Indian Institute of Information Technology-Allahabad

PROJECT REPORT

SEMESTER – VIII

Distributed framework for Back-Testing and Post Trade Analytics

Submitted By: Mayank Vijay

IIT2013127

CANDIDATES' DECLARATION

I hereby certify that the work which is being presented in the B.Tech. Project Report entitled "**Distributed framework for back-testing and post trade analytics**", being submitted as a part of VIIIth Semester Project Evaluation to the Department of Information Technology of Indian Institute of Information Technology, Allahabad, is an authenticated record of my original work in **Edelweiss Securities Ltd** from February 2017 to June 2017 carried out under the guidance of **Mr. Gaurav D Shah.** The project was done in full compliance with the requirements and constraints of the prescribed curriculum.

Date: 25th July 2017

Place: Mumbai

Student's Name: Mayank Vijay

Gaurav D. Shah

Vice President, **Systematic Trading**

Edelweiss Securities Ltd

TABLE OF CONTENTS

ABSTRACT

At Global Markets, we have several business requirements that require very quick analytics on huge data that helps the researchers and traders in making big profits.

Big data is really promising and differentiating for financial services companies that rely heavily on making money from money than selling products. With no physical products to manufacture, data is one of arguably their most important assets. The business of banking and financial management sector, especially if you work for department that thrives on profits in stock markets is where transactions are conducted in millions everyday, is driven by the stock market data received from the National Stock Exchange everyday during the market hours.

Simultaneously researchers apply different algorithms on it to make more and more profits for the company and traders trade heavily which also generates huge data everyday. Thus, the dataset is really huge which needs to be handled really efficiently as the business relies on back testing and analytics, for the creation of strategies and testing these strategies for authenticity and profit making ability. The performance and the ability to judge the optimal strategy greatly increase, as the dataset utilized in the build and back test strategies increase. Thus, handling Big Data is of at most importance for these firms.

The methods of data collection and analytics were traditionally used through excel and sheets for the ease of use fail to compete with the sheer size and variety of this data, thus it is required to design a system that can handle these humongous amounts of data. The best and optimal solution to tackle this problem is the creation of distributed systems and setting up frameworks that can work on distributed environment which are fast, fault-tolerant and resilient. This, project aims to build such a system of distributed network, where in humungous amounts of data can be stored, processed and analysed.

INTRODUCTION & MOTIVATION

At Global Markets we deal with various kinds of Algorithmic and Systematic Trading. Systematic trading is a way of defining trade goals, risk controls and rules that can make investment and trading decisions in a methodical way. Algorithmic trading is pre-programmed trading instructions to buy/sell shares to achieve maximum profits. Algorithmic and Systematic Trading identify futures by analysing models, trends, patterns across days or months or years, in various forms of trading like high-frequency trading, day trading, low-frequency trading etc . This requires deep analysis of large historical and live streaming data, which comprises of variables like date, product type, stock symbol etc.

The analysis of such large data can't be achieved on a single system. It requires the use of a distributed system.

A distributed system is a model in which components located on networked computers communicate and coordinate their actions by passing messages. The components interact with each other in order to achieve a common goal.

This project aims to design and implement such a distributed system which can be effectively utilized for trade research analytics and back testing of trade strategies.

The Distributed system will utilize various big data technologies such as Spark for running queries on Esper engine for the purpose of back testing and research analytics. Instead of storing the data and running queries against stored data, the Esper engine allows applications to store queries and run the data through. Response from the Esper engine is real-time when conditions occur that match queries. Different team was working on the esper engine part. I worked on making this task distributed on various nodes in hadoop cluster using spark framework and thereby reducing the burden on single machine machine by reducing the time taken to done the backtesting.

Post Trade Analytics is one of the most important aspects of this project done which required technologies such as Elastic Search, Logstash and Kibana (popularly called the ELK Stack). After the strategies that have been run for Backtesting as well as on live market data for trading, log files are generated using the logger class of the log4j API, a reliable, fast and flexible logging

framework written in Java. Moreover, various analysis is done everyday on distibuted framework using spark as well as by the various individual researchers on their systems which results in tremendous log files generated every day which can range from 100 GB to about 200 GB (will increase with setting up more systems, running more startegies and thus generating more logs) on last trading day of well as well as month. Thus writing a java code using spark API's is time consuming task for a process where data size can actually vary compared to the fix amount of market data received everyday. Thus we need a tool that makes this task simple for different varities of logs generated and works in a distributed manner.

Moreover, mining knowledge out of this varieties of data requires expertise and a large team, thus we need a tool that can transform plain logs to extract useful information out of it, store this data on a distributed cluster in a fast and effiecient manner which can be easily searched usign queriesas well as visualized so that it becomes easy for the non technical background employees such as traders because they are the ones who really need to see the results of the startegies used in their trades and do the post trade analytics to generate even better profits on the next trades they perform.

Keeping this scenario into account, ELK Stack is the most sought after technology and a distributed framework that helps us achieve an efficient way of doing post trade analytics.

PROBLEM DEFINITION

The aim of this project is to create a distributed system framework for back testing and post trade analytics. This project utilizes various technologies such as Spark, Elasticsearch, Logsatsh, Kibana and optimally run thousands of terabytes of data on thousands of commodity hardware nodes, producing results in the shortest amount of time possible and handle node failure. The problem can be broken down into 2 stages mainly:

- Using the power of distributed processing engine Spark to reduce the time taken by Esper (Complex Event Processing Engine) for backtesting on live simulated market data.
- Post Trade Analytics (By using logs generated from the strategies (Java Programs) run on data:
    a. Using Logstash to transform logs (generated by running various strategies on market data) and using filebeat to ship logs from production server to testing server.
    b. Storing the useful data extracted from logstash in ElasticSearch, a NoSQL Database as well as an effiecient full text search engine where the meaningful log data generated by logstash as JSON events can be stored as well as searched with very very less latency.
    c. Using Kibana for visualizing the data stored in ElasticSearch where those with no knowledge about about programming/technology but tremendous knowledge on how the financial markets work can view the results and perform the post trade analytics.

LITERATURE SURVEY

- Analyzing Log Analysis: An Empirical Study of User Log Mining by S. Alspaugh (UC Berkeley, 2014)

  Log analysis is the process of transforming raw log data into information for solving problems. The market for log analysis software is huge and growing as more business insights are obtained from logs. Stakeholders in this industry need detailed, quantitative data about the log analysis process to identify inefficiencies, streamline workflows, automate tasks, design high-level analysis languages, and spot outstanding challenges. For these purposes, it is important to understand log analysis.

- Geo-identification of web users through logs using ELK stack by Tarun Prakash (2017)

  Although, a lot of log management exist but they either fail to scale or are costly. Here efforts have been made to solve the shortcomings of prevailing log analyzer tools and this paper demonstrates the working of ELK ecosystem i.e. Elasticsearch, Logstash and Kibana clubbed together to efficiently analyze the log files and provide an interactive and easily understandable insights. Log management systems built on ELK stack are desired to analyze large log data sets while making the whole computation process easy to monitor through an interactive interface. Being from open source community ELK stack has many useful features for log analysis. Elasticsearch is used as Indexing, storage and retrieval engine. Logstash acts as a Log input slicer and dicer and output writer while Kibana performs Data visualization using dashboards

METHODOLOGY

1. Spark:

Spark is a fast, in-memory data processing engine with elegant and expressive development APIs to allow data

workers to efficiently execute streaming or SQL workloads that require fast iterative access to datasets.

