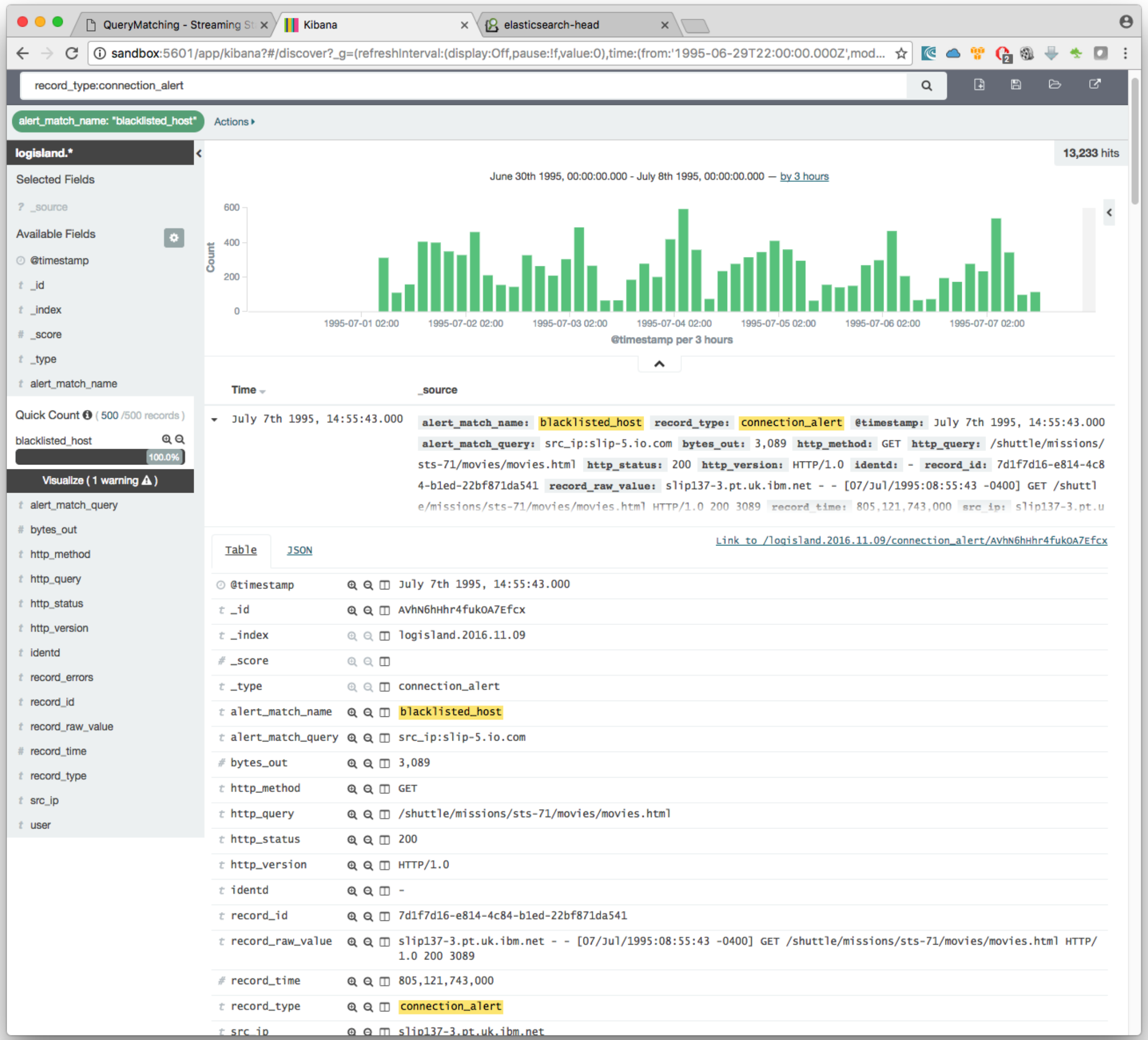


```
sandbox:5601/app/kibana?#/discover?_g=(refreshInterval:(display:Off,pause:!f,value:0),time:(from:'1995-06-29T22:00:00.000Z',mod...
```

```
record_type:connection_alert
```

```
July 7th 1995, 06:31:54.000   record_type: connection_alert @timestamp: July 7th 1995, 06:31:54.000 alert_match_name: blacklisted_host
                              alert_match_query: src_ip:slip-5.io.com bytes_out: 49,152 http_method: GET http_query: /shuttle/mission
                              s/sts-71/movies/sts-71-hatch-open.mpg http_status: 200 http_version: HTTP/1.0 identd: - record_id: a5177
                              e07-4337-4cfe-ac7a-35621688cf3e record_raw_value: slip42.van2.pacifier.com - - [07/Jul/1995:00:31:54 -0400]
                              GET /shuttle/missions/sts-71/movies/sts-71-hatch-open.mpg HTTP/1.0 200 49152 record_time: 805,091,514,000

July 7th 1995, 06:31:52.000   record_type: connection_alert @timestamp: July 7th 1995, 06:31:52.000 alert_match_name: blacklisted_host
                              alert_match_query: src_ip:slip-5.io.com bytes_out: 40,310 http_method: GET http_query: /shuttle/countdow
                              n/count.gif http_status: 200 http_version: HTTP/1.0 identd: - record_id: 0e4cbd53-1118-4c4e-aedd-5df2899
                              60972 record_raw_value: slip42.van2.pacifier.com - - [07/Jul/1995:00:31:52 -0400] GET /shuttle/countdown/co
                              unt.gif HTTP/1.0 200 40310 record_time: 805,091,512,000 src_ip: slip42.van2.pacifier.com user: - _id: AV

July 7th 1995, 06:30:27.000   record_type: connection_alert @timestamp: July 7th 1995, 06:30:27.000 alert_match_name: edu_host
                              alert_match_query: src_ip:edu bytes_out: 1,932 http_method: GET http_query: /shuttle/resources/orbiters/
                              orbiters-logo.gif http_status: 200 http_version: HTTP/1.0 identd: - record_id: ffc876bb-9b3f-43a4-91a2-e
                              8764e131bab record_raw_value: legt-143.dorms.tamu.edu - - [07/Jul/1995:00:30:27 -0400] GET /shuttle/resourc
                              es/orbiters/orbiters-logo.gif HTTP/1.0 200 1932 record_time: 805,091,427,000 src_ip: legt-143.dorms.tamu.e

July 7th 1995, 06:30:16.000   record_type: connection_alert @timestamp: July 7th 1995, 06:30:16.000 alert_match_name: blacklisted_host
                              alert_match_query: src_ip:slip-5.io.com bytes_out: 66,554 http_method: GET http_query: /shuttle/countdow
                              n/video/livevideo2.gif http_status: 200 http_version: HTTP/1.0 identd: - record_id: f5f7455f-8b8f-44c8-8
                              e43-f1c024e70cca record_raw_value: slip239.riq.qc.ca - - [07/Jul/1995:00:30:16 -0400] GET /shuttle/countdow
                              n/video/livevideo2.gif HTTP/1.0 200 66554 record_time: 805,091,416,000 src_ip: slip239.riq.qc.ca user: -

July 7th 1995, 06:30:12.000   record_type: connection_alert @timestamp: July 7th 1995, 06:30:12.000 alert_match_name: edu_host
                              alert_match_query: src_ip:edu bytes_out: 28,426 http_method: GET http_query: /images/mlp-logo.gif
                              http_status: 200 http_version: HTTP/1.0 identd: - record_id: 833d6869-a774-4b09-8e40-b8e20347cc9b
                              record_raw_value: legt-143.dorms.tamu.edu - - [07/Jul/1995:00:30:12 -0400] GET /images/mlp-logo.gif HTTP/1.
                              0 200 28426 record_time: 805,091,412,000 src_ip: legt-143.dorms.tamu.edu user: - _id: AVhN6flzr4fukOA7EZ

July 7th 1995, 06:30:06.000   record_type: connection_alert @timestamp: July 7th 1995, 06:30:06.000 alert_match_name: blacklisted_host
```

by adding another filter on `alert_match_name:blacklisted_host` you'll only get request from `slip-5.io.com` which is a host we where monitoring.

