

***Programa de Experto en Big Data***

PROYECTO FINAL DE MÁSTER

***[ALB01]Análisis de eventos de logs en tiempo real***

**José Carlos García Serrano**

08/01/2014

ÍNDICE

[INTRODUCCIÓN 3](#_Toc408452845)

[OBJETIVOS 5](#_Toc408452846)

[PROBLEMÁTICA 6](#_Toc408452847)

[ARQUITECTURA 7](#_Toc408452848)

[MATERIAL Y TECNOLOGÍAS 10](#_Toc408452849)

[CONFIGURACIÓN 11](#_Toc408452850)

[IMPLEMENTACIÓN 15](#_Toc408452851)

[MONITORIZACIÓN 17](#_Toc408452852)

[VISUALIZACIÓN 20](#_Toc408452853)

[CONCLUSIONES 21](#_Toc408452854)

[MEJORAS 22](#_Toc408452855)

[AGRADECIMIENTOS 23](#_Toc408452856)

[BIBLIOGRAFÍA 24](#_Toc408452857)

# INTRODUCCIÓN

El proyecto final de máster del programa de Experto en Big Data que se desarrolla a continuación se basa en el Análisis de datos y Eventos de Logs en Tiempo Real.

En la actualidad todas las aplicaciones de todos los dispositivos tecnológicos generan grandes cantidades de eventos. Estos eventos tienen un volumen tan elevado que requiere de arquitecturas y tecnologías Big Data para poder procesarlos y analizarlos.

Dentro de esta problemática se desarrollará una arquitectura Big Data que será capaz de almacenar y analizar un flujo de datos en tiempo real para su posterior búsqueda y visualización de una forma rápida y eficiente. La solución debe de ser escalable en cada uno de sus elementos principales, de forma que no tenga puntos únicos de fallo.

El proyecto está dividido en varias fases, donde cada una forma parte del flujo de información y es una pieza clave en todo proyecto Big Data:

1. **Generación de datos sintéticos.**

Para poder realizar un sistema que analice datos en tiempo real se ha valorado la necesidad de crear un sistema que simule un flujo continuo de datos. En este caso en particular se simulan eventos de log de aplicaciones y servidores Web.

1. **Extracción de los datos.**

Cuando ya se ha definido un flujo de datos de entrada se deben de usar técnicas y herramientas que permitan la captación y el pre-procesamiento de dichos datos.

1. **Almacenamiento de la información.**

Todo el flujo de datos pre-procesado es necesario almacenarlo en un sistema en el que posteriormente se puedan realizar consultas y analíticas en tiempos razonables.

1. **Análisis y alertas en tiempo real.**

El flujo de datos debe de ser captado por una arquitectura que permita realizar un análisis rápido para poder extraer métricas y agregaciones que puedan ser monitorizadas en tiempo real. Por otro lado la arquitectura debe de ser capaz de emitir alertas cuando detecte patrones como errores o sobrecargas en las redes.

1. **Presentación de la información almacenada y del análisis.**

Todo el flujo de información debe de ser representado mediante una interfaz amigable que permita al usuario tanto monitorizar el canal de información como realizar búsquedas y filtros.

# OBJETIVOS

Se pretenden conseguir los siguientes objetivos:

* Crear una **arquitectura escalable** capaz procesar un flujo continuo de información y que permita realizar búsquedas sobre grandes volúmenes de datos.
* **Simulación de un canal de eventos** de logs que se desean monitorizar.
* Definir e implementar un **sistema de ingestión** de eventos de logs.
* Implementar un **sistema de encolado** de información en varias fases de la arquitectura para ser persistente a fallos en todo el conjunto.
* Implementar un sistema de **análisis de eventos** de log en tiempo real.
* Implementar un sistema de **alertas de eventos** de log en tiempo real.
* Definir e implementar un **sistema de almacenamiento** de toda la información y del análisis en tiempo real que sea escalable y permita búsquedas Full Text.
* Realizar un sistema de **visualización de los datos**, las analíticas y las búsquedas mediante Dashboards.

# PROBLEMÁTICA

En la actualidad se plantea una gran necesidad en cuanto al análisis y al almacenamiento de la información para grandes, medianas y pequeñas empresas de todos los ámbitos. Con frecuencia estas empresas poseen sistemas que no son capaces de escalar y cuando su flujo o cantidad de información crece, necesitan montar una arquitectura Big Data que resuelva el problema. Gracias a este proyecto que será publicado como **Open Source** pueden obtener una idea, un esqueleto o nociones básicas sobre cómo montar y configurar dicha arquitectura.