Spark enables us to use distributed collection data-structure, known as Resilient distributed datasets (RDD) (later on replaced by datasets to offer more features) to make references to data, which can then be used in map-reduce jobs. Resilient Distributed Datasets (RDD) is a fundamental data structure of Spark. It is an immutable distributed collection of objects. Each dataset in RDD is divided into logical partitions, which may be computed on different nodes of the cluster. RDDs can contain any type of Python, Java, or Scala objects, including user-defined classes.

Esper is is Complex Event Processing engine for executing EPL(Event Processing Language) queries, which are SQL like queries on streaming data for backtesting and analytics. But the issue is that it takes time for a single machine to process queries on huge incoming data and thus spark comes into picture, for performing this analytics in a distributed manner.

Major Spark API used for this purpose is **Spark SQL** which is a Spark component that supports querying data either via SQL or via the Hive Query Language. Spark SQL is a Spark module for structured data processing and is all about distributed in memory computations on massive scale. It is the entry point for working with structured data (rows and columns) in Spark. Various functionalities of spark used are:

```
import org.apache.spark.sql.SparkSession
val spark = SparkSession
  .builder()
  .appName("Spark SQL basic example")
  .config("spark.some.config.option", "some-value")
  .getOrCreate()
```

```
val sqlDF = spark.sql("SELECT * FROM people")
sqlDF.show()
OUTPUT =>    // | age|  name|
             // |  30|  Andy|
             // |  19|  Justin|
```

- **Encoder** is the fundamental concept in the serialization and deserialization (SerDe) framework in Spark SQL 2.0. They are used to convert a JVM object of type T to and from the internal Spark SQL representation.
- **Row** is a generic row object with an ordered collection of fields that can be accessed by an index or a name. Row belongs to org.apache.spark.sql.Row package. The traits of Row are -

  length or size - Row knows the number of elements (columns).
  schema - Row knows the schema RowEncoder takes care of assigning a schema to a Row

  To create a new Row, use RowFactory.create() in Java or Row.apply() in Scala. RowEncoder is part of the Encoder Class and acts as a encoder for the DataFrames (or DataSets).

- **StructType** is a built-in data type in Spark SQL to represent a collection of StructFields (It has a name, the type and whether or not it be empty) that together define a schema or its part. A schema is the description of the structure of your data (which together create a Dataset in Spark SQL). StructType and StructField belong to the org.apache.spark.sql.types package.
- A **Dataset** is a distributed collection of data. It's an evolution of RDDs in later versions of spark. Dataset is a interface that provides benefits of Spark RDDs as well as Spark SQL's optimized execution engine. A Dataset can be constructed from JVM objects or from data extracted by running SQL query and then manipulated using functional transformations (map, flatMap, filter, etc.). Operations available on Datasets are divided into transformations and actions. Transformations are the ones that produce new Datasets, and actions are the ones that trigger computation and return results. Example transformations include map, mapPartitions, etc and actions include collect, show, or writing data out to file systems

  A DataFrame (also known as Dataset of Rows) is dataset organized into named columns. It is conceptually equivalent to a table in a relational database, but with richer optimizations under the hood. DataFrames can be constructed from a wide array of sources such as: structured data files, tables in Hive, external databases, or existing RDDs. Dataset<Row> is used to represent a DataFrame.

  To efficiently support domain-specific objects, an Encoder is required

- **SqlContext is** the entry point in spark for working with structured data (rows and columns) like data stored in MySQL or Hive.

Pseudo Code:

```
Encoder<Feed> feedEncoder = Encoders.bean(Feed.class);
Dataset<Feed> dataset = sparkSql.sql(sqlQuery).as(feedEncoder)
StructField[] sf = new StructField[sizeRow]
sf[0] = DataTypes.createStructField("day", DataTypes.DateType, false).....
ExpressionEncoder<Row> rowEncoder = RowEncoder.apply(new StructType(sf))
List<Row> = dataset.mapPartitions((MapPartitionsFunction<Feed, Row>) iterator -> {
        Esper Engine Instance created
        The methods called where order are placed and trades take place
        Esper Instance assigned to every partition of data thus making Esper Process parallelized
        return resultsStore.getResults()
}, rowEncoder).collectAsList()
```

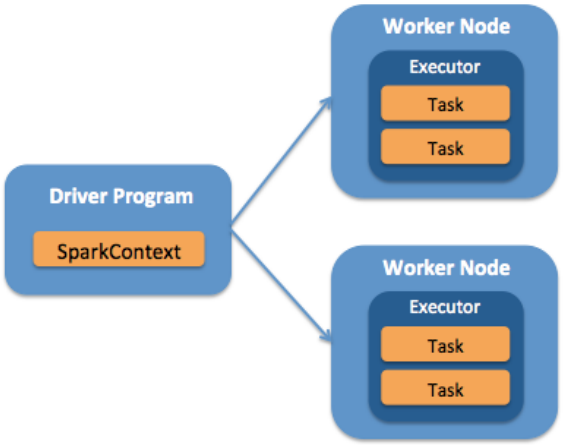


Figure 1: How Spark executor executes tasks in separate nodes

Result

The benefit of executing esper queries using spark increased the speed by 4 times when tested on dataset of 4 days. And bigger the dataset, more fast it becomes as we can increase the excutors and thus leverage the benefits of Distributed Programming via spark.

2) Post Trade Analytics

Why is Log Analysis Becoming More Important?

As money at Edelweiss Global Markets is earned by putting money in markets, thus verifying logs generated after strategy (Java application on single machine or distributed cluster) is run and analyzing these logs is becoming more and more critical.

In cloudbased infrastructures or distributed infrastructure, performance isolation is extremely difficult to reach particularly whenever systems are heavily loaded. The performance of virtual machines in the cloud can greatly fluctuate based on the specific loads, infrastructure servers, environments, and number of active users. As a result, reliability and node failures can become significant problems. Log management platforms can monitor all of these infrastructure issues as well as process operating system logs.

Problems faced when dealing with Logs Data

No Consistency — The variety of systems and absence of standards means that it's difficult to be a jack-of-all trades.

- Logging is different for each app, system, or device
- Specific knowledge is necessary for interpreting various types of logs
- Variation in format makes it challenging to search
- Many types of time formats

No centralization — Simply put, log data is everywhere:

- Logs in many locations on various servers
- Many locations of various logs on each server

Accessibility of Log Data— Much of the data is difficult to locate and manage. Although some of the log data may be highly valuable, many admins face these steep challenges:

- Access is often difficult
- High expertise to mine data
- Logs can be difficult to find
- Immense size of Log Data

The ELK stack helped us manage each of these challenges, and more. ELK is best for time-series data-anything with a time stamp-such as you'll find in most web server logs, transaction logs, and stock data listings. To be intelligible, these logs usually need substantial clean-up.

What is the ELK Stack?

The ELK Stack is a collection of three open source products: Elasticsearch, Logstash, and Kibana from Elastic. Elasticsearch is a NoSQL database that is based on the Lucene search engine. Logstash is a log pipeline tool that accepts inputs from various sources, executes different transformations, and exports the data to various targets. Kibana is a visualization layer that works on top of Elasticsearch.

Together, these three different open source products are most commonly used in log analysis in IT environments (though there are many more use cases for the ELK Stack starting including business intelligence, security and compliance, and web analytics). Logstash collects and parses logs, and then Elasticsearch indexes and stores the information. Kibana then presents the data in visualizations that provide actionable insights into the log data.