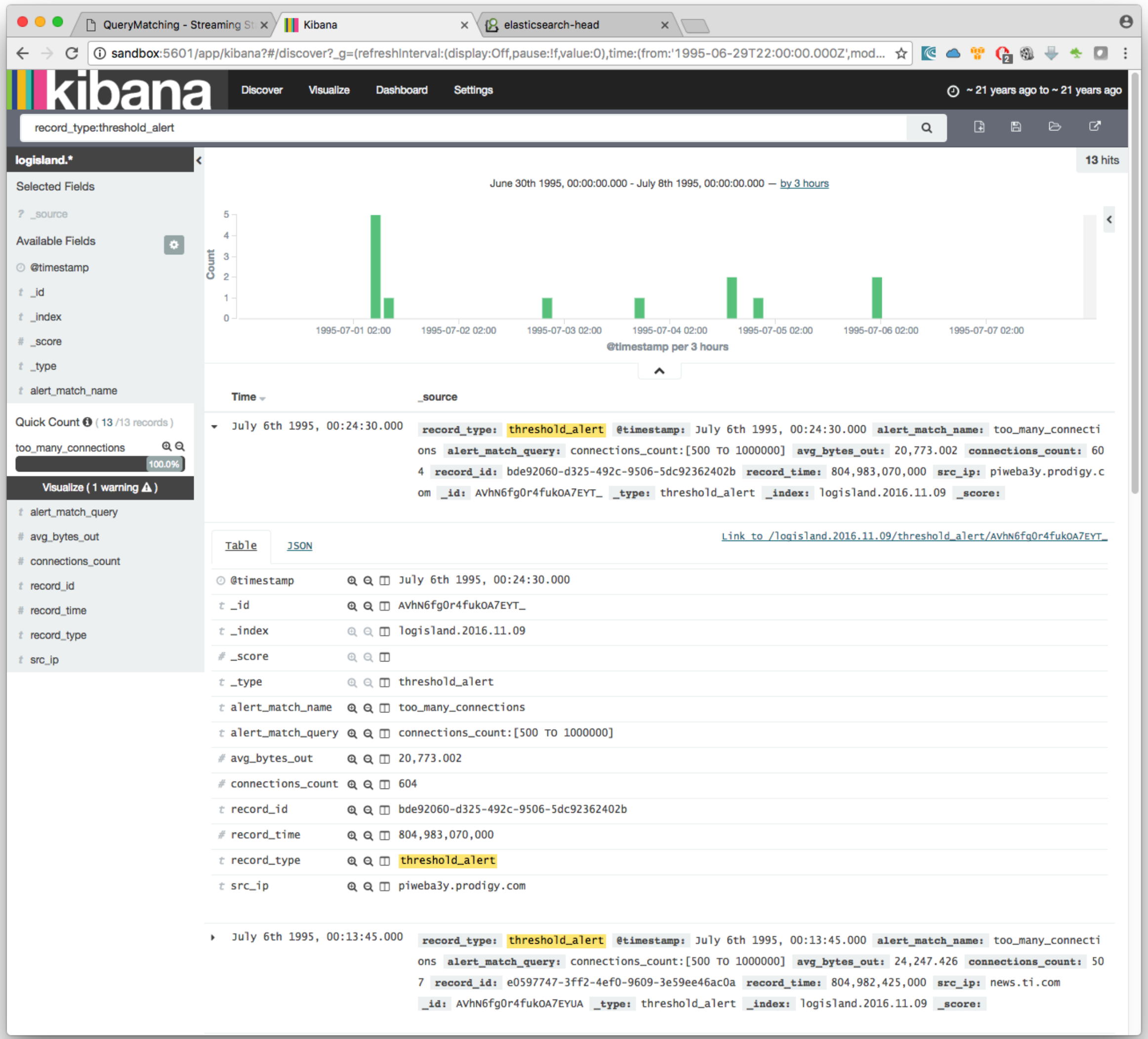


```
July 7th 1995, 14:55:43.000  alert_match_name: blacklisted_host record_type: connection_alert @timestamp: July 7th 1995, 14:55:43.000
alert_match_query: src_ip:slip-5.io.com bytes_out: 3,089 http_method: GET http_query: /shuttle/missions/
sts-71/movies/movies.html http_status: 200 http_version: HTTP/1.0 identd: - record_id: 7d1f7d16-e814-4c8
4-b1ed-22bf871da541 record_raw_value: slip137-3.pt.uk.ibm.net - - [07/Jul/1995:08:55:43 -0400] GET /shuttl
e/missions/sts-71/movies/movies.html HTTP/1.0 200 3089 record_time: 805,121,743,000 src_ip: slip137-3.pt.u
```

Link to /logisland.2016.11.09/connection_alert/AVhN6hHhr4fukOA7Efcx

@timestamp	July 7th 1995, 14:55:43.000
_id	AVhN6hHhr4fukOA7Efcx
_index	logisland.2016.11.09
_score	
_type	connection_alert
alert_match_name	blacklisted_host
alert_match_query	src_ip:slip-5.io.com
bytes_out	3,089
http_method	GET
http_query	/shuttle/missions/sts-71/movies/movies.html
http_status	200
http_version	HTTP/1.0
identd	-
record_id	7d1f7d16-e814-4c84-b1ed-22bf871da541
record_raw_value	slip137-3.pt.uk.ibm.net - - [07/Jul/1995:08:55:43 -0400] GET /shuttle/missions/sts-71/movies/movies.html HTTP/1.0 200 3089
record_time	805,121,743,000
record_type	connection_alert
src_ip	slip137-3.pt.uk.ibm.net

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.



record_type:threshold_alert

June 30th 1995, 00:00:00.000 - July 8th 1995, 00:00:00.000 — by 3 hours

```
July 6th 1995, 00:24:30.000   record_type: threshold_alert @timestamp: July 6th 1995, 00:24:30.000 alert_match_name: too_many_connecti
                              ons alert_match_query: connections_count:[500 TO 1000000] avg_bytes_out: 20,773.002 connections_count: 60
                              4 record_id: bde92060-d325-492c-9506-5dc92362402b record_time: 804,983,070,000 src_ip: piweba3y.prodigy.c
                              om _id: AVhN6fg0r4fukOA7EYT_ _type: threshold_alert _index: logisland.2016.11.09 _score:
```

Link to /logisland.2016.11.09/threshold_alert/AVhN6fg0r4fukOA7EYT_

```
@timestamp          July 6th 1995, 00:24:30.000
_id                 AVhN6fg0r4fukOA7EYT_
_index              logisland.2016.11.09
_score
_type               threshold_alert
alert_match_name    too_many_connections
alert_match_query   connections_count:[500 TO 1000000]
avg_bytes_out       20,773.002
connections_count   604
record_id           bde92060-d325-492c-9506-5dc92362402b
record_time         804,983,070,000
record_type         threshold_alert
src_ip              piweba3y.prodigy.com

July 6th 1995, 00:13:45.000   record_type: threshold_alert @timestamp: July 6th 1995, 00:13:45.000 alert_match_name: too_many_connecti
                              ons alert_match_query: connections_count:[500 TO 1000000] avg_bytes_out: 24,247.426 connections_count: 50
                              7 record_id: e0597747-3ff2-4ef0-9609-3e59ee46ac0a record_time: 804,982,425,000 src_ip: news.ti.com
                              _id: AVhN6fg0r4fukOA7EYUA _type: threshold_alert _index: logisland.2016.11.09 _score:
```

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
  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
    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:
        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
```

5. Check your alerts with Kibana

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. Transform Bro events into Logisland records

For this tutorial we will receive Bro events and notices and send them to Elastiscearch. 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 *3. 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

  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:

```
# 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.

```
# 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.

3. 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
```

