**Tables Contains:**

1 Spring Boot 2.0

1.0 Logstash

1.0 Elastic search

1.2Kibana Configuration

2 Spring Boot Dev Tools Features

1.0 Property Defaults

1.1 Automatic Restart

1.2 Live Reload

1.3 Remote Debug Tunnelling

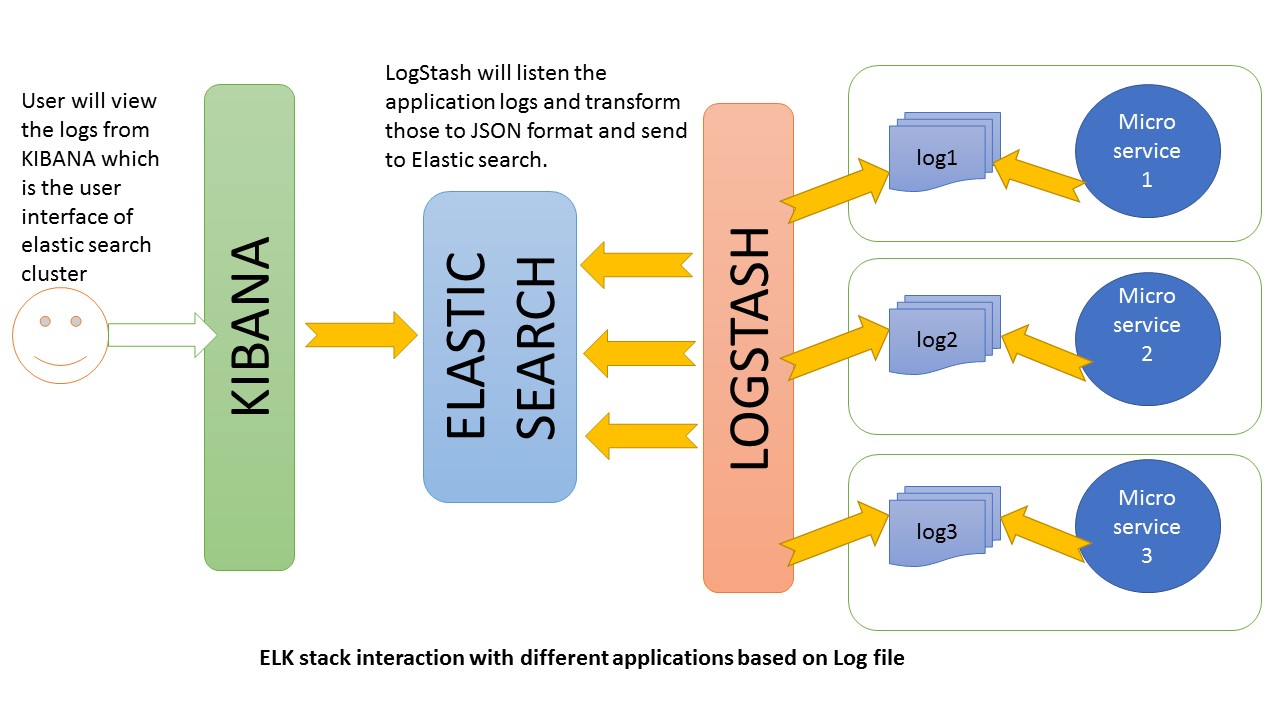
1.4 Remote Update and Restart

**1. What is ELK Stack**

1. **Elasticsearch** is a distributed, JSON-based *search and analytics engine* designed for horizontal scalability, maximum reliability, and easy management.
2. **Logstash** is a dynamic *data collection pipeline* with an extensible plugin ecosystem and strong Elasticsearch synergy.
3. **Kibana** gives the *visualization* of data through a UI.

**1.1. ELK Stack Architecture**

Logstash processes the application log files based on the filter criteria we set and sends those logs to Elasticsearch. Through Kibana, we view and analyze those logs when required.

[](https://howtodoinjava.com/wp-content/uploads/2017/08/ELK.jpg)ELK In Action

**2. ELK stack configuration**

All these three tools are based on JVM and before start installing them, please verify that JDK has been properly configured. Check that standard JDK 1.8 installation, JAVA\_HOME and PATH set up is already done.