To sum up, ELK stack setup has three main components:

- **Logstash**: Collects, parses and processes incoming logs
- **Elasticsearch**: Stores all of the logs
- **Kibana**: Web interface for searching and visualizing logs

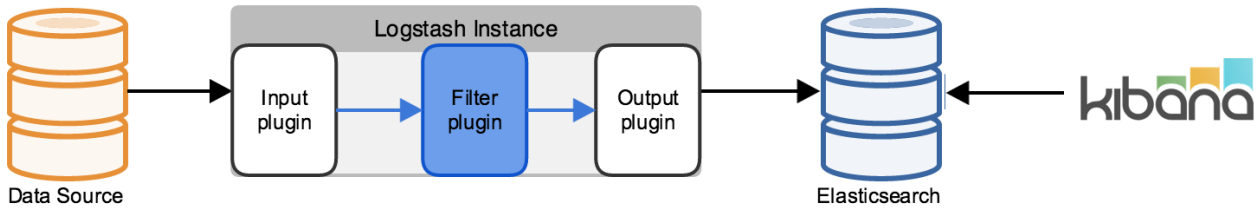


Figure 2: The ELK stack

Why is ELK So Popular?

The ELK Stack is popular because it fulfills a need in the log analytics space. Splunk's enterprise software has long been the market leader, but its numerous functionalities are increasingly not worth the expensive price.

After all, top tech companies of the world like Netflix, Facebook, Microsoft, LinkedIn, and Cisco monitor their logs using the ELK Stack.

2.1) Logstash

A great use for the ELK Stack is the storing, visualization, and analysis of logs and other time-series data. Logstash is an integral part of the data workflow from the source to Elasticsearch and further. Not only does it allow us to pull data from a wide variety of sources, it also gives us the tools to filter, massage, and shape the data so that it's easier to work with.

Why use Logstash

- **The supply engine for Elasticsearch**

  Scalable data pipeline closely integrated with elasticsearch

- **Providing the benefits of tremendous amount of plugins**

  Match the inputs and transform them into useful data and output it to Elasticseach or any other output form like file with the help of several plugins

- **Ease of use due to DSL (Domain Specific Language) and is thus user friendly**

  Over 200 plugins available, which make it easy for it to be used with any input and output

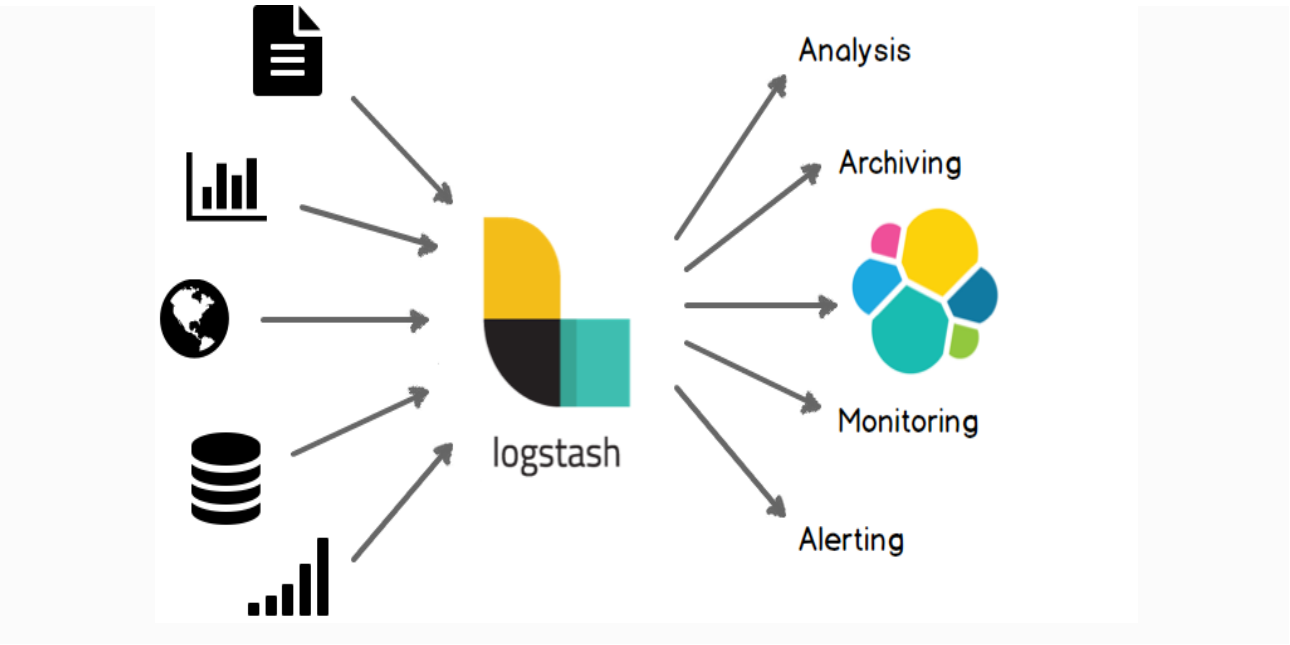


Figure 2.1.1: Logstash supports many plugins

Basic Logstash Concepts

- **Event:** A single unit of information, containing a timestamp plus additional data. An event arrives via an input, and is subsequently parsed, transformed using filters and passed through the Logstash pipeline.

- **Pipeline:** A term used to describe the flow of events through the Logstash workflow. A pipeline typically consists of a series of input, filter, and output stages. Input stages get data from a source and generate events. Filter stages modify the event data, and output stages write the data to a destination.

- **Indexer:** A Logstash instance that is tasked with interfacing with an Elasticsearch cluster in order to index event data.

- **Plugins:** A self-contained software package that implements one of the stages in the Logstash event processing pipeline. The list of available plugins includes input plugins, output plugins, codec plugins, and filter plugins. The plugins are implemented as Ruby gems and hosted on RubyGems.org. You define the stages of an event processing pipeline by configuring plugins.
- **Shipper:** An instance of Logstash that send events to logstash or any other output
- **Worker:** The filter thread model used by Logstash, where each worker receives an event and applies all filters, in order, before emitting the event to the output queue. This allows scalability across CPUs because many filters are CPU intensive.

15:29:19,116 Level[INFO] [pool-6-thread-2] ## New Order=> Portfolio = JUNE9900P;Security = NIFTY17JUN8200PE::NSE_FO;Order ID = 61686479;Account =2225;Side =SELL;RejectionReason: null;Quantity = 750;NewQuantity = 750;Price = 0.15;NewPrice = 0.15;TriggerPrice = 0.0;NewTriggerPrice = 0.0;OrderStatus = NOT_SENT;NewOrderStatus = NOT_SENT;OrderType = IOC;NewOrderType = IOC;FilledQuantity = 0;AverageFillPrice = 0.0;ExpectedPrice = 0.15;Bid = 0.15;LTP = 0.15;Ask = 0.2;ParentOrderID=0;ExchangeOrderID=0;OriginalExchangeOrderID=0;stsnId=9111074429831258055;Comment= ## com.edelweiss.algo.portfolio.UnsynchronizedDefaultPortfolioSecurity.placeOrder(UnsynchronizedDefaultPortfolioSecurity.java:494)

09:15:26,662 Level[ERROR] [pool-6-thread-2] ## Bid/Ask of securities is not set for constituent ID 3 NIFTY17AUG9900PE::NSE_FO AUG8000C ## com.edelweiss.algo.arb.ml.fourLeg.strategy.FourLegStrategy.priceUpdate(FourLegStrategy.java:144)

Figure 2.1.2: Strategy Logs are of various types, but the common part of all logs is the Order Placed information or error generated. This is an example of sample log data generated:

How Logstash Works

The Logstash event processing pipeline has three stages: inputs → filters → outputs. Inputs gener ate events, filters modify them, and outputs ship them elsewhere. Inputs and outputs support code cs that enable us to encode or decode the data as it enters or exits the pipeline without having to use a separate filter. The transformation pipeline of logstash works using the help of following plugins.