4. 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:

```
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.

5. 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,
    "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",
      "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).

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" },
        "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`).

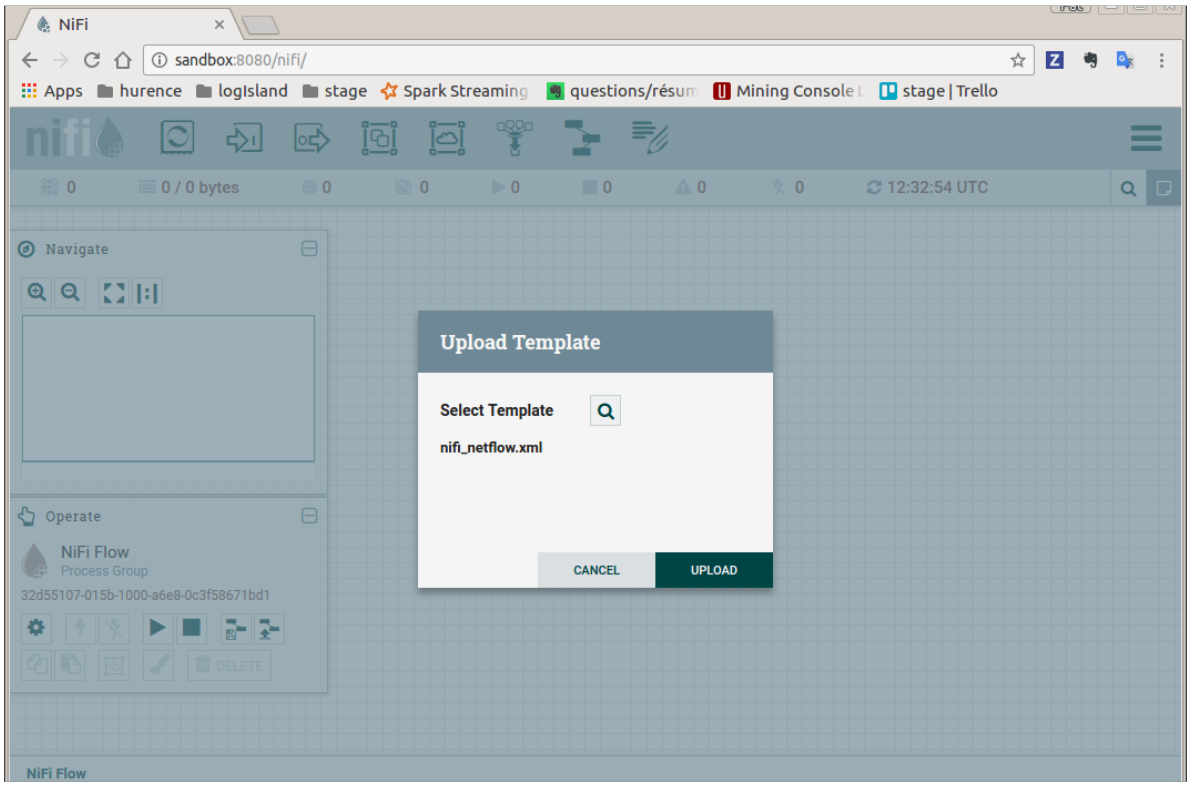
1. Start nifi

   ```
   docker exec -ti logisland bash
   cd /usr/local/nifi-1.1.1
   bin/nifi.sh start
   ```

   browse http://sandbox:8080/nifi/

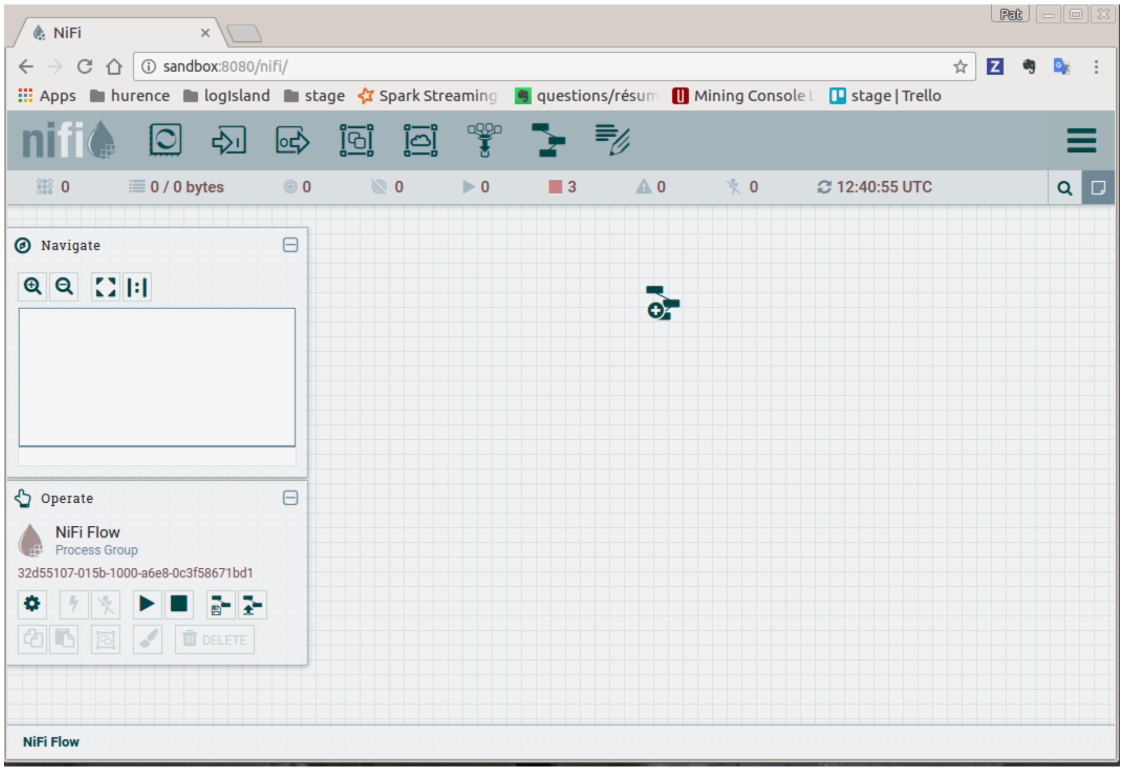
2. Import flow template

   Download this nifi template and import it using "Upload Template" in "Operator" toolbox.

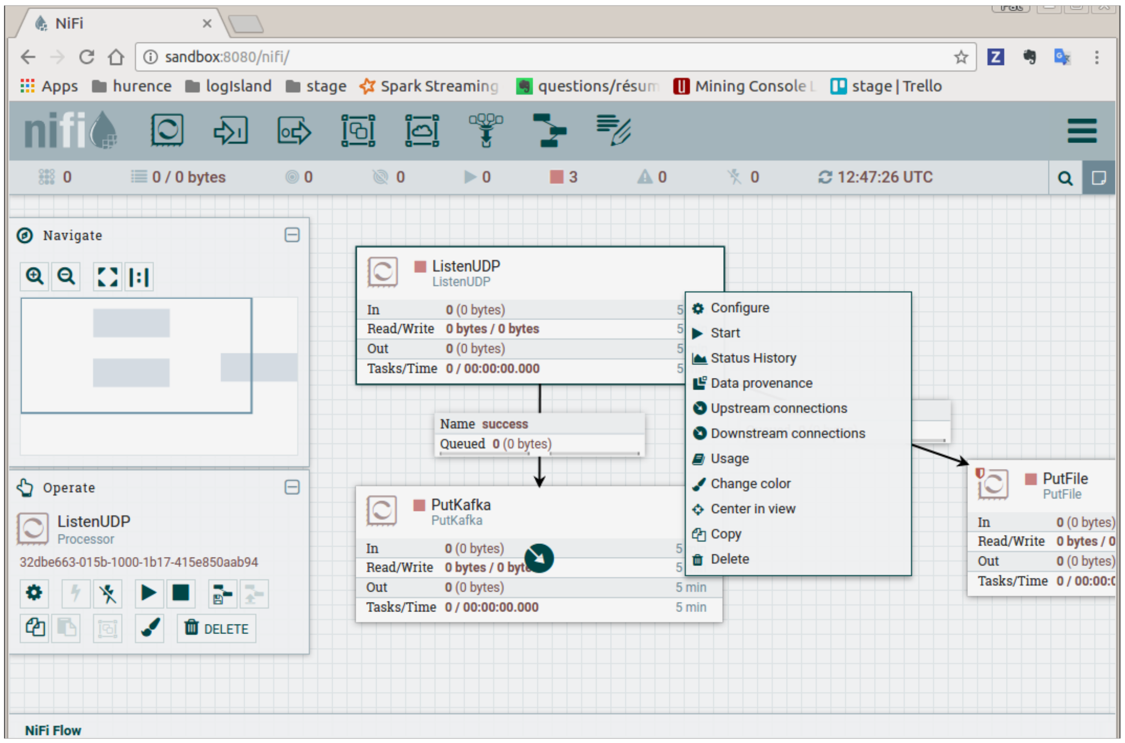


3. Use this template to create the nifi flow

   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
  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
    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

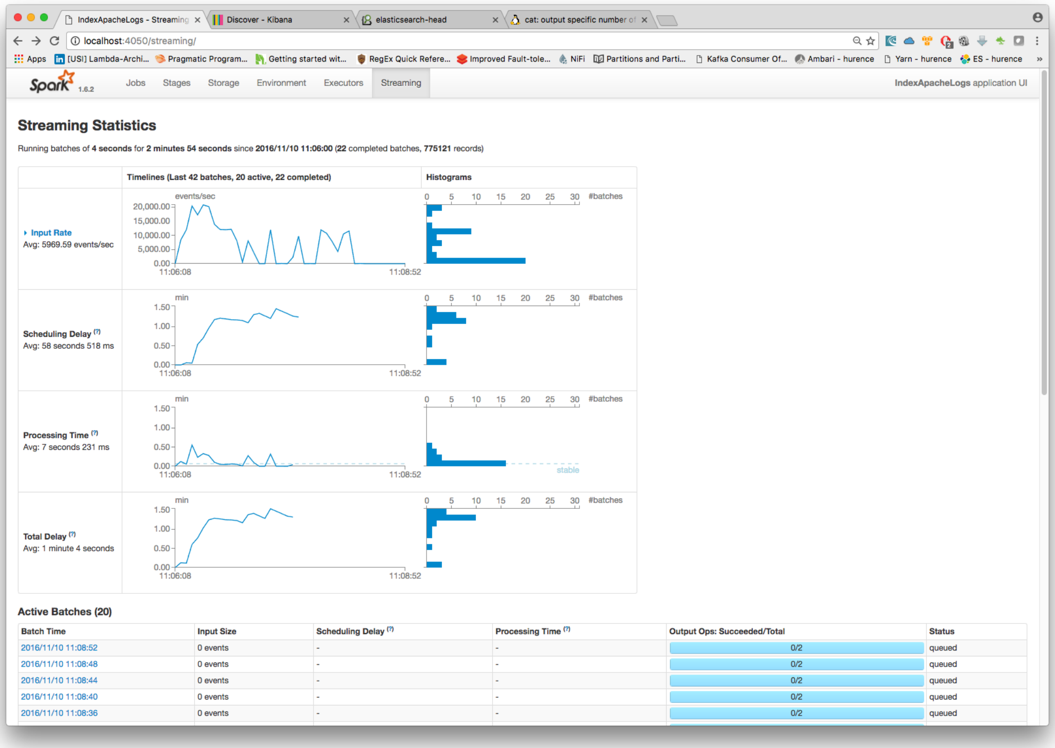
Generate Netflow events to port 2055 of localhost

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



Streaming Statistics

Running batches of **4 seconds** for **2 minutes 54 seconds** since **2016/11/10 11:06:00** (**22** completed batches, **775121** records)

	Timelines (Last 42 batches, 20 active, 22 completed)	Histograms
Input Rate Avg: 5969.59 events/sec		
Scheduling Delay (?) Avg: 58 seconds 518 ms		
Processing Time (?) Avg: 7 seconds 231 ms		