Esta solución está adaptada a un problema muy común dentro del ámbito de las IT, ya que es capaz de monitorizar los datos y eventos que las aplicaciones o dispositivos van generando día a día. eventos

# ARQUITECTURA

La arquitectura del proyecto se detalla en la siguiente ilustración, más adelante se explicará tanto el flujo de información como cada uno de los componentes del sistema:

* **Generación.**

**Databene** ofrece la posibilidad de crear programas con Eclipse en **Java** que permiten la generación de datos de forma aleatoria, distribuida y estructurada. Mediante un fichero XML configuramos los posibles campos y formatos.

La información generada se almacenará en RabbitMq mediante el mismo programa que genera los eventos.

* **Encolado.**

**RabbitMq** es un servidor de encolado de mensajes.

La ventaja principal de RabbitMq radica en que es persistente respecto a fallos en el sistema, además es escalable y está perfectamente integrado con los demás componentes de la arquitectura.

* **Administración.**

**LogStash** es una herramienta que permite la extracción y creación de flujos de logs, filtrado y pre-procesado de estos, para su posterior almacenaje.

Gracias a esta herramienta se pueden insertar directamente los datos sin tratar en ElasticSearch. Se generará automáticamente un índice por cada día de recogida de logs.

Por otro lado, se va a generar un flujo de todos los logs hacia una cola RabbitMq. Esta es la información que se va a procesar en real-time para sacar estadísticas por ventanas de tiempo.

* **Tratamiento y jobs.**

En **RabbitMq** se mantendrán activas tres colas, una dedicada a los datos de entrada, otra con los datos que van hacia el análisis en tiempo real y la última con los trabajos que necesitamos ejecutar y monitorizar en nuestro sistema en tiempo real.

Los jobs se generan mediante una aplicación **Java** que simularán ventanas de tiempo la analítica en tiempo real que se va ejecutando.

* **Alarmas y analítica en tiempo real.**

**Storm** es una herramienta de procesamiento en tiempo real que va a permitir el análisis y la definición de alertas para el canal o flujo de logs de entrada.

En Storm gracias a una topología con varios Spouts se pueden introducir los logs y los jobs localizados en RabbitMq. Los Bolts son los encargados de realizar los cálculos, el almacenamiento y la emisión de alertas.

Para el control y coordinación de todos estos componentes es necesaria la instalación y configuración de **ZooKeeper**.

La información intermedia que se genera y se pre-calcula se almacena en una Base de Datos **Redis** ya que almacena los datos en memoria y es tolerante respecto a fallos del sistema.

* **Almacenamiento.**

Todo el almacenamiento de datos en crudo, el resultado de pre-análisis y las alertas realizadas en tiempo real va a ser indexado en **ElasticSearch**.

Gracias a **ElasticSearch** se podrán realizar consultas avanzadas de los logs y buscar información en el contenido de estos en un sistema distribuido, escalable y con una alta velocidad de consulta.

* **Visualización.**

**Kibana** es la herramienta de visualización y consulta que se integra con ElasticSearch, gracias a ella se pueden generar dashboards donde se podrán realizar varias tareas importantes:

* Monitorizar el estado del clúster.
* Visualizar los datos analíticos pre-calculados
* Crear filtros y consultas avanzadas para explorar y agregar los datos.

# MATERIAL Y TECNOLOGÍAS

El material empleado para la realización del proyecto ha sido un ordenador personal portátil con las siguiente características:

* Procesador: Intel Core i7- 3632QM CPU 2.20GHz
* Memoria RAM: 16GB
* Sistema Operativo: Ubuntu 14.04 LTS

Las versiones de las distintas tecnologías y herramientas usadas para el funcionamiento del sistema han sido las siguientes:

* RABBITMQ 3.4.2
* ELASTICSEARCH 1.4.0
* LOGSTASH 1.4.2
* ZOOKEEPER 3.4.6
* STORM 0.9.3
* REDIS 2.8.19
* KIBANA 3.1.2
* Java JDK 1.7.0\_72
* Maven 3.1
* Apache httpd 2.4.10
* Eclipse Java EE IDE for Web Developers. Version: Kepler Service Release 2
* Git 1.9.1

# CONFIGURACIÓN

A continuación se van a dar algunas pautas de configuración de alguno de los componentes, además de la forma de ejecutar las distintas herramientas en una máquina Linux.