Input plugins

Input plugins are used to extract input from some input source as events to logstash pipeline. Most popular input plugins are:

- **file**: reads the log file from some file present in the system
- **beats**: process events received from beats, a log shipper tool useful when logs generated in many from many sources.

Filter plugins

Filters are intermediary processing devices in the Logstash pipeline. We can combine filters with conditionals to perform an action on an event if it meets certain criteria. Some useful fi lters include:

- **grok**: Grok is currently the best way in Logstash to parse unstructured log data into something structured and queryable. It consists of 120 patterns thus there is very less need to use regular expressions to match the data.
- **mutate**: perform general transformations on event fields. We can rename, remove, replace, and modify fields in your events.
- **drop**: drop an event completely, for example, *debug* events.
- **clone**: make a copy of an event, possibly adding or removing fields.
- **geoip**: add information about geographical location of IP addresses (also displays amazing charts in Kibana)

Output plugins

Outputs are the final phase of the Logstash pipeline. An event can pass through multiple outputs, but once all output processing is complete, the event has finished its execution. Some commonly used outputs include:

- **elasticsearch**: send event data to Elasticsearch. If we want to save our data in an efficient, convenient, and easily queryable format. Elasticsearch is the way to go.
- **file**: write event data to a file on disk.
- **stdout**: print event data directly on console

Codec plugins

Codecs are basically stream filters that can operate as part of an input or output. Codecs enable you to easily separate the transport of your messages from the serialization process. Popular codecs include:

- **json**: encode or decode data in the JSON format.
- **multiline**: used to merge multiple line events to a single event, specially in case of java stack trace messages

It works on the principle of ETL (Extract, Transform, and Load) where much less time is spent training Logstash to normalize the data, getting Elasticsearch to process the data, and then visualize it with Kibana. With Logstash, it's super easy to take all those logs and store them in a central location. The only prerequisite is a Java runtime, and it takes just two commands to get Logstash up and running.

Using Elasticsearch as a backend datastore and Kibana as a frontend dashboard (see below), Logstash will serve as the workhorse for storage, querying and analysis of your logs. Since it has an arsenal of ready made inputs, filters, codecs, and outputs, we can grab hold of a very powerful feature-set with a very little effort on your part.

We can think of Logstash as a pipeline for event processing: it takes precious little time to choose the inputs, configure the filters, and extract the ***relevant, high value*** data from your logs. Take a few more steps, make it available to Elasticsearch and we get superfast queries against our huge data.

Execution Model

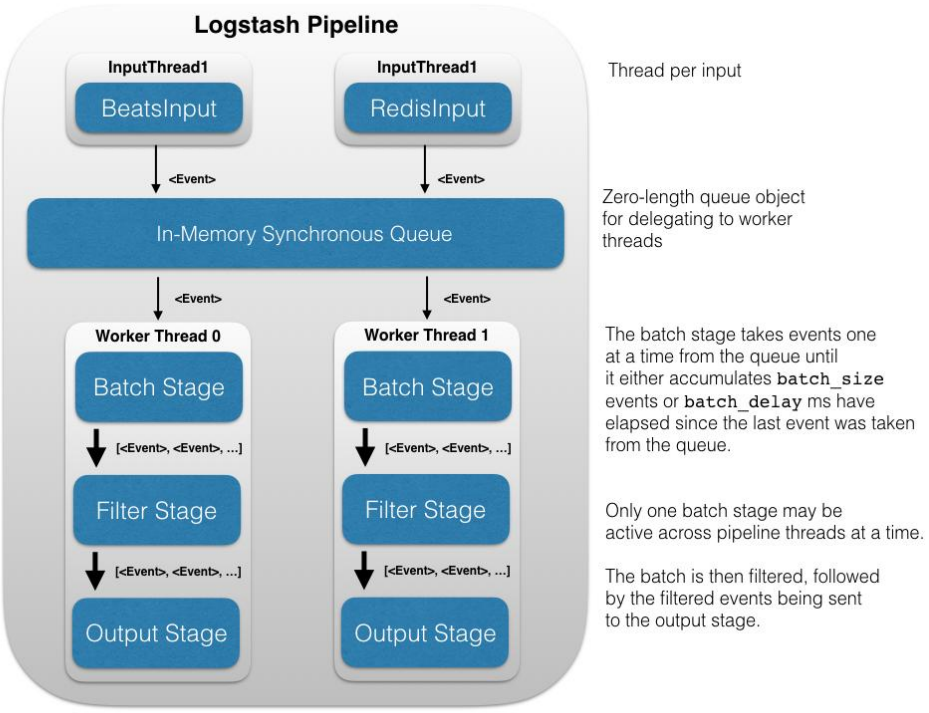


Figure 2.1.3: Logstash Pipeline Execution Workflow

The Logstash event processing pipeline coordinates the execution of inputs, filters, and outputs.

Each input stage in the Logstash pipeline runs in its own thread. Inputs write events to a common Java SynchronousQueue. This queue holds no events, instead transferring each pushed event to a free worker, blocking if all workers are busy. Each pipeline worker thread takes a batch of events off this queue, creating a buffer per worker, runs the batch of events through the configured filters, then runs the filtered events through any outputs. The size of the batch and number of pipeline worker threads can be configured.

By default, Logstash uses in-memory bounded queues between pipeline stages (input → filter and filter → output) to buffer events. If Logstash terminates unsafely, any events that are stored in memory will be lost. To prevent data loss, you can enable Logstash to persist in-flight events to disk.

Sample Logstash Configuration:

```
input {
    file {
        path => "/home/mayank/StrategyName.log"
        start_position => "beginning"
    }
}


filter {

    grok {
        match => ["path","/%{DATA}/%{DATA}/%{DATA:StrategyName}.log”]
    }

    grok {
        match => { "message" => "%{TIME:Time},%{NUMBER:Millis}
            Level\[%{WORD:LogLevel}\] %{DATA} ## New Order=> %{DATA:KeyValues}
            ## %{GREEDYDATA:LogSouce}" }
        add_tag => ["valid"]
        add_field => { "Order_Type" => "new_order" }
    }

    if "valid" not in [tags] {
        drop { }
    }

    kv {
        source => "KeyValues"
        field_split => ";"
        value_split => "="
        trim_key => " "
    }
}

output {
    elasticsearch {
        action => "index"
        hosts => [ "DataNode1IP:9200" , "DataNode2IP:9200" ]
        index => "%{Order_Type}"
        document_type => "%{StrategyName}"
    }
}
```

2.2) Elasticsearch

Elasticsearch is a highly scalable opensource full text search and analytics engine. It allows us to store, search, and analyze big volumes of data quickly and in near real time. It is generally used as the underlying engine/technology that powers applications that have complex search features and requirements.

Elasticsearch is a NoSQL database that is based on the Lucene search engine. That means it stores data in an unstructured way such that we cannot use SQL to query it. Unlike most NoSQL databases, though, Elasticsearch has a strong focus on search capabilities and features so much so, in fact, that the easiest way to get data from Elasticsearch is to search for it using the REST API.

Key Features of ElasticSearch:

- **Real-time data and real-time analytics.** The ELK stack gives us the power of real-time data insights, with the ability to perform superfast data extractions from virtually all structured or unstructured data sources. Real-time extraction, and real time analytics. Elasticsearch is the engine that gives you both the power and the speed.