Total Delay (?) Avg: 1 minute 4 seconds		

Active Batches (20)

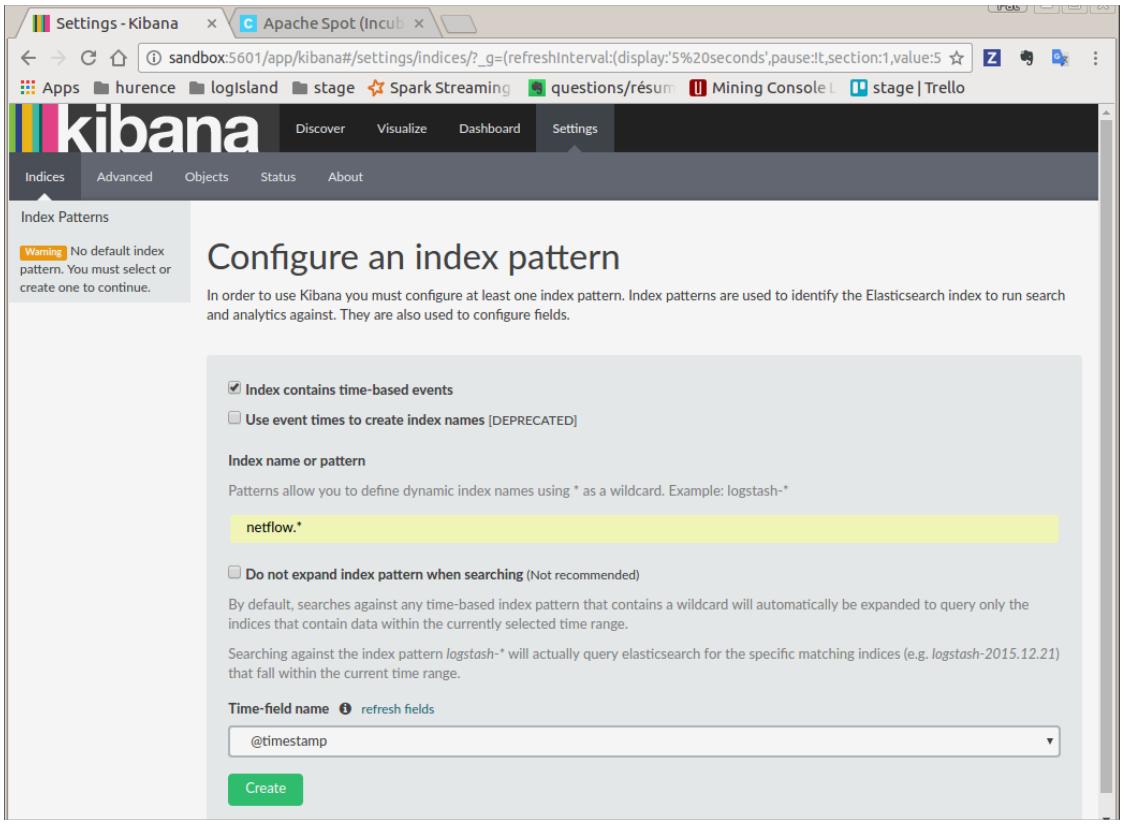
Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Output Ops: Succeeded/Total	Status
2016/11/10 11:08:52	0 events	-	-	0/2	queued
2016/11/10 11:08:48	0 events	-	-	0/2	queued
2016/11/10 11:08:44	0 events	-	-	0/2	queued
2016/11/10 11:08:40	0 events	-	-	0/2	queued
2016/11/10 11:08:36	0 events	-	-	0/2	queued

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.



Warning No default index pattern. You must select or create one to continue.

Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to identify the Elasticsearch index to run search and analytics against. They are also used to configure fields.

☑ Index contains time-based events

☐ Use event times to create index names [DEPRECATED]

Index name or pattern

Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*

netflow.*

☐ Do not expand index pattern when searching (Not recommended)

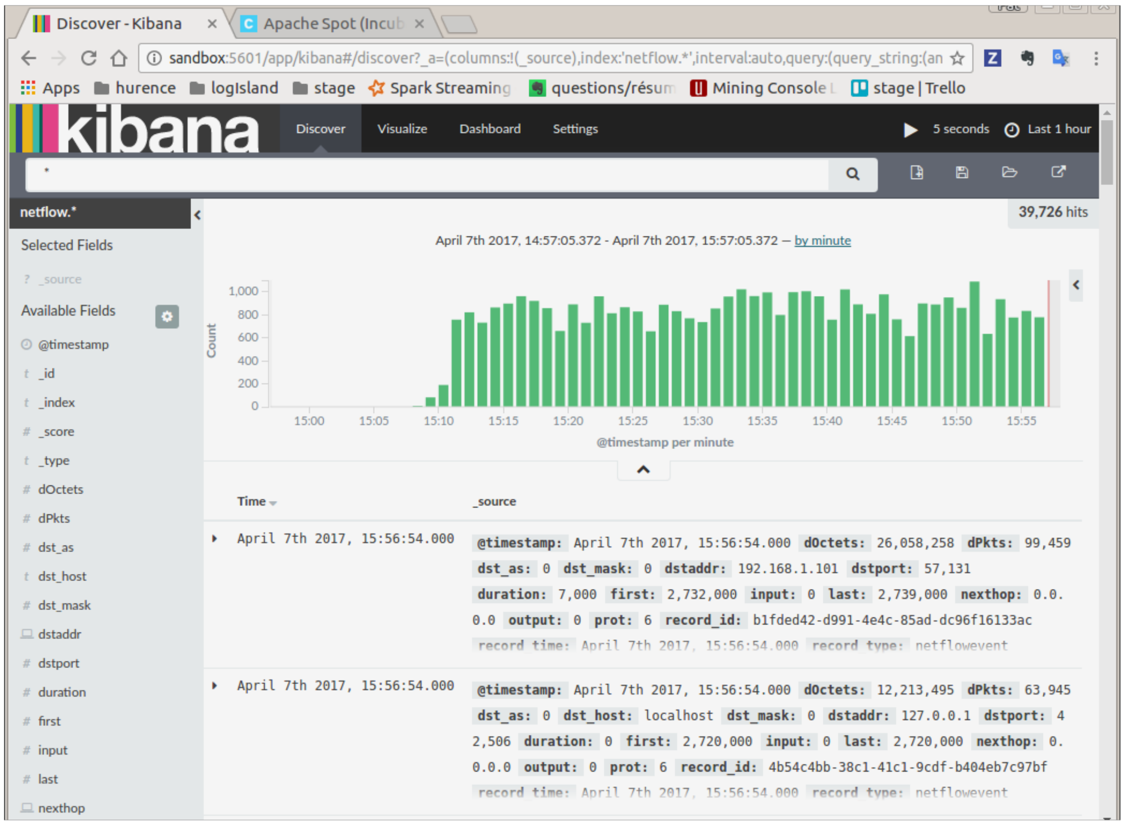
By default, searches against any time-based index pattern that contains a wildcard will automatically be expanded to query only the indices that contain data within the currently selected time range.

Searching against the index pattern *logstash-** will actually query elasticsearch for the specific matching indices (e.g. *logstash-2015.12.21*) that fall within the current time range.

Time-field name refresh fields

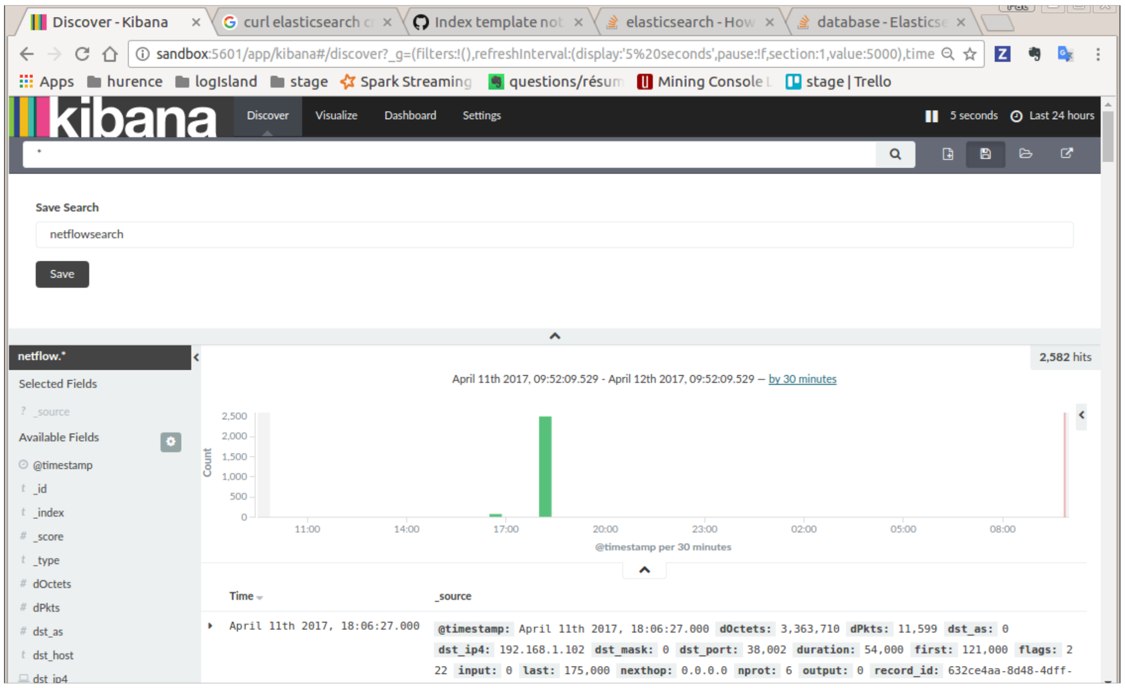
@timestamp

Then browse "Discover" tab, you should be able to explore your Netflow events.