**2.1. Elasticsearch**

* Download latest version of Elasticsearch from this [download page](https://www.elastic.co/downloads/elasticsearch) and unzip it any folder.
* Run bin\elasticsearch.bat from command prompt.
* By default, it would start at [http://localhost:9200](http://localhost:9200/)

**2.2. Kibana**

* Download the latest distribution from [download page](https://www.elastic.co/downloads/kibana) and unzip into any folder.
* Open config/kibana.yml in an editor and set elasticsearch.url to point at your Elasticsearch instance. In our case as we will use the local instance just uncomment elasticsearch.url: "http://localhost:9200"
* Run bin\kibana.bat from command prompt.
* Once started successfully, Kibana will start on default port 5601 and Kibana UI will be available at [http://localhost:5601](http://localhost:5601/)

**2.3. Logstash**

* Download the latest distribution from [download page](https://www.elastic.co/downloads/logstash) and unzip into any folder.
* Create one file logstash.conf as per [configuration instructions](https://www.elastic.co/guide/en/logstash/current/configuration.html). We will again come to this point during actual demo time for exact configuration.

Now run bin/logstash -f logstash.conf to start logstash

ELK stack is not up and running. Now we need to create few microservices and point logstash to the API log path.

**3. ELK stack example – Create Microservice**

**3.1. Create Spring Boot Project**

Let’s [create an application using spring boot](https://howtodoinjava.com/spring/spring-boot/spring-boot-tutorial-with-hello-world-example/) for faster development time. Follow those steps to start this service.

**3.2. Add REST Endpoints**

Add one RestController class which will expose few endpoints like http://localhost:1198/SmsSender/byid1/10 Actually we are going to test few log statements only, so feel free to add/modify logs as per your choice.

@RestController

@RequestMapping("/SmsSender")

public class SmsSender {

private static final Logger LOG = Logger.getLogger(SmsSender.class.getName());

@Autowired

Producer producer;

@Autowired

NokiasubServices nokiasubServices;

@GetMapping(value="/send/{msg}")

public String sendMsg(@PathVariable(name="msg") String msg){

producer.produceMsg(msg);

LOG.log(Level.INFO, msg);

return "Done ";

}

@PostMapping(value="/publish" )

public String save(@RequestBody NokiaSubs nokiaSubs)

{

producer.produceMsg2(nokiaSubs.toString());

LOG.log(Level.INFO, nokiaSubs);

nokiasubServices.save(nokiaSubs); /// db

System.out.println("message has store in mongodatabase ..."+nokiaSubs);

return "message has successfully store!.........";

**}3.3. Configure Spring boot Logging**

Open application.properties under resources folder and add below configuration entries.

|  |
| --- |
| logging.file=c:/sbusbriberlog.text  logging.file=c:/producer.text  spring.application.name = SDL producer  spring.application.name = SDL consumer |

**3.4. Verify Microservice Generated Logs**

Do a final maven build using mvn clean install and start the application using command java -jar target\elk-example-spring-boot-0.0.1-SNAPSHOT.jar and test by browsing http://localhost:1198/SmsSender/byid1/10

Go to the application root directory and verify that the log file i.e. elk-example.log has been created and do a couple of visits to the endpoints and verify that logs are getting added in the log file.