- **Scalable, high-availability, multitenant-**
  It is built to scale horizontally out of the box. As you need more capacity, simply add another node and let the cluster reorganize itself to accommodate and exploit the extra hardware. Elasticsearch clusters are resilient, since they automatically detect and remove node failures. You can set up multiple indices and query each of them independently or in combination.

- **Full text search.** Under the cover, Elasticsearch uses Lucene to provide the most powerful fulltext search capabilities available in any opensource product. The search features come with multi language support, an extensive query language, geolocation support, and context-sensitive suggestions, and auto completion.

- **Document orientation.** You can store complex, real world entities in Elasticsearch as structured JSON documents. All fields have a default index, and you can use all the indices in a single query to get precise results in the blink of an eye.

Basic Concepts: There are a few concepts that are core to Elasticsearch

- **Near Real Time:** Elasticsearch is a near real time search platform. What this means is there is a slight latency (normally 10 - 100ms) from the time we index a document until the time it becomes searchable.
- **Node** is a running instance of elasticsearch which belongs to a cluster. Multiple nodes can be started on a single server for testing purposes, but usually you should have one node per server. At startup, a node will use unicast to discover an existing cluster with the same cluster name and will try to join that cluster.
- **Cluster** consists of one or more nodes which share the same cluster name. Each cluster has a single master node which is chosen automatically (or we can specify in the elasticsearch conf.yml file of on each node to specify which nodes to chose as the master nodes) by the cluster and which can be replaced if the current master node fails.
- **Document** is a JSON document which is stored in elasticsearch. It is like a row in a table in a relational database. Each document is stored in an index and has a type and an id. A document is a JSON object (also known in other languages as a hash / hashmap / associative array) which contains zero or more fields, or key and value pairs. The original JSON document that is indexed will be stored in the _source field, which is returned by default when getting or searching for a document.
- **Shards** is a single Lucene instance. It is a low-level "worker" unit which is managed automatically by elasticsearch. An index is a logical namespace which points to primary and replica shards. Other than defining the number of primary and replica shards that an index should have, you never need to refer to shards directly. Instead, our code should deal only with an index. Elasticsearch distributes shards amongst all nodes in the cluster and our data in index is distributed across shards, and can move shards automatically from one node to another in the case of node failure, or the addition of new nodes.

  Each document is stored in a single primary shard. When you index a document, it is indexed first on the primary shard, then on all replicas of the primary shard. By default, an index has 5 primary shards. You can specify fewer or more primary shards to scale the number of documents that your index can handle. You cannot change the number of primary shards in an index, once the index is created.
- **Index** is like a *table* in a relational database. It is a collection of documents that have some what similar characteristics. For example, if we have an index for customer data, another index for a product catalog, and yet another index for order data. An index is identified by a name (that must be all lowercase) and this name is used to refer to the index when performing indexing, search, update, and delete operations against the documents in it. In a single cluster, we can define as many indexes as you want.

Figure 2.2.1: How are shards allocated across nodes

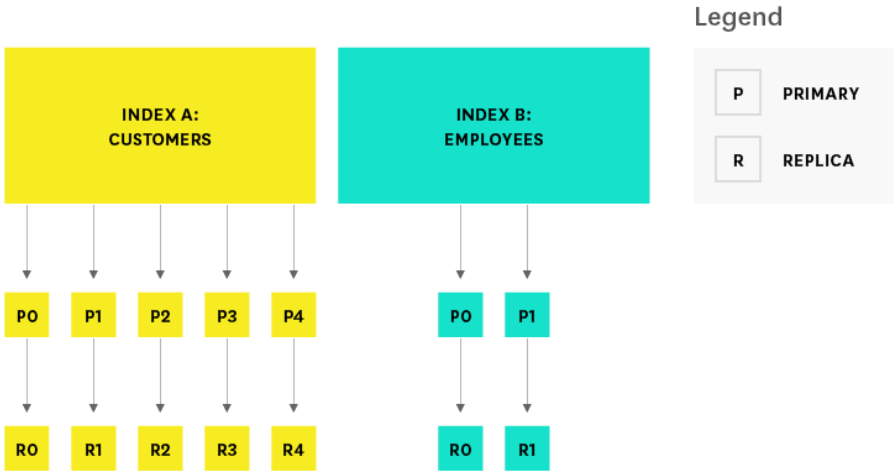


Figure 2.2.2: How are documents indexed across shards

- **Type** represents the *type* of document. Within an index, you can define one or more types. A type is a logical category/partition of your index whose semantics is completely up to you. In general, a type is defined for documents that have a set of common fields. Thus, the search API can filter documents by type.
- **ID** of a document identifies a document. The index/type/id of a document must be unique. If no ID is provided, then it will be auto-generated.

- **Field:** A document contains a list of fields, or key-value pairs. The value can be a simple (scalar) value (eg a string, integer, date), or a nestedstructure like an array or an object. A field is similar to a column in a table in a relational database. The mapping for each field has a field *type* that indicates the type of data that canbe stored in that field, eg integer, string, object. The mapping also allows you to define how the value for a field should be analyzed.

How Elasticsearch stores Data

In the real world, though, not all entities of the same type look the same. One person might have a home telephone number, while another person has only a cell phone number, and another might have both. One of the reasons that object oriented programming languages are so popular is that objects help us represent and manipulate real world entities with potentially complex data structures.

The problem comes when we need to store these entities. Traditionally, we have stored our data in columns and rows in a relational database. All the flexibility gained from using object s is lost because of the inflexibility of our storage medium. But what if we could store our objects as objects? Instead of modeling our application around the limitations of relational database, we can instead focus on *using* the data. The flexibility of objects is returned to us.

An *object* is a language specific, inmemory data structure. To send it across the network or store it, we need to be able to represent it in some standard format. JSON is away of representing objects in human readable text. It has become the de facto standard for exchanging da ta in the NoSQL world. When an object has been serialized into JSON, it is known as a *JSON document*. Elasticsearch is a distributed *document* store. It can store and retrieve complex data structures, serialized as JSON documents,in *real time*. In other words, as soon as a document has been stored in Elasticsearch, it can be retrievedfrom any node in the cluster.

Of course, we don't need to only store data; we must also query it and at speed. While NoSQL solutions exist that allow us to store objects as documents, they still requireus to think about how we want to query our data, and which fields require an index in order to make data retrieval fast. In Elasticsearch, *all data in every field* is *indexed by default*. That is, every field has a dedicated inverted index for fast retrieval. And, unlike most other databases, it can use all of those inverted indices *in the same query*, to return results at breathtaking speed.

```
        ⇨  curl '10.250.33.249:9200/_cat/indices?v'
health status index            uuid pri rep docs.count docs.deleted store.size
pri.store.size
green  open   new_order_update UA5   5   1      51085            0    107.2mb
54.2mb
green  open   execution_update sJ55   5   1       1350            0      6.7mb
3.2mb
green  open   new_order        8W45   5   1      25245            0       55mb
27.5mb
green  open   replace_update   sa34   5   1         25            0    946.7kb
473.3kb
```