```
April 7th 2017, 15:56:54.000   @timestamp: April 7th 2017, 15:56:54.000 dOctets: 26,058,258 dPkts: 99,459
                               dst_as: 0 dst_mask: 0 dstaddr: 192.168.1.101 dstport: 57,131
                               duration: 7,000 first: 2,732,000 input: 0 last: 2,739,000 nexthop: 0.0.
                               0.0 output: 0 prot: 6 record_id: b1fded42-d991-4e4c-85ad-dc96f16133ac
                               record_time: April 7th 2017, 15:56:54.000 record_type: netflowevent

April 7th 2017, 15:56:54.000   @timestamp: April 7th 2017, 15:56:54.000 dOctets: 12,213,495 dPkts: 63,945
                               dst_as: 0 dst_host: localhost dst_mask: 0 dstaddr: 127.0.0.1 dstport: 4
                               2,506 duration: 0 first: 2,720,000 input: 0 last: 2,720,000 nexthop: 0.
                               0.0.0 output: 0 prot: 6 record_id: 4b54c4bb-38c1-41c1-9cdf-b404eb7c97bf
                               record_time: April 7th 2017, 15:56:54.000 record_type: netflowevent
```

You have now to save your search by clicking the save icon. Save this search as “netflowsearch”

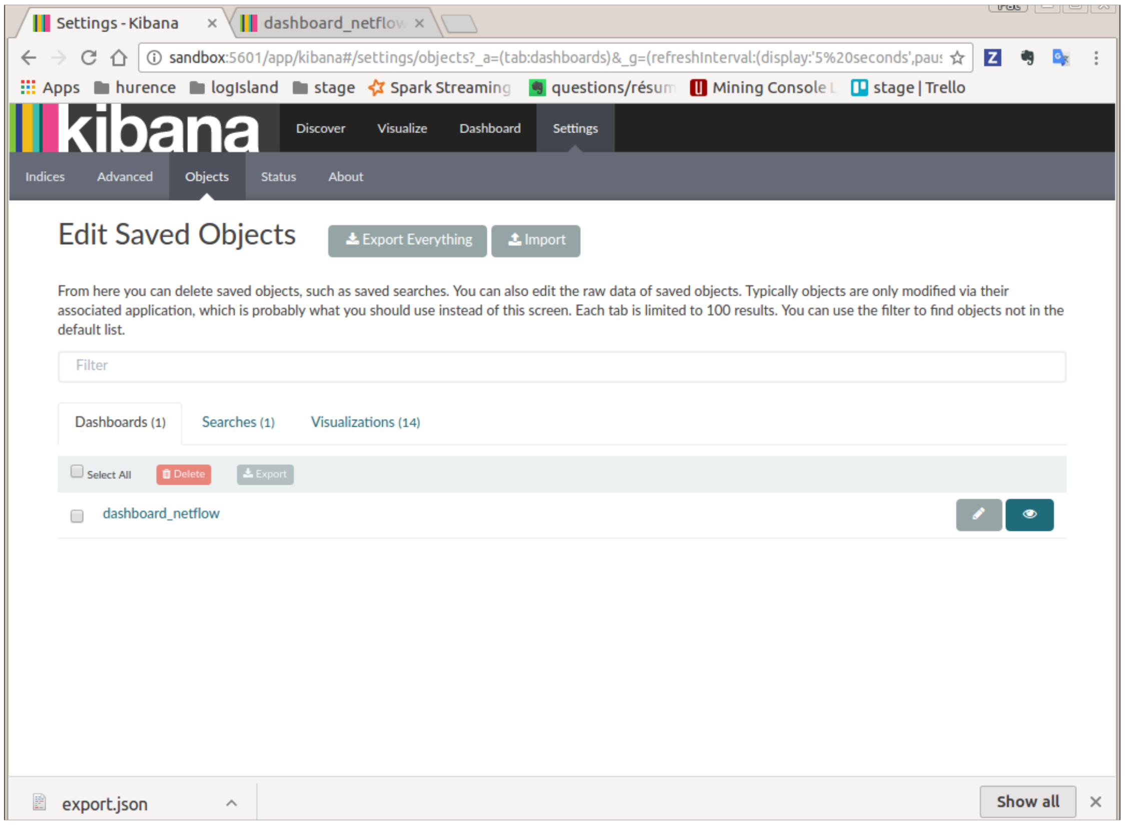


```
April 11th 2017, 18:06:27.000   @timestamp: April 11th 2017, 18:06:27.000 dOctets: 3,363,710 dPkts: 11,599 dst_as: 0
                                dst_ip4: 192.168.1.102 dst_mask: 0 dst_port: 38,002 duration: 54,000 first: 121,000 flags: 2
                                22 input: 0 last: 175,000 nexthop: 0.0.0.0 nprot: 6 output: 0 record_id: 632ce4aa-8d48-4dff-
```

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)

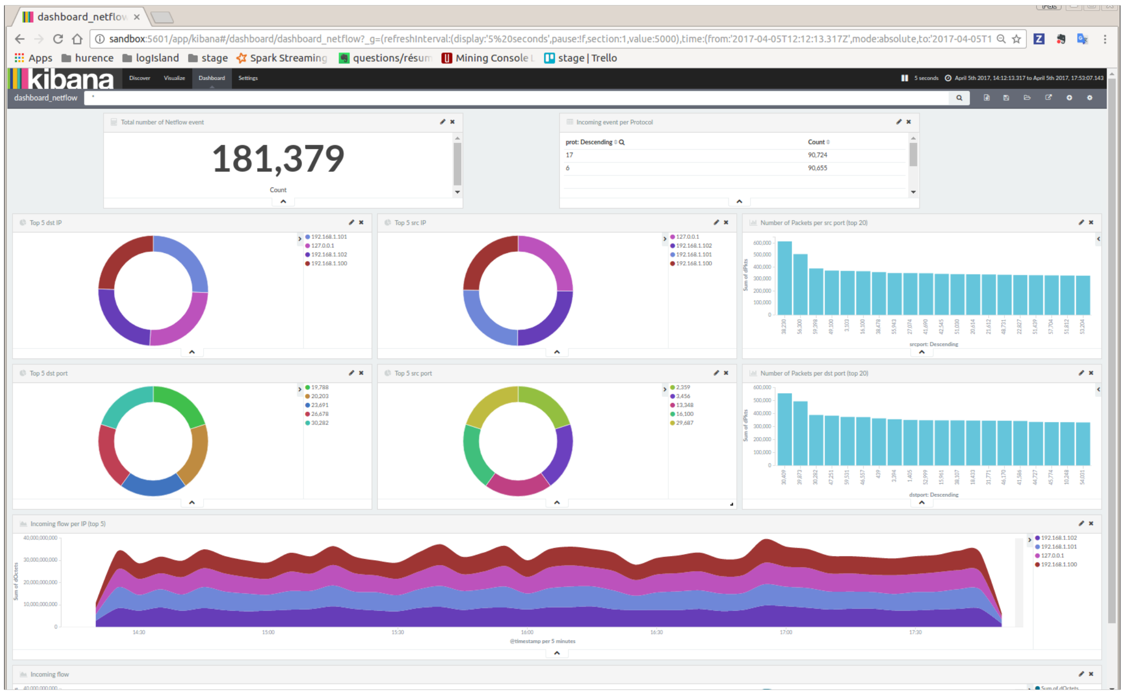


Edit Saved Objects

From here you can delete saved objects, such as saved searches. You can also edit the raw data of saved objects. Typically objects are only modified via their associated application, which is probably what you should use instead of this screen. Each tab is limited to 100 results. You can use the filter to find objects not in the default list.

dashboard_netflow

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)


dashboard_netflow
kibana
Discover
Visualize
Dashboard
Settings
Total number of Netflow event
181,379
Count
Incoming event per Protocol
Top 5 dst IP
Top 5 src IP
Number of Packets per src port (top 20)
Top 5 dst port
Top 5 src port
Number of Packets per dst port (top 20)
Incoming flow per IP (top 5)
Incoming flow

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
 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:
    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.