* **RABBITMQ.**

Para una rápida instalación se puede realizar mediante el comando apt-get:

wget http://www.rabbitmq.com/rabbitmq-signing-key-public.asc

sudo apt-key add rabbitmq-signing-key-public.asc

sudo apt-get install rabbitmq-server

Para arrancar el servicio con el comando:

sudo service rabbitmq-server start

Acceso a la WEB UI:

http://localhost:15672/

user:guest

pass:guest

* **ELASTICSEARCH.**

Las principales opciones de configuración para ejecutar ElasticSearch en local son las siguientes y se deben realizar en el fichero YML con el comando:

vim ~/elasticsearch/elasticsearch-1.4.0/config/elasticsearch.yml

cluster.name: elasticsearch-logs

node.name: "ES-logstash"

bootstrap.mlockall: true

network.host: 127.0.0.1

transport.tcp.port: 9309

http.port: 9201

http.enabled: true

http.cors.enabled: true

http.cors.allow-origin: "/.\*/"

discovery.zen.ping.unicast.hosts: ["127.0.0.1"]

discovery.zen.ping.multicast.enabled: false

Para iniciar ElasticSearch:

~/elasticsearch/elasticsearch-1.4.0/bin/elasticsearch

Un aspecto importante de cara a controlar el estado de los Índices creados es instalar plugins creados por la comunidad:

~/elasticsearch/elasticsearch-1.4.0/bin/plugin -install lmenezes/elasticsearch-kopf

~/elasticsearch/elasticsearch-1.4.0/bin/plugin -install mobz/elasticsearch-head

~/elasticsearch/elasticsearch-1.4.0/bin/plugin -install lukas-vlcek/bigdesk

Podemos acceder a ellos con las siguientes direcciones:

http://127.0.0.1:9201/\_plugin/head/

http://127.0.0.1:9201/\_plugin/bigdesk/#nodes

http://127.0.0.1:9201/\_plugin/kopf

* **LOGSTASH.**

Para iniciar LogStash se necesita editar el siguiente fichero de configuración, determinamos cuáles son sus datos de entrada y de salida, en este caso su entrada es una cola de RabbitMq y sus salidas son ElasticSearch y otra cola de RabbitMq:

vim ~/logstash/logstash-1.4.2/bin/logstash-simple.conf

input {

rabbitmq {

host => "localhost"

queue => "logsQueue"

exchange => "logsExchange"

key => "logsRoute"

durable => true

}}

output {

elasticsearch {

host => "127.0.0.1"

port => "9309"

cluster => "elasticsearch-logs"

node\_name => "ES-logstash"

}

rabbitmq {

exchange => "stormExchange"

host => "localhost"

exchange\_type => "direct"

key => "stormRoute"

durable => true

}}

Para iniciar el agente de LogStash tenemos que ejecutar el siguiente comando:

~/logstash/logstash-1.4.2/bin/logstash agent -f ~/logstash/logstash-1.4.2/bin/logstash-simple.conf

* **ZOOKEEPER.**

El fichero de configuración que se debe editar es el siguiente, así como su contenido:

vim ~/zookeeper/zookeeper-3.4.6/conf/zoo.cfg

tickTime=2000

initLimit=10

syncLimit=5

dataDir=/tmp/zookeeper

clientPort=2181

Para iniciar Zookeeper basta con ejecutar el siguiente comando:

sudo ~/zookeeper/zookeeper-3.4.6/bin/zkServer.sh start

* **STORM.**

El fichero de configuración y su contenido es el siguiente:

vim ~/storm/apache-storm-0.9.3/conf/storm.yaml

storm.zookeeper.servers:

- "localhost"

nimbus.host: "localhost"

Para iniciar Storm se deben de iniciar tres demonios con los comandos:

~/storm/apache-storm-0.9.3/bin/storm supervisor

~/storm/apache-storm-0.9.3/bin/storm nimbus

~/storm/apache-storm-0.9.3/bin/storm ui

Accediendo a la siguiente direccion Web tenemos acceso a una herramienta de visualizacion de estadisticas y metricas de los miebros del cluster que ejecutan el sistema de analisis en tiempo real:

http://127.0.0.1:8080/index.html

* **EJECUCIONES JAVA.**
* Topología Storm.