Figure 2.2.3: Getting the status of indices in elasticsearch via REST API

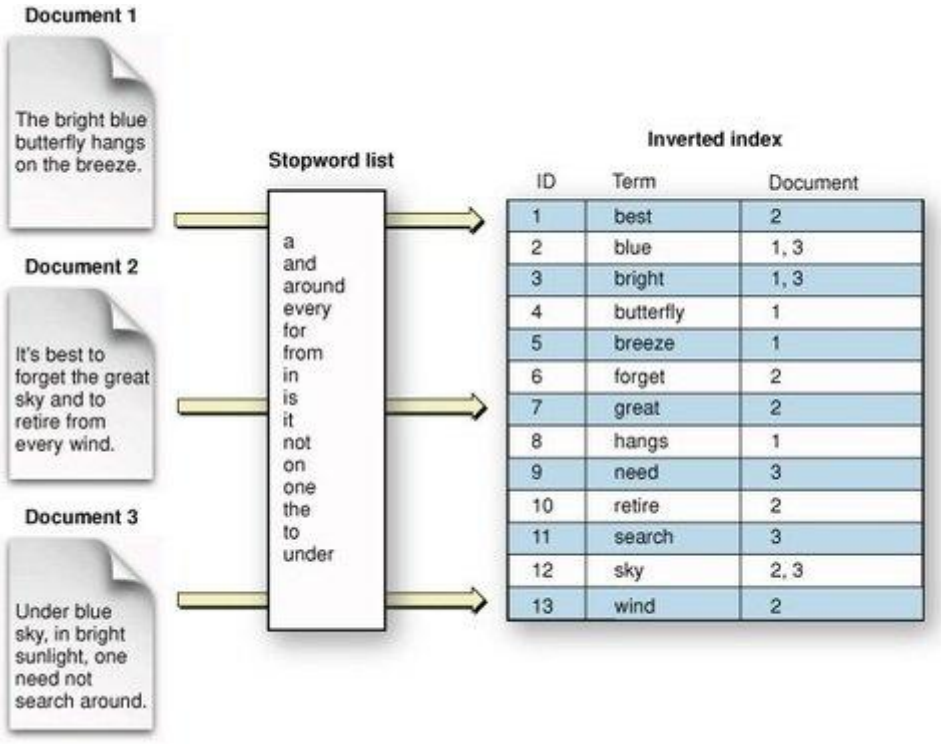


Figure 2.2.4: How is document data stored in inverted indices for fast retrieval when queried/searched

The Concept of Sharding

Sharding is important for two primary reasons:

- It allows us to horizontally split / scale our content volume
- It allows us to distribute and parallelize operations across shards (potentially on multiple nodes) thus increasing performance / throughput. The mechanics of how a shard is distributed and also how its documents are aggregated back into search requests are completely managed by Elasticsearch and is transparent to us as the user.

In a network/cloud environment where failures can be expected anytime, it is very useful and highly recommended to have a failover mechanism in case a shard/node somehow goes offline or disappears for whatever reason. For this reason, Elasticsearch allows us to make one or more copies of your index's shards into what are called replica shards.

Replication is important for two primary reasons:

- It provides high availability in case a shard/node fails. For this reason, it is important to note that a replica shard is never allocated on the same node as the original / primary shard that it was copied from.
- It allows us to scale out your search volume/throughput since searches can be executed on all replicas in parallel.

Each Elasticsearch shard is a Lucene index. There is a maximum number of documents we can have in a single Lucene index. As of LUCENE-5843, the limit is 2,147,483,519 (= Integer.MAX_VALUE - 128) documents. You can monitor shard sizes using the _cat/shards api.

```
                ⇨  curl '10.250.33.249:9200/_cat/shards?v'
   index            shard prirep state   docs   store ip            node
new_order_update 2     r      STARTED 4115   2.1mb 10.250.33.115 DataNode-1
new_order_update 2     p      STARTED 4115   2.1mb 10.250.33.249 DataNode-2
new_order_update 4     r      STARTED 4104   2.1mb 10.250.33.115 DataNode-1
new_order_update 4     p      STARTED 4104   2.1mb 10.250.33.249 DataNode-2
new_order_update 1     p      STARTED 4050   2.1mb 10.250.33.115 DataNode-1
execution_update 1     p      STARTED  369 372.4kb 10.250.33.115 DataNode-1
execution_update 1     r      STARTED  369 456.8kb 10.250.33.249 DataNode-2
execution_update 3     p      STARTED  412 502.5kb 10.250.33.115 DataNode-1
execution_update 3     r      STARTED  412 384.9kb 10.250.33.249 DataNode-2
execution_update 0     p      STARTED  349 376.6kb 10.250.33.115 DataNode-1
execution_update 0     r      STARTED  349 465.7kb 10.250.33.249 DataNode-2
.kibana          0     p      STARTED    6  43.3kb 10.250.33.115 DataNode-1
.kibana          0     r      STARTED    6  43.3kb 10.250.33.249 DataNode-2
replace_update   1     p      STARTED    9 137.3kb 10.250.33.115 DataNode-1
replace_update   1     r      STARTED    9 137.3kb 10.250.33.249 DataNode-2
replace_update   3     p      STARTED    6 121.4kb 10.250.33.115 DataNode-1
replace_update   3     r      STARTED    6 121.4kb 10.250.33.249 DataNode-2
replace_update   0     p      STARTED    8 152.8kb 10.250.33.115 DataNode-1
replace_update   0     r      STARTED    8 152.8kb 10.250.33.249 DataNode-2
new_order        2     p      STARTED 1995 866.9kb 10.250.33.115 DataNode-1
new_order        2     r      STARTED 1995   1.1mb 10.250.33.249 DataNode-2
new_order        4     p      STARTED 2079   1.1mb 10.250.33.115 DataNode-1
new_order        4     r      STARTED 2079   1.1mb 10.250.33.249 DataNode-2
new_order        1     r      STARTED 1954 891.8kb 10.250.33.115 DataNode-1
```

Figure 2.2.5: Getting the status of shards present in nodes in cluster

To summarize, each index can be split into multiple shards. An index can be replicated zero (meaning no replicas) or more times.Once replicated, each index has primary shards (the original shards that were replicated from) and replica shards (the copies of the primary shards). The number of shards andreplicas can be defined per index at the time the index is created. After the index iscreated, we can change the number of replicas dynamically anytime but you cannot change the number of shards after-the-fact.

By default, each index in Elasticsearch is allocated 5 primary shards and 1 replica which means that if you have at least two nodes in your cluster, your index will have 5 primaryshards and another 5 replica shards (1 complete replica) for a total of 10 shards per index.

2.3) Kibana

Till now, we've stored the useful information in logs in a queryable format. But to use elasticsearch, you need to know how to write domain specific queries. But the problem is that, the log data is more useful to the guy with tremendous knowledge on how markets work, rather than who develops this framework for use. And for those traders and researchers with market knowledge, we need a tool where we can view data stored in elasticsearch as graphs, or as tables by simply selecting the fields we want to view with respect to some factor like timestamp. This is where the power Kibana of ELK stack comes in to picture.

Kibana is an open source analytics and visualization platform designed to work with Elasticsearch. We use Kibana to search, view, and interact with data stored in Elasticsearch indices. You can easily perform advanced data analysis and visualize your data in a variety of charts, tables, and maps.

Kibana makes it easy to understand large volumes of data. Its simple, browser-based interface enables you to quickly create and share dynamic dashboards that display changes to Elasticsearch queries in real time.

Setting up Kibana is really easy. We can install Kibana, run its instance and start exploring our Elasticsearch indices with very less latency, no code, no additional infrastructure required.