**4. Logstash Configuration**

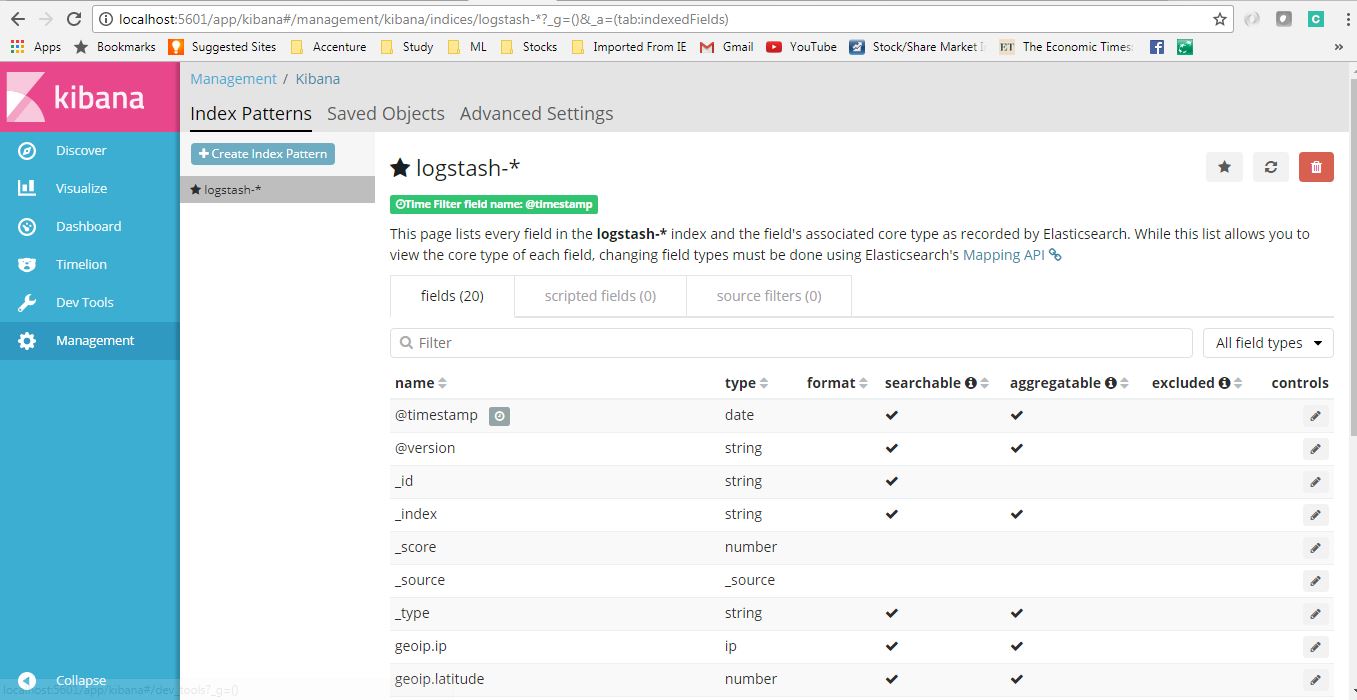
We need to create a logstash configuration file so that it listen to the log file and push log messages to elastic search. Here is the logstash [configuration](https://www.elastic.co/guide/en/logstash/current/configuration.html) used in the example, please change the log path as per your setup.

|  |
| --- |
| input {    file {      type => "java"      path => "c:/subscriberlog.txt"      codec => multiline {        pattern => "^%{YEAR}-%{MONTHNUM}-%{MONTHDAY} %{TIME}.\*"        negate => "true"        what => "previous"      }    }  }    filter {    #If log line contains tab character followed by 'at' then we will tag that entry as stacktrace    if [message] =~ "\tat" {      grok {        match => ["message", "^(\tat)"]        add\_tag => ["stacktrace"]      }    }     grok {      match => [ "message",                 "(?<timestamp>%{YEAR}-%{MONTHNUM}-%{MONTHDAY} %{TIME})  %{LOGLEVEL:level} %{NUMBER:pid} --- \[(?<thread>[A-Za-z0-9-]+)\] [A-Za-z0-9.]\*\.(?<class>[A-Za-z0-9#\_]+)\s\*:\s+(?<logmessage>.\*)",                 "message",                 "(?<timestamp>%{YEAR}-%{MONTHNUM}-%{MONTHDAY} %{TIME})  %{LOGLEVEL:level} %{NUMBER:pid} --- .+? :\s+(?<logmessage>.\*)"               ]    }        date {      match => [ "timestamp" , "yyyy-MM-dd HH:mm:ss.SSS" ]    }  }    output {      stdout {      codec => rubydebug    }      # Sending properly parsed log events to elasticsearch    elasticsearch {      hosts => ["localhost:9200"]    }  } |