```
# 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
```

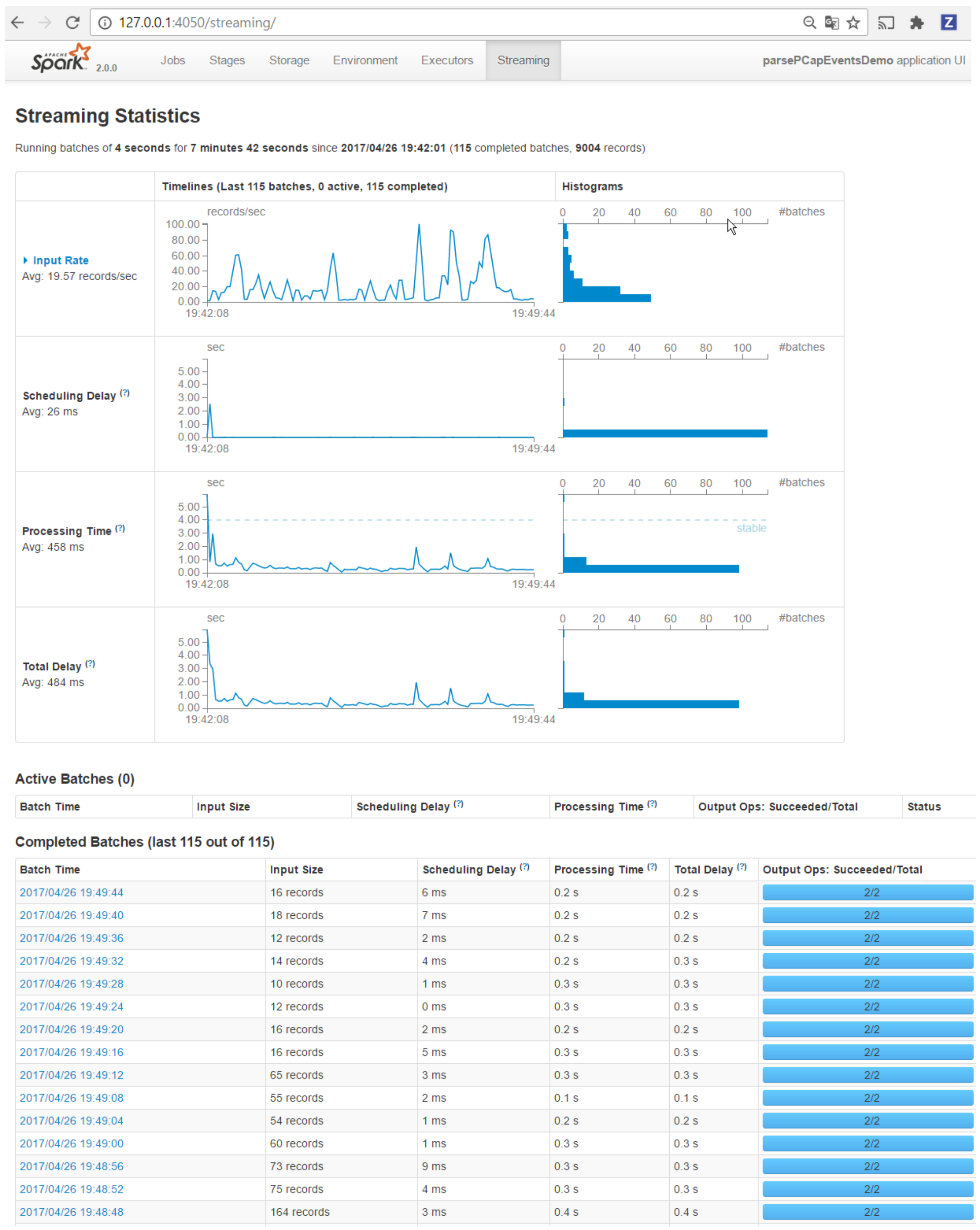
Capture network packets

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



Streaming Statistics

Running batches of **4 seconds** for **7 minutes 42 seconds** since **2017/04/26 19:42:01** (**115** completed batches, **9004** records)

	Timelines (Last 115 batches, 0 active, 115 completed)	Histograms
Input Rate Avg: 19.57 records/sec		
Scheduling Delay (?) Avg: 26 ms		
Processing Time (?) Avg: 458 ms		
Total Delay (?) Avg: 484 ms		

Active Batches (0)

Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Output Ops: Succeeded/Total	Status

Completed Batches (last 115 out of 115)

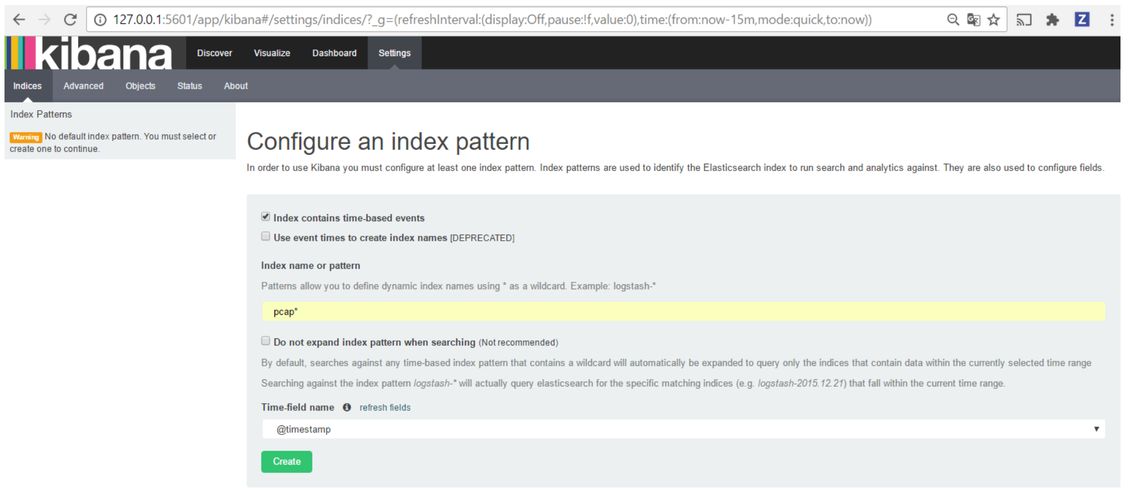
Batch Time	Input Size	Scheduling Delay (?)	Processing Time (?)	Total Delay (?)	Output Ops: Succeeded/Total
2017/04/26 19:49:44	16 records	6 ms	0.2 s	0.2 s	2/2
2017/04/26 19:49:40	18 records	7 ms	0.2 s	0.2 s	2/2
2017/04/26 19:49:36	12 records	2 ms	0.2 s	0.2 s	2/2
2017/04/26 19:49:32	14 records	4 ms	0.2 s	0.3 s	2/2
2017/04/26 19:49:28	10 records	1 ms	0.3 s	0.3 s	2/2
2017/04/26 19:49:24	12 records	0 ms	0.3 s	0.3 s	2/2
2017/04/26 19:49:20	16 records	2 ms	0.2 s	0.2 s	2/2
2017/04/26 19:49:16	16 records	5 ms	0.3 s	0.3 s	2/2
2017/04/26 19:49:12	65 records	3 ms	0.3 s	0.3 s	2/2
2017/04/26 19:49:08	55 records	2 ms	0.1 s	0.1 s	2/2
2017/04/26 19:49:04	54 records	1 ms	0.2 s	0.2 s	2/2
2017/04/26 19:49:00	60 records	1 ms	0.3 s	0.3 s	2/2
2017/04/26 19:48:56	73 records	9 ms	0.3 s	0.3 s	2/2
2017/04/26 19:48:52	75 records	4 ms	0.3 s	0.3 s	2/2
2017/04/26 19:48:48	164 records	3 ms	0.4 s	0.4 s	2/2

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.


127.0.0.1:5601/app/kibana#/settings/indices/?_g=(refreshInterval:(display:Off,pause:!f,value:0),time:(from:now-15m,mode:quick,to:now))
kibana
Discover
Visualize
Dashboard
Settings
Indices
Advanced
Objects
Status
About
Index Patterns

Warning No default index pattern. You must select or create one to continue.

Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to identify the Elasticsearch index to run search and analytics against. They are also used to configure fields.

[✓] Index contains time-based events

[ ] Use event times to create index names [DEPRECATED]

Index name or pattern

Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*

pcap*

[ ] Do not expand index pattern when searching (Not recommended)

By default, searches against any time-based index pattern that contains a wildcard will automatically be expanded to query only the indices that contain data within the currently selected time range

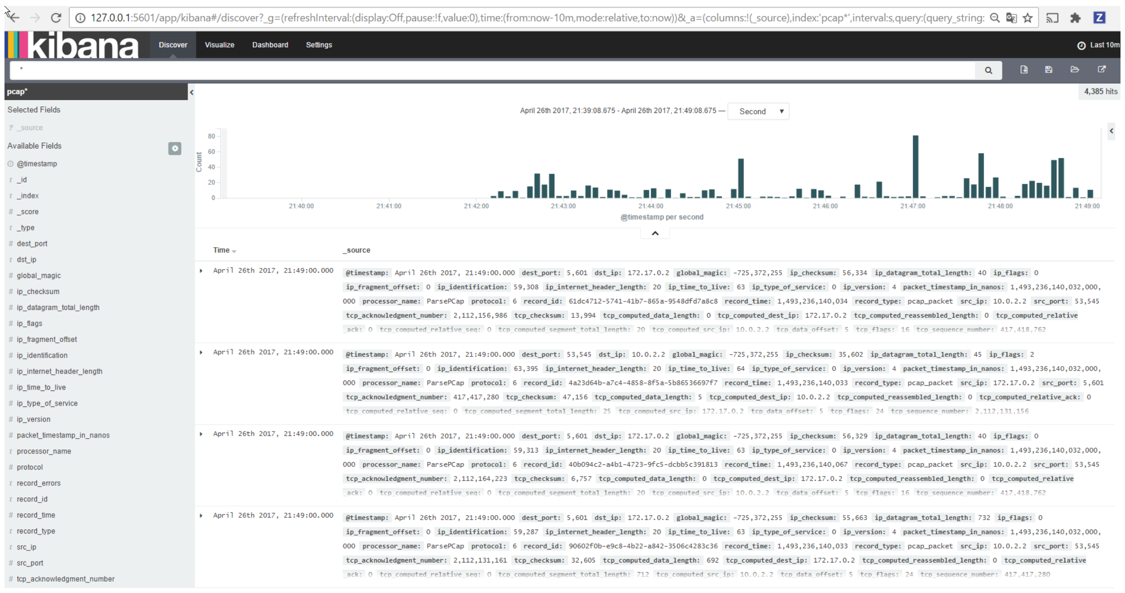
Searching against the index pattern *logstash-** will actually query elasticsearch for the specific matching indices (e.g. *logstash-2015.12.21*) that fall within the current time range.

Time-field name refresh fields

@timestamp

Create

Then browse "Discover" tab, you should be able to explore your network packet records :