storm jar ~/workspace/logs-storm/target/logs-storm-0.0.1-SNAPSHOT-jar-with-dependencies.jar project.ebd.LogsTopology -cluster -application 'WEB SERVER'

* Generador de eventos.

java jar logs-generator-1.0-SNAPSHOT.jar

* Generador de trabajos.

java jar ~/workspace/jobs-generator/target/jobs-generator-1.0-SNAPSHOT.jar project.ebd.JobGenerator -debug -application 'WEB\_SERVER,LOGGIN\_SERVER'

* **KIBANA.**

En primer se debe de instalar un servidor http donde pueda ejecutarse Kibana, con los siguientes comandos conseguimos arrancarlo y configurarlo:

sudo apt-get install apache2

sudo mkdir /var/www/html/kibana

sudo cp -R ~/kibana/kibana-3.1.2/\* /var/www/html/kibana

sudo vim /var/www/html/kibana/config.js

elasticsearch: "http://localhost:9201"

sudo vim /etc/apache2/sites-enabled/000-default.conf

DocumentRoot /var/www/html/kibana

sudo vim /etc/apache2/apache2.conf

<Directory /var/www/html/kibana>

Options Indexes FollowSymLinks

AllowOverride None

Require all granted

</Directory>

sudo service apache2 restart

# IMPLEMENTACIÓN

Para llevar a cabo la implementación del proyecto se han elaborado tres programas en Java, se han configurado cada una de las herramientas necesarias y se han elaborado dos dashboards para la visualización.

A continuación se detalla la funcionalidad de cada uno de los programas:

* **logs-generator**

Este programa desarrollado en Java es el encargado de generar datos sintéticos, gracias a una librería externa llamada Benerator que se configura mediante un fichero XML (benerator.xml) se pueden generar datos tanto de entidades como países, direcciones, personas, compañías, etc.. como de datos relativos a valores numéricos, cadenas, etc.. haciendo distribuciones de probabilidad tanto Random, como Gausianas o distribuidas.

La finalidad de este programa es insertar los datos generados en un formato JSON en la cola que se crea si no existe en RabbitMq. La cola encargada de gestionar estos datos de eventos se llama logsQueue.

La mayor dificultad que se afronta en este programa es la investigación de las bondades de Benerator y la generación de datos que se adapten a lo que se desea analizar en el resto de los componentes del sistema.

* **logs-storm**

Este programa desarrollado en Java es el encargado de crear y ejecutar una topología que sea capaz de procesar datos leídos de una cola RabbitMq, realizando un cálculo de estadísticas como son máximos, mínimos o medias, aceptando como parámetros de entrada la monitorización de determinadas aplicaciones. De esta forma es capaz de generar datos generales y datos agregados para una determinada entidad.

A su misma vez tiene que detectar si se ha alcanzado una cantidad determinada de errores para poder generar alertas a una cuenta de email donde se van notificando estas incidencias.

Para poder llevar a cabo todo el cálculo y conteo de la información que se va recibiendo, los distintos componentes del sistema hacen uso de un BBDD Redis que almacena Hashes con dicha información.

Para poder crear ventanas de tiempo en las que se van calculando las distintas métricas, se plantearon dos soluciones, una era usar DRPCs y la otra hacer uso de los sistemas que tenemos en la arquitectura, se optó por la segunda opción, de forma que se leen trabajos a ejecutar de una cola de trabajos de RabbitMq.

Este programa se encarga de volcar los resultados en ElasticSearch, de forma que hace uso del driver nativo para Java de esta herramienta.

Las mayores dificultades que se han afrontado en la realización de este programa han sido la comprensión en detalle de Storm, más allá de lo realizado en las sesiones de clase. Esto abarca por ejemplo la asincronía de lectura, borrado e inserción en Redis de los distintos componentes, la creación de Streams y flujos de datos en topologías avanzadas y la configuración de parámetros que sin ellos se pierden mensajes, Dichos parámetros aumentan el tamaño de los buffers de entrada y salida de los distintos componentes de la topología, son los siguientes para nuestra aplicación:

conf.put(Config.TOPOLOGY\_RECEIVER\_BUFFER\_SIZE, 16);

conf.put(Config.TOPOLOGY\_TRANSFER\_BUFFER\_SIZE, 32);

conf.put(Config.TOPOLOGY\_EXECUTOR\_RECEIVE\_BUFFER\_SIZE, 16384);

conf.put(Config.TOPOLOGY\_EXECUTOR\_SEND\_BUFFER\_SIZE, 16384);

En la siguiente ilustración podemos ver la topología que se ha implementado:

* **jobs-generator**

Este programa implementado en Java tiene una estructura básica, acepta como parámetros de entrada la ventana de tiempo en la que se desean que se envíen trabajos a RabbitMq, los cuales son recogidos por Storm para volcar las métricas calculadas hasta ese momento.