**5. Kibana Configuration**

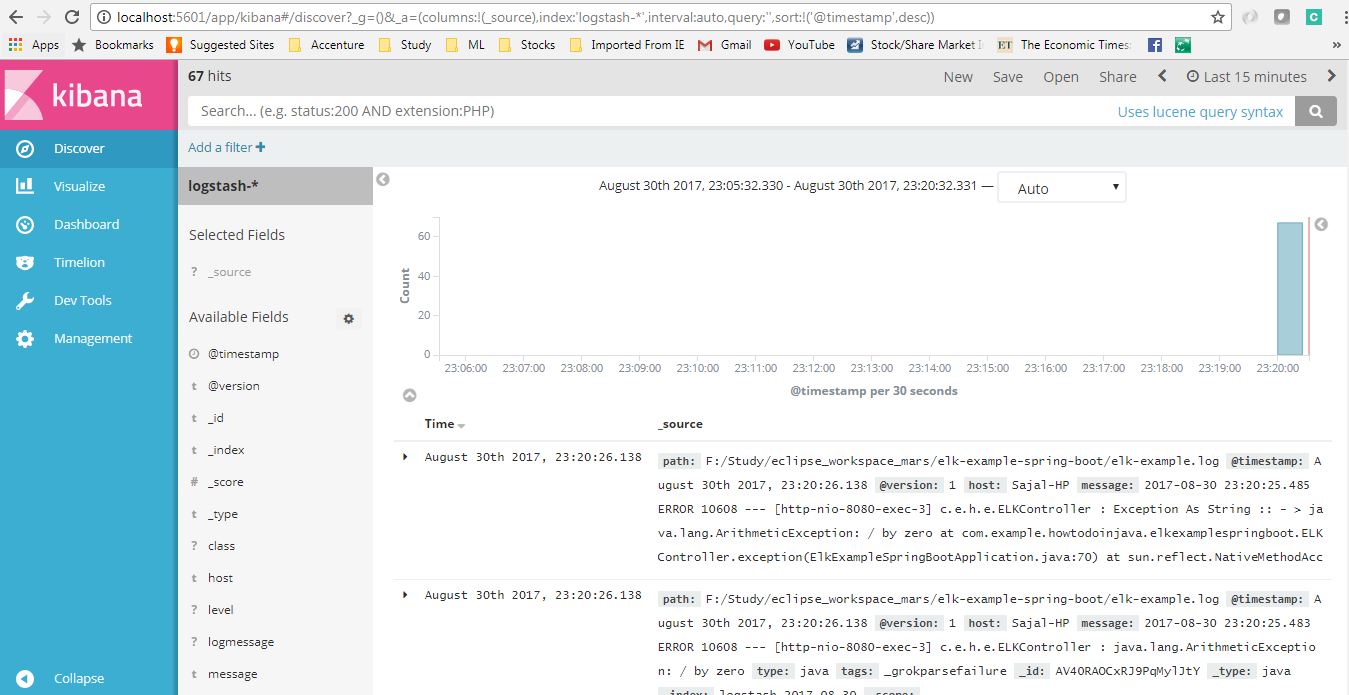
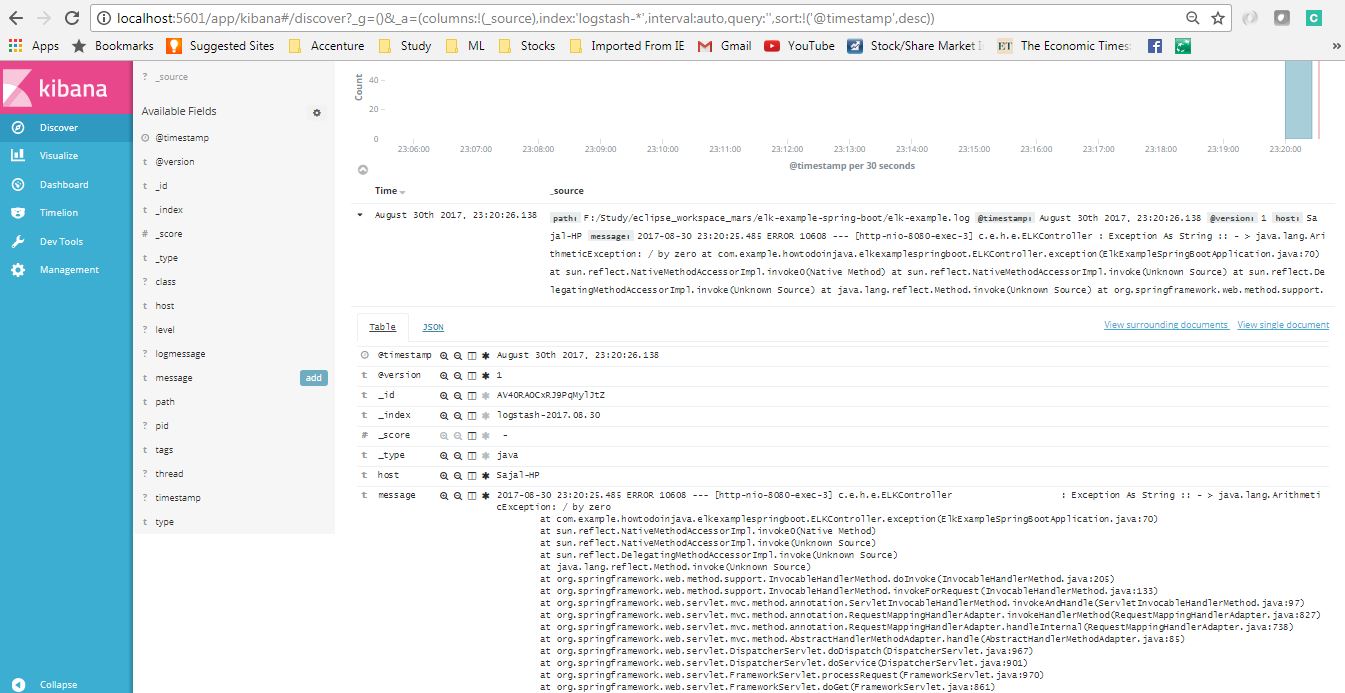
Before viewing the logs in Kibana, we need to configure the Index Patterns. We can configure logstash-\* as default configuration.