```
April 26th 2017, 21:49:00.000  @timestamp: April 26th 2017, 21:49:00.000 dest_port: 5,601 dst_ip: 172.17.0.2 global_magic: -725,372,255 ip_checksum: 56,334 ip_datagram_total_length: 40 ip_flags: 0
                               ip_fragment_offset: 0 ip_identification: 59,308 ip_internet_header_length: 20 ip_time_to_live: 63 ip_type_of_service: 0 ip_version: 4 packet_timestamp_in_nanos: 1,493,236,140,032,000,
                               000 processor_name: ParsePCap protocol: 6 record_id: 61dc4712-5741-41b7-865a-9548dfd7a8c8 record_time: 1,493,236,140,034 record_type: pcap_packet src_ip: 10.0.2.2 src_port: 53,545
                               tcp_acknowledgment_number: 2,112,156,986 tcp_checksum: 13,994 tcp_computed_data_length: 0 tcp_computed_dest_ip: 172.17.0.2 tcp_computed_reassembled_length: 0 tcp_computed_relative
                               _ack: 0 tcp_computed_relative_seq: 0 tcp_computed_segment_total_length: 20 tcp_computed_src_ip: 10.0.2.2 tcp_data_offset: 5 tcp_flags: 16 tcp_sequence_number: 417,418,762
April 26th 2017, 21:49:00.000  @timestamp: April 26th 2017, 21:49:00.000 dest_port: 53,545 dst_ip: 10.0.2.2 global_magic: -725,372,255 ip_checksum: 35,602 ip_datagram_total_length: 45 ip_flags: 2
                               ip_fragment_offset: 0 ip_identification: 63,395 ip_internet_header_length: 20 ip_time_to_live: 64 ip_type_of_service: 0 ip_version: 4 packet_timestamp_in_nanos: 1,493,236,140,032,000,
                               000 processor_name: ParsePCap protocol: 6 record_id: 4a23d64b-a7c4-4858-8f5a-5b86536697f7 record_time: 1,493,236,140,033 record_type: pcap_packet src_ip: 172.17.0.2 src_port: 5,601
                               tcp_acknowledgment_number: 417,417,280 tcp_checksum: 47,156 tcp_computed_data_length: 5 tcp_computed_dest_ip: 10.0.2.2 tcp_computed_reassembled_length: 0 tcp_computed_relative_ack: 0
                               tcp_computed_relative_seq: 0 tcp_computed_segment_total_length: 25 tcp_computed_src_ip: 172.17.0.2 tcp_data_offset: 5 tcp_flags: 24 tcp_sequence_number: 2,112,131,156
April 26th 2017, 21:49:00.000  @timestamp: April 26th 2017, 21:49:00.000 dest_port: 5,601 dst_ip: 172.17.0.2 global_magic: -725,372,255 ip_checksum: 56,329 ip_datagram_total_length: 40 ip_flags: 0
                               ip_fragment_offset: 0 ip_identification: 59,313 ip_internet_header_length: 20 ip_time_to_live: 63 ip_type_of_service: 0 ip_version: 4 packet_timestamp_in_nanos: 1,493,236,140,032,000,
                               000 processor_name: ParsePCap protocol: 6 record_id: 40b094c2-a4b1-4723-9fc5-dcbb5c391813 record_time: 1,493,236,140,067 record_type: pcap_packet src_ip: 10.0.2.2 src_port: 53,545
                               tcp_acknowledgment_number: 2,112,164,223 tcp_checksum: 6,757 tcp_computed_data_length: 0 tcp_computed_dest_ip: 172.17.0.2 tcp_computed_reassembled_length: 0 tcp_computed_relative
                               _ack: 0 tcp_computed_relative_seq: 0 tcp_computed_segment_total_length: 20 tcp_computed_src_ip: 10.0.2.2 tcp_data_offset: 5 tcp_flags: 16 tcp_sequence_number: 417,418,762
April 26th 2017, 21:49:00.000  @timestamp: April 26th 2017, 21:49:00.000 dest_port: 5,601 dst_ip: 172.17.0.2 global_magic: -725,372,255 ip_checksum: 55,663 ip_datagram_total_length: 732 ip_flags: 0
                               ip_fragment_offset: 0 ip_identification: 59,287 ip_internet_header_length: 20 ip_time_to_live: 63 ip_type_of_service: 0 ip_version: 4 packet_timestamp_in_nanos: 1,493,236,140,032,000,
                               000 processor_name: ParsePCap protocol: 6 record_id: 90602f0b-e9c8-4b22-a842-3506c4283c36 record_time: 1,493,236,140,033 record_type: pcap_packet src_ip: 10.0.2.2 src_port: 53,545
                               tcp_acknowledgment_number: 2,112,131,161 tcp_checksum: 32,605 tcp_computed_data_length: 692 tcp_computed_dest_ip: 172.17.0.2 tcp_computed_reassembled_length: 0 tcp_computed_relative
                               _ack: 0 tcp_computed_relative_seq: 0 tcp_computed_segment_total_length: 712 tcp_computed_src_ip: 10.0.2.2 tcp_data_offset: 5 tcp_flags: 24 tcp_sequence_number: 417,417,280
```

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.

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);
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 ComponentContext 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 ComponentContext context,
                                  final Collection<Record> collection) {
    // log inputs
    collection.stream().forEach(record -> {
        logger.info("mock processing record : {}", record)
    });

    // output a useless record
    Record mockRecord = new Record("mock_record");
    mockRecord.setField("incomingEventsCount", FieldType.INT, collection.size());
    mockRecord.setStringField("message",
                               context.getProperty(FAKE_MESSAGE).asString());

    return Collections.singleton(mockRecord);
}
```

}

The runtime context : Instance

you can use your wonderful processor by setting its configuration and asking the `ComponentFactory` to give you one `ProcessorInstance` which is a `ConfiguredComponent`.

```
String message = "logisland rocks !";
Map<String, String> conf = new HashMap<>();
conf.put(MockProcessor.FAKE_MESSAGE.getName(), message );

ProcessorConfiguration componentConfiguration = new ProcessorConfiguration();
componentConfiguration.setComponent(MockProcessor.class.getName());
componentConfiguration.setType(ComponentType.PROCESSOR.toString());
componentConfiguration.setConfiguration(conf);

Optional<StandardProcessorInstance> instance =
    ComponentFactory.getProcessorInstance(componentConfiguration);
assertTrue(instance.isPresent());
```

Then you need a `ComponentContext` to run your processor.

```
ComponentContext context = new StandardComponentContext(instance.get());
Processor processor = instance.get().getProcessor();
```

And finally you can use it to process records

```
Record record = new Record("mock_record");
record.setId("record1");
record.setStringField("name", "tom");
List<Record> records =
    new ArrayList<>(processor.process(context, Collections.singleton(record)));

assertEquals(1, records.size());
assertTrue(records.get(0).hasField("message"));
assertEquals(message, records.get(0).getField("message").asString());
```

Chaining processors in a stream : RecordStream

Warning: @todo

Running the processor's flow : Engine

Warning: @todo

Packaging and conf

The end user of logisland is not the developer, but the business analyst which does understand any line of code. That's why we can deploy all our components 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
```

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);
// 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);
```

REST API

You can extend logisland with the Java high-level REST API as described below.

Design Tools

The REST API is designed with Swagger

You can use the docker image for the swagger-editor to edit the swagger yaml file and generate source code.

```
docker pull swaggerapi/swagger-editor
docker run -d -p 80:8080 swaggerapi/swagger-editor
```

If you're under mac you can setup swagger-codegen

```
brew install swagger-codegen

# or
wget https://oss.sonatype.org/content/repositories/releases/io/swagger/swagger-
↪codegen-cli/2.2.1/swagger-codegen-cli-2.2.1.jar
```

You can then start to generate the source code from the swgger yaml file

```
swagger-codegen generate \
    --group-id com.hurence.logisland \
    --artifact-id logisland-agent \
    --artifact-version 0.10.0-rc1 \
    --api-package com.hurence.logisland.agent.rest.api \
    --model-package com.hurence.logisland.agent.rest.model \
    -o logisland-framework/logisland-agent \
    -l jaxrs \
    --template-dir logisland-framework/logisland-agent/src/main/swagger/templates \
    -i logisland-framework/logisland-agent/src/main/swagger/api-swagger.yaml
```

Swagger Jetty server

This server was generated by the swagger-codegen project. By using the OpenAPI-Spec from a remote server, you can easily generate a server stub. This is an example of building a swagger-enabled JAX-RS server.

This example uses the JAX-RS framework.

To run the server, please execute the following:

```
cd logisland-framework/logisland-agent
mvn clean package jetty:run
```

You can then view the swagger.json .

> Note that if you have configured the *host* to be something other than localhost, the calls through swagger-ui will be directed to that host and not localhost!