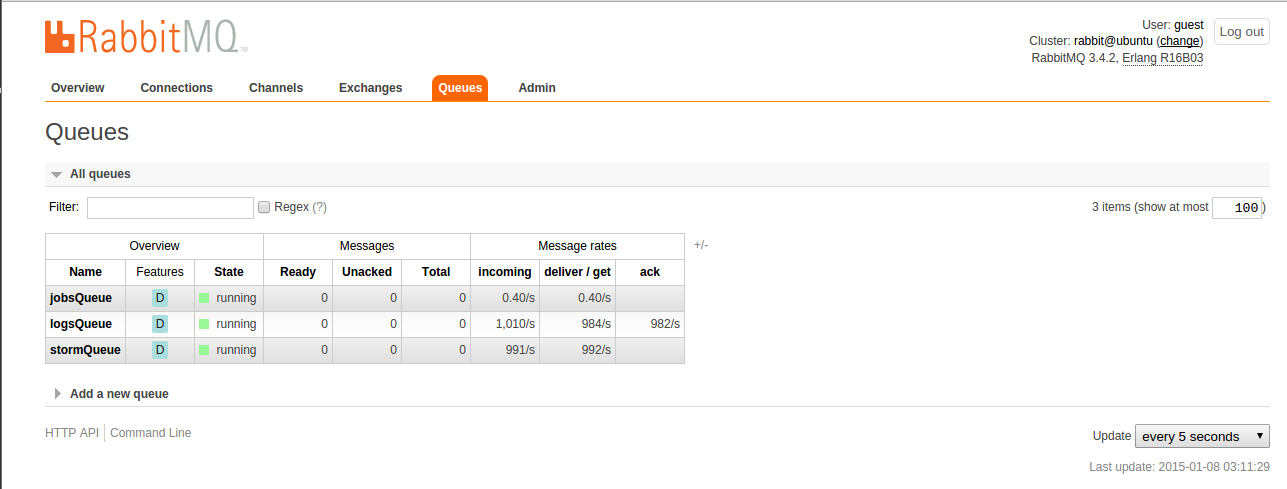
Como parámetro de entrada acepta los valores de la entidad que se desean monitorizar y almacenar en ElasticSearch para su visualización.

# MONITORIZACIÓN Y PRUEBAS

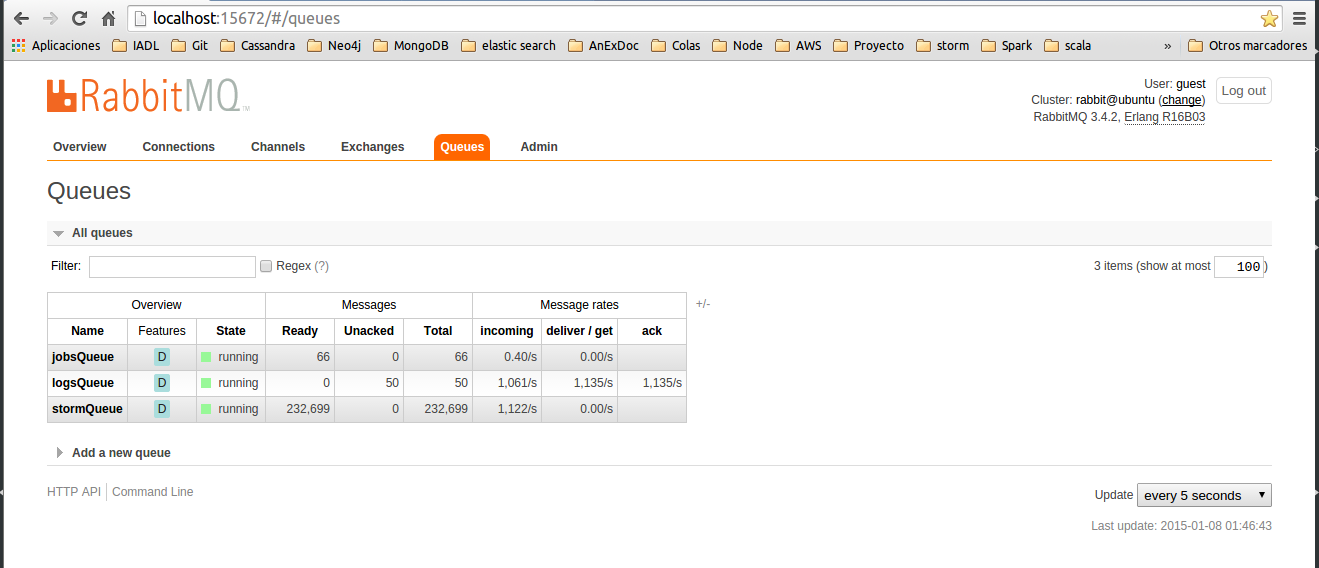
Los distintos componentes de la arquitectura tienen Web UI para ver el estado de la ejecución, a continuación se muestran algunas capturas de pantalla con los distintos estados de la ejecución:

* **RABBITMQ.**

En la primera ilustración se puede ver el estado de las colas en plena ejecución de todo el sistema, como es de esperar, nunca hay mensajes acumulados si todos trabajan simultáneamente:



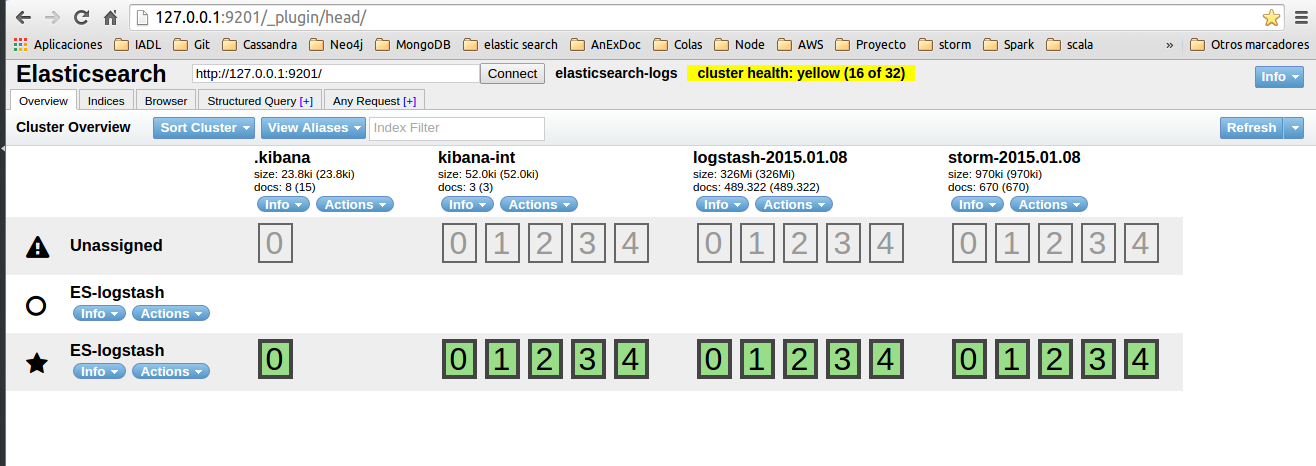
En esta segunda ilustración se puede observar el estado de las colas si está funcionando LogStash y ElasticSearch pero no Storm:



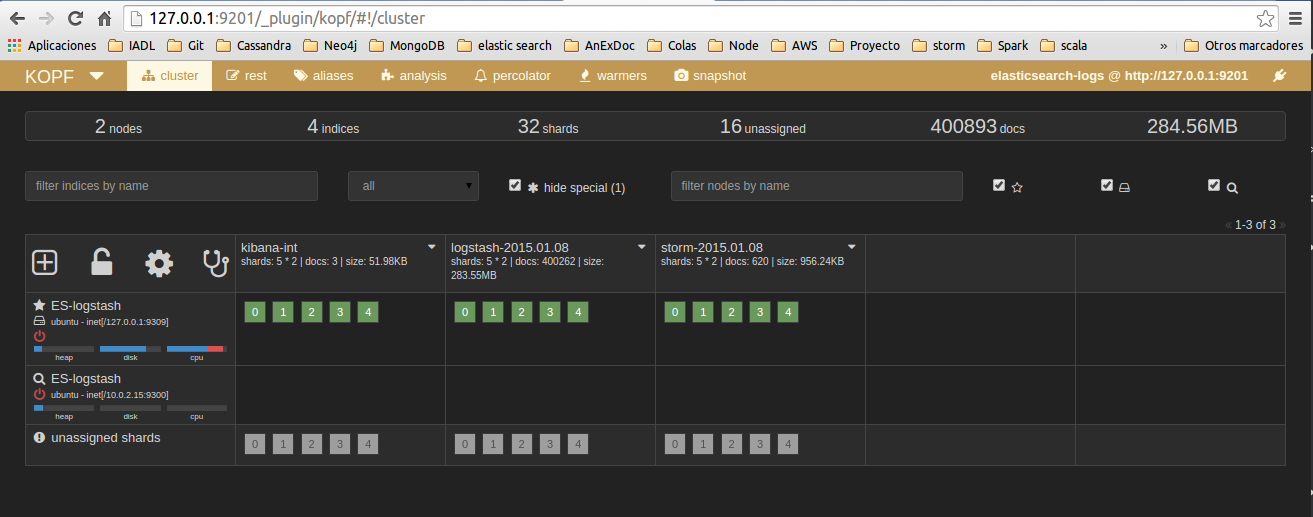
* **ELASTICSEARCH.**

ElasticSearch tiene una gran cantidad de plugins para monitorizar los indices y el estado del cluster, dos de los mas comunes de software libre son Head y Kopf.

* Head:

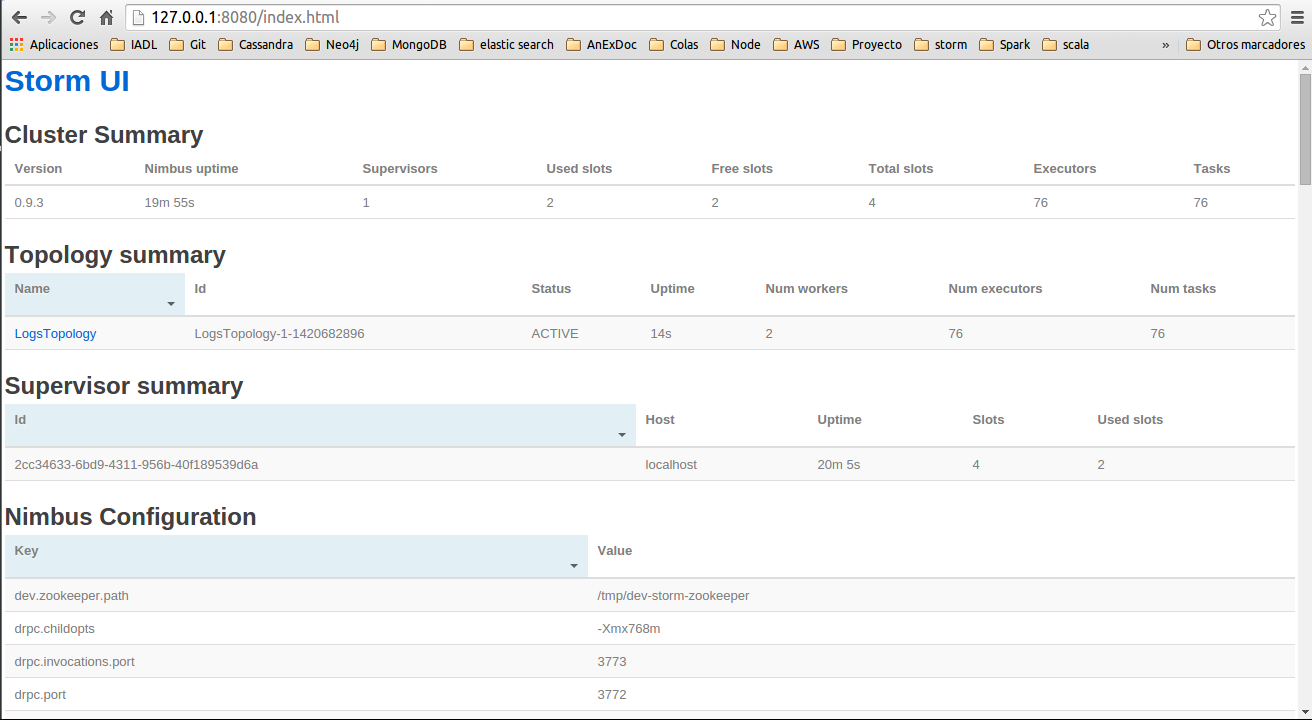


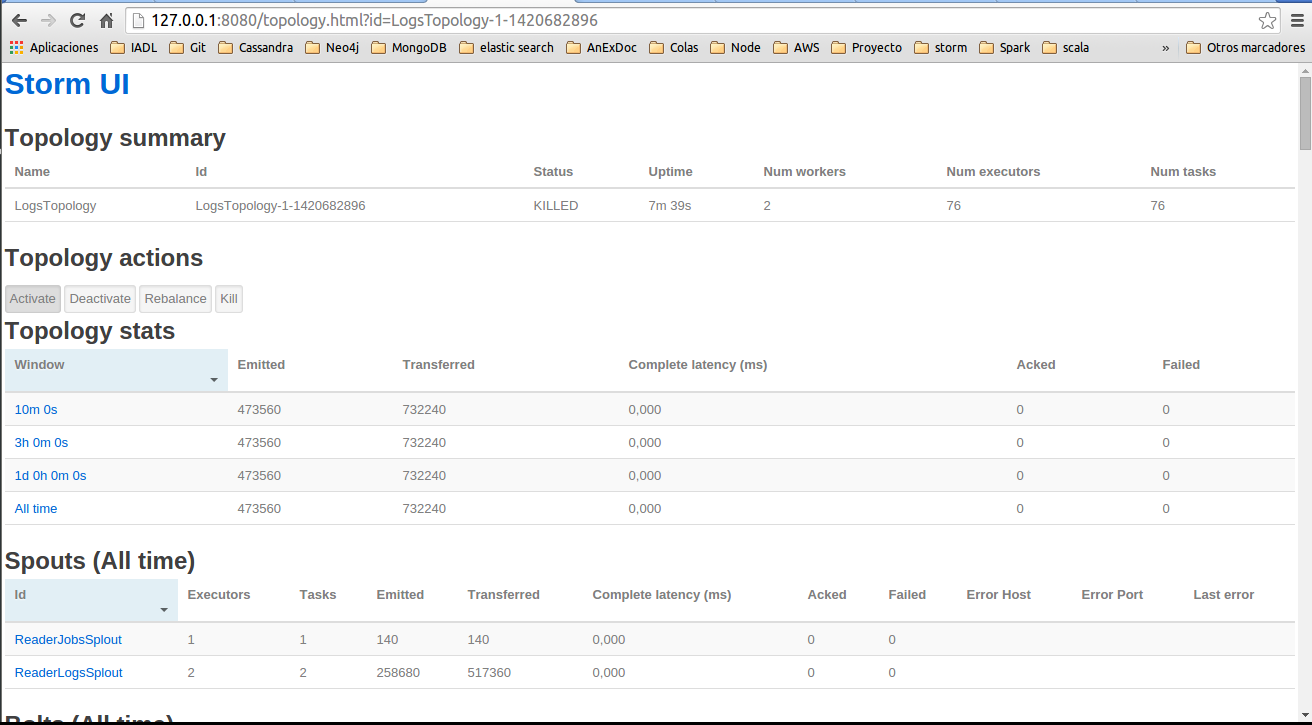
* Kopf:



* **STORM.**

Storm nos ofrece la posibilidad de visualizar estadísticas del clúster y las topologías en ejecución, veamos dos ilustraciones con la ejecución del programa de analítica en tiempo real:





Todo el sistema ejecutándose simultáneamente ha funcionado correctamente.

El generador de datos, es capaz de generar cerca de 2000 eventos por segundo, mientras que la topología ha sido capaz de llegar a consumir 4000 eventos por segundo de su cola correspondiente de RabbitMq.

Por otro lado LogStash ha cumplido las expectativas porque realiza tres funciones importantes, ya que lee y escribe resultados en dos lugares diferentes.

La visualización, monitorización y búsquedas en Kibana sobre los datos almacenados en ElasticSearch ha sido perfecta, siempre y cuando no esté todo el sistema funcionando simultáneamente, ya que el consumo de memoria es muy elevado. Todas las métricas y gráficos se refrescan en milisegundos como era de esperar.

# VISUALIZACIÓN

# CONCLUSIONES

# MEJORAS

# AGRADECIMIENTOS

# BIBLIOGRAFÍA