The index pattern management page will look like below. With this configuration we are pointing Kibana to Elasticsearch index(s) of your choice. Logstash creates indices with the name pattern of logstash-YYYY.MM.DD We can do all those configuration in Kibana console <http://localhost:5601/app/kibana> and going to Management link in left panel.

Logstash configuration in Kibana

**6. Verify ELK Stack**

Now when all components are up and running, let’s verify the whole ecosystem.

Kibana Logs OverviewKibana Logs details screen

**7. ELK Stack Tutorial – Summary**

In this **ELK example**, we learned to **configure ELK stack** and the saw how we can point our application log files to ELK and view and analyze logs in Kibana

Instead logstash to listen to our logs, we can use logback configuration to use TCP appender to send logs to a remote Logstash instance via TCP protocol.

* We can point multiple log files with Logstash.
* We can use more sophisticated filters in the logstash configuration file to do more as per our need.
* We can use remote ELK cluster to point to our log files, or push logs into, this is basically required when all the applications will be deployed in cloud.
* Create different index patterns in logstash.

ELK stack require Java 1.8 to be configured properly. I faced difficulty starting up those tools due to older JDK in system path.

2 Spring Boot Dev Tools Features

Spring Boot 1.3 provides another module called Spring Boot DevTools. DevTools stands for **Developer Tool.** The aim of the module is to try and improve the development time while working with the Spring Boot application. Spring Boot DevTools pick up the changes and restart the application.

Note:- We can implement the DevTools in our project by adding the following dependency in the pom.xml file.

Maven dependency:

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime<scope >

</dependency>

**Automatic Restart:** Auto-restart means reloading of Java classes and configure it at the server-side. After the server-side changes

The classes that do not change (third-Jars) are loaded in the **base ClassLoader.**

The classes that we are actively developing are loaded in the **restart ClassLoader.**

property **spring.devtools.restart.enabled** to **false.**

Remember:

The DevTools always monitors the classpath resources.

There is only a way to trigger a restart is to update the classpath.

DevTools required a separate application classloader to work properly. By default, Maven fork the application process.

Auto-restart works well with LiveReload.

DevTools depends on the application context's shutdown hook to close it during the restart.

LiveReload: The Spring Boot DevTools module includes an embedded server called LiveReload. It allows the application to automictically trigger a browser refresh whenever we make changes in the resources. It is also known as auto-refresh.

Note: We can disable the LiveReload by setting the property spring.devtools.livereload.enabled to false.

It provides browser extensions for Chrome, Firefox, and Safari. By default, LiveReload is enabled. The LiveReload works on the following path:

/META-INF/maven

/META-INF/resources

/resources

/static

/public

/templates

Thanku!!