Components

You'll find here the list of all usable Processors, Engines, Services and other components that can be usable out of the box in your analytics streams

---

BulkAddElasticsearch

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

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 1.1: 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		
default.index	The name of the index to insert into		null		**true**
default.type	The type of this document (used by Elasticsearch for indexing and searching)		null		**true**
timebased.index	do we add a date suffix	No date (no date added to default index), Today's date (today's date added to default index), yesterday's date (yesterday's date added to default index)	no		
es.index.field	the name of the event field containing es index name => will override index value if set		null		
es.type.field	the name of the event field containing es doc type => will override type value if set		null		

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)

Class

com.hurence.logisland.processor.consolidateSession.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 1.2: 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		
session.timeout	session timeout in sec		1800		
sessionid.field	the name of the field containing the session id => will override default value if set		sessionId		
timestamp.field	the name of the field containing the timestamp => will override default value if set		h2kTimestamp		
visitedpage.field	the name of the field containing the visited page => will override default value if set		location		
userid.field	the name of the field containing the userId => will override default value if set		userId		
fields.to.return	the list of fields to return		null		
firstVisitedPage.out.field	the name of the field containing the first visited page => will override default value if set		firstVisitedPage		
lastVisitedPage.out.field	the name of the field containing the last visited page => will override default value if set		lastVisitedPage		
isSessionActive.out.field	the name of the field stating whether the session is active or not => will override default value if set		is_sessionActive		
sessionDuration.out.field	the name of the field containing the session duration => will override default value if set		sessionDuration		
eventsCounter.out.field	the name of the field containing the session duration => will override default value if set		eventsCounter		
firstEventDateTime.out.field	the name of the field containing the date of the first event => will override default value if set		firstEventDateTime		
lastEventDateTime.out.field	the name of the field containing the date of the last event => will override default value if set		lastEventDateTime		
sessionInactivityDuration.out.field	the name of the field containing the session inactivity duration => will override default value if set		sessionInactivityDuration		

ConvertFieldsType

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

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 1.3: dynamic-properties

Name	Value	Description	EL
field	the new type	convert field value into new type	**true**

DebugStream

This is a processor that logs incoming records

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 1.4: allowable-values

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

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.

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 1.5: allowable-values

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

EnrichRecordsElasticsearch

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

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 .

Table 1.6: allowable-values

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

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

Class

com.hurence.logisland.processor.EvaluateJsonPath

Tags

JSON, evaluate, JsonPath

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 1.7: dynamic-properties

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

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.

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 .

FilterRecords

Keep only records based on a given field value

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 1.8: allowable-values

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

GenerateRandomRecord

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

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 1.9: allowable-values

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

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

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 1.10: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
numeric.fields	a comma separated string of numeric field to be matched		null		
output.record.type	the output type of the record		alert_match		
include.input.records	if set to true all the input records are copied to output		true		

Dynamic Properties

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

Table 1.11: dynamic-properties

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

ModifyId

modify id of records or generate it following defined rules

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 1.12: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
id.generation.strategy	the strategy to generate new Id	generate a random uid (generate a randomUid using java library), generate a hash from fields (generate a hash from fields), generate a string from java pattern and fields (generate a string from java pattern and fields), generate a concatenation of type, time and a hash from fields (generate a concatenation of type, time and a hash from fields (as for generate_hash strategy))	randomUuid		
fields.to.hash	the comma separated list of field names (e.g. : 'policyid,date_raw'		record_raw_value		
hash.charset	the charset to use to hash id string (e.g. 'UTF-8')		UTF-8		
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		
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		
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, vo, wa, wo, xh, yi, yo, za, zh, zu	en		

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

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 1.13: 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		
es.index.field	the name of the incoming records field containing es index name to use in multiget query.		null		
es.type.field	the name of the incoming records field containing es type name to use in multiget query		null		
es.ids.field	the name of the incoming records field containing es document Ids to use in multiget query		null		
es.includes.field	the name of the incoming records field containing es includes to use in multiget query		null		
es.excludes.field	the name of the incoming records field containing es excludes to use in multiget query		null		

NormalizeFields

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

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 1.14: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
conflict.resolution.policy	waht to do when a field with the same name already exists ?	nothing to do (leave record as it was), overwrite existing field (if field already exist), keep only old field and delete the other (keep only old field and delete the other), keep old field and new one (creates an alias for the new field)	do_nothing		

Dynamic Properties

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

Table 1.15: dynamic-properties

Name	Value	Description	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	**true**

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

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 1.16: 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		

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.

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 1.17: 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		
output.record.type	the output type of the record		netflow-event		
enrich.record	Enrich data. If enabledthe netflow record is enriched with inferred data		false		

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.

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 1.18: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
debug	Enable debug.		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		

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

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 1.19: allowable-values

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

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.

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 1.20: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
debug	Enable debug.		false		
cache.enabled	Enable caching. Caching to avoid to redo the same computation for many identical User-Agent strings.		true		
cache.size	Set the size of the cache.		1000		
user-agent.field	Must contain the name of the field that contains the User-Agent value in the incoming record.		null		
user-agent.keep	Defines if the field that contained the User-Agent must be kept or not in the resulting records.		true		
confidence.enabled	Enable confidence reporting. Each field will report a confidence attribute with a value comprised between 0 and 10000.		false		
ambiguity.enabled	Enable ambiguity reporting. Reports a count of ambiguities.		false		
fields	Defines the fields to be returned.		DeviceClass, DeviceName, DeviceBrand, DeviceCpu, DeviceFirmwareVersion, DeviceVersion, OperatingSystemClass, OperatingSystemName, OperatingSystemVersion, OperatingSystemNameVersion, OperatingSystemVersionBuild, LayoutEngineClass, LayoutEngineName, LayoutEngineVersion, LayoutEngineVersionMajor, LayoutEngineNameVersion, LayoutEngineNameVersionMajor, LayoutEngineBuild, AgentClass, AgentName, AgentVersion, AgentVersionMajor, AgentNameVersion, AgentNameVersionMajor, AgentBuild, AgentLanguage, AgentLanguageCode, AgentInformationEmail, AgentInformationUrl, AgentSecurity, AgentUuid, FacebookCarrier, FacebookDeviceClass, FacebookDeviceName, FacebookDeviceVersion, FacebookFBOP, FacebookFBSS, FacebookOperatingSystemName, FacebookOperatingSystemVersion, Anonymized, HackerAttackVector, HackerToolkit, KoboAffiliate, KoboPlatformId, IECompatibilityVersion, IECompatibilityVersionMajor, IECompatibilityNameVersion, IECompatibilityNameVersionMajor, __SyntaxError__, Carrier, GSAInstallationID, WebviewAppName,		

PutHBaseCell

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

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 1.21: 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		
table.name.field	The field containing the name of the HBase Table to put data into		null		**true**
row.identifier.field	Specifies field containing the Row ID to use when inserting data into HBase		null		**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		
column.family.field	The field containing the Column Family to use when inserting data into HBase		null		**true**
column.qualifier.field	The field containing the Column Qualifier to use when inserting data into HBase		null		**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		
record.schema	the avro schema definition for the Avro serialization		null		
record.serializer	the serializer needed to i/o the record in the HBase row	kryo serialization (serialize events as json blocs), json serialization (serialize events as json blocs), avro serialization (serialize events as avro blocs), no serialization (send events as bytes)	com.hurence.logisland.serializer.KryoSer		
table.name.default	The table table to use if table name field is not set		null		
column.family.default	The column family to use if column family field is not set		null		
column.qualifier.default	The column qualifier to use if column qualifier field is not set		null		

RemoveFields

Removes a list of fields defined by a comma separated list of field names

Class

com.hurence.logisland.processor.RemoveFields

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 1.22: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
fields.to.remove	the comma separated list of field names (e.g. 'policyid,date_raw'		null		

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.

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 1.23: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
script.code.imports	For inline mode only. This is the pyhton code that should hold the import statements if required.		null		
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		
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		
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		
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		
logisland.dependencies.path	The path to the directory containing the python dependencies shipped with logisland. You should not have to tune this parameter.		null		

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

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 1.24: allowable-values

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

SelectDistinctRecords

Keep only distinct records based on a given field

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 1.25: allowable-values

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

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.

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 1.26: allowable-values

Name	Description	Allowable Values	Default Value	Sensitive	EL
debug	Enable debug. If enabled, debug information are written to stdout.		false		
smtp.server	FQDN, hostname or IP address of the SMTP server to use.		null		
smtp.port	TCP port number of the SMTP server to use.		25		
smtp.security.username	SMTP username.		null		
smtp.security.password	SMTP password.		null		
smtp.security.ssl	Use SSL under SMTP or not (SMTPS). Default is false.		false		
mail.from.address	Valid mail sender email address.		null		
mail.from.name	Mail sender name.		null		
mail.bounce.address	Valid bounce email address (where error mail is sent if the mail is refused by the recipient server).		null		
mail.replyto.address	Reply to email address.		null		
mail.subject	Mail subject.		[LOGISLAND] Automatic email		
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		
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		
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		

SplitText

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

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 1.27: allowable-values

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

Dynamic Properties

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

Table 1.28: dynamic-properties

Name	Value	Description	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	**true**

See Also:

com.hurence.logisland.processor.SplitTextMultiline

SplitTextMultiline

No description provided.

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 1.29: allowable-values

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

SplitTextWithProperties

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

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 1.30: allowable-values

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

Dynamic Properties

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

Table 1.31: dynamic-properties

Name	Value	Description	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	**true**

See Also:

com.hurence.logisland.processor.SplitTextMultiline

What's new in logisland ?

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

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

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
-

Frequently Asked Questions.

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.

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.

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.

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

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
```

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 \
      -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
```

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/

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.

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