Discovering our Data

Click **Discover** in the side navigation to display Kibana's data discovery functions:

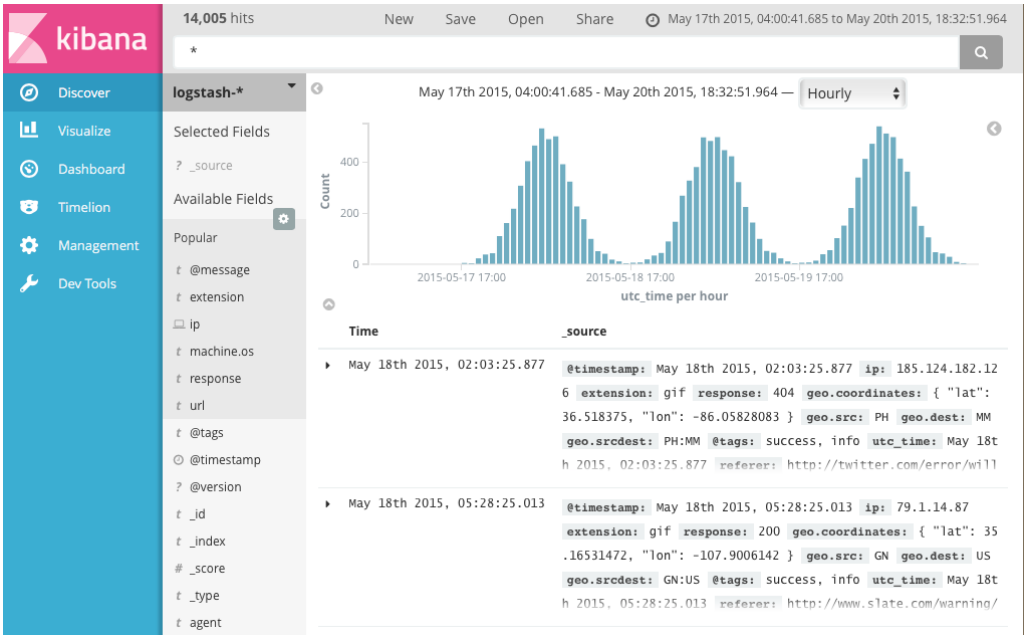


Figure 2.3.1: Data discovery in Kibana

In the query bar, we can enter an Elasticsearch query to search your data. We can explore the results in Discover and create visualizations of saved searches in Visualize.

The current index pattern is displayed beneath the query bar. The index pattern determines which indices are searched when you submit a query. To search a different set of indices, select different

pattern from the drop down menu. To add an index pattern, go to **Management/Kibana/Index Patterns** and click **Add New**.

We can construct searches by using the field names and the values we're interested in. With numeric fields you can use comparison operators such as greater than (>), less than (<), or equals (=). You can link elements with the logical operators AND, OR, and NOT, all in uppercase.

To try it out, select the ba* index pattern and enter the following query string in the query bar:

```
account_number:<100 AND balance:>47500
```

This query returns all account numbers between zero and 99 with balances in excess of 47,500. When searching the sample bank data, it returns 5 results: Account numbers 8, 32, 78, 85, and 97.

Visualizing our Data

Visualizaion enables us to create built in graphs such as line chart, pie char, bar char, etc for various fields values or aggregations with respect to some fields like timestamp, date, security traded in markets, etc. Thus by clicking on this option, traders can achieve what they want to, ie, nalysis of results there strategies generated and thus make more profits in near future.

To start visualizing your data, we click **Visualize** in the side navigation.

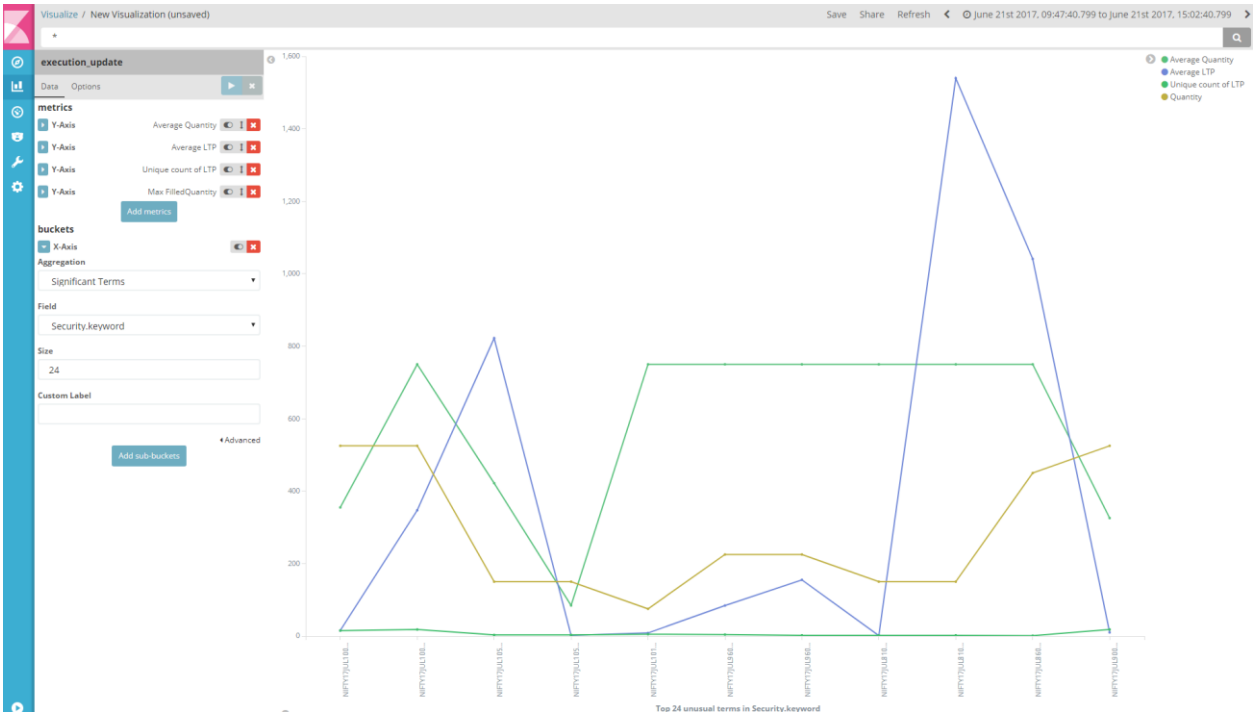


Figure 2.3.2: Data visualization in kibana

Thus, the overall workflow of kibana can be visualized as:

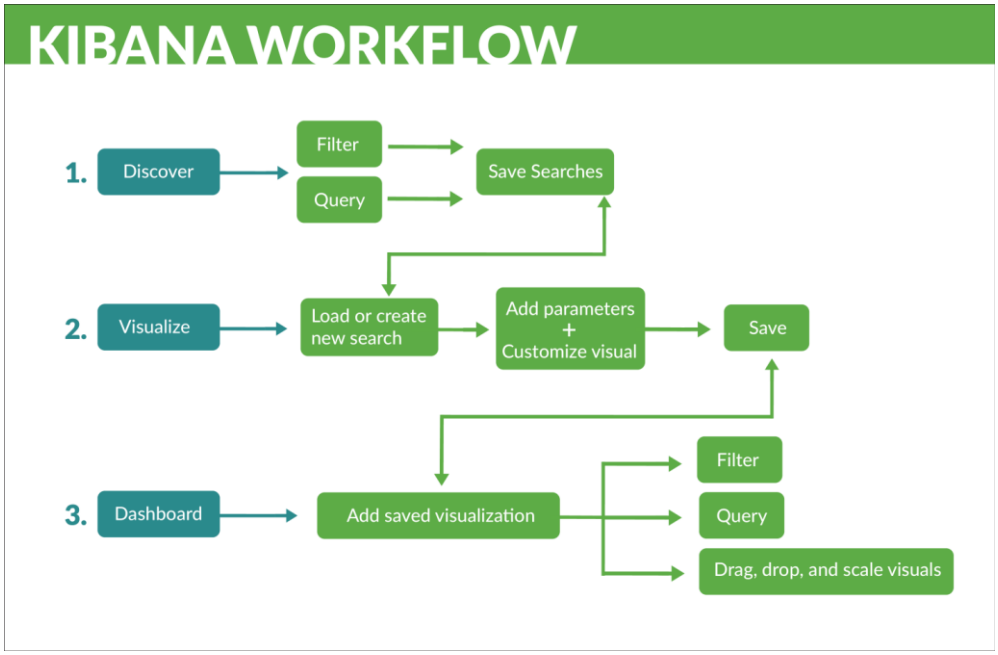


Figure 2.3.3: Kibana Workflow

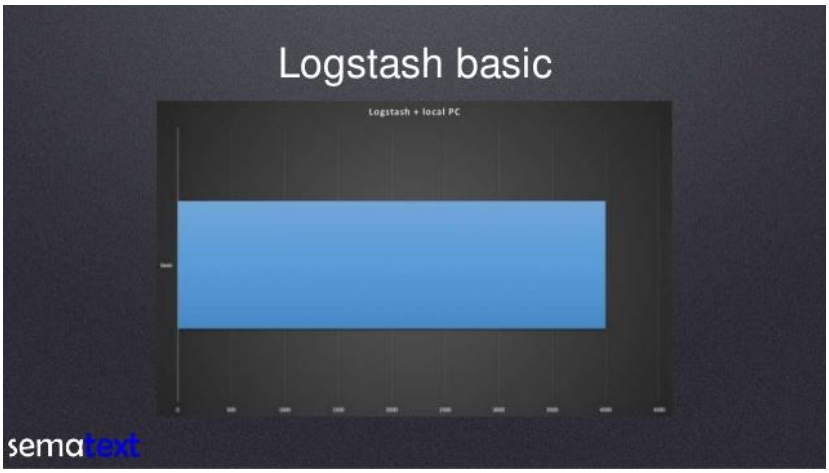
RESULTS AND ANALYSIS

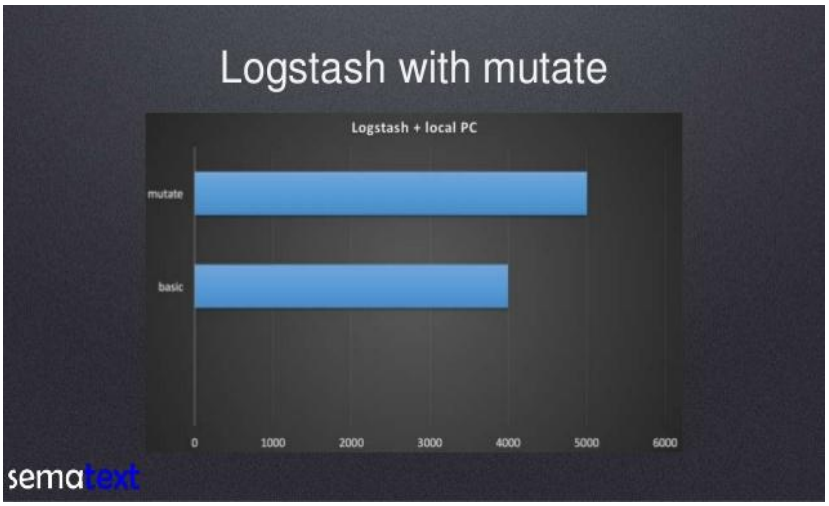
As we could see, the main time taken by ELK stack depends on how fast logstash parses and trasforms the logs and sends it to Elasticsearch for storage. Since the time interval betwenn event obtained by elasticsearch and till it gets stored as document is almost fixed (<10 ms) and moreover this time will improve as we improve number of nodes in elasticsearch cluster for obvious reasons, the time taken maily depends on how fast can logstash process the event and send it to Elasticsearch.

Logstash consists of an important metric filter plugin, which helps us in understanding the metrics behind logstash, like total count of events, per second event rate in a 1-minute sliding window (or a 5 or a 15 minute sliding window), the maximum/minimum values for the corresponding metric, etc.

Case1: Basic Pipeline

Generally, our setup has a 3 node cluster of which 2 are data nodes, the number of events transformed and sent per second is nearly 4K events / second.



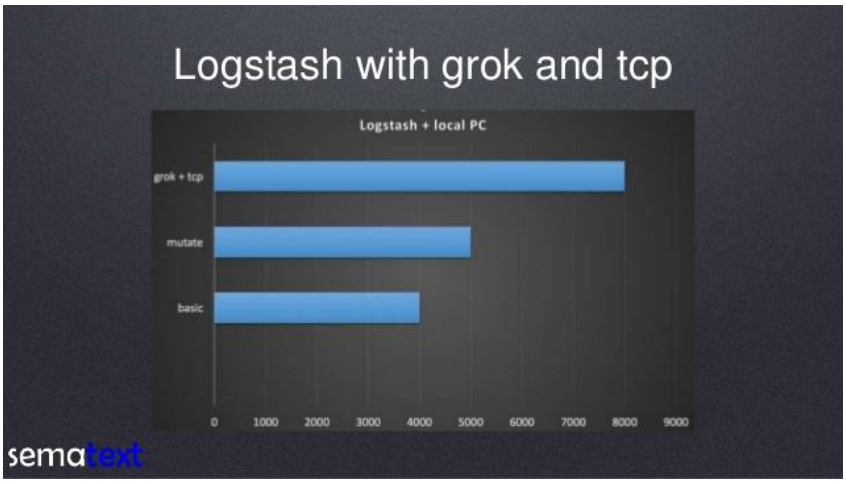


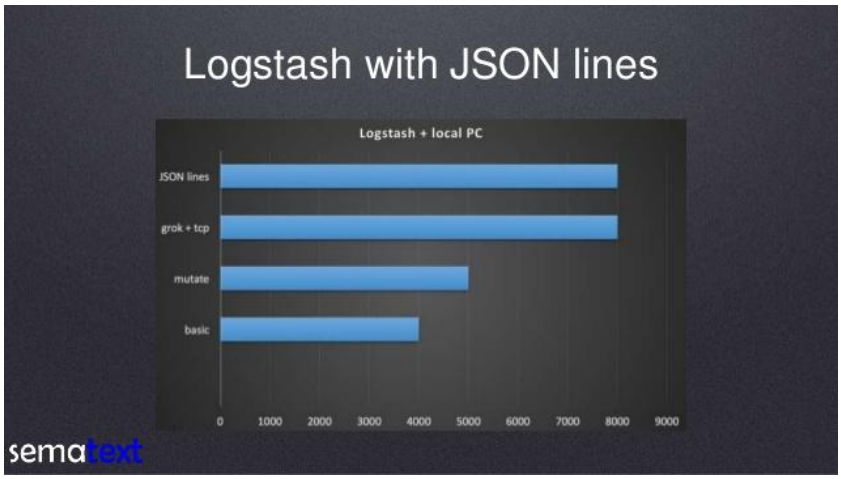
Case2: Using mutate filter to remove redundant and unnecessary fields

5K events are transformed and sent per second

Case 3: Using grok filters to parse the events in a better manner and structure the events

8K events are transformed and sent per second with memory usage of 330 MB





Case 4: If input is in JSON format

8K events are transformed and sent per second with memory usage of 320 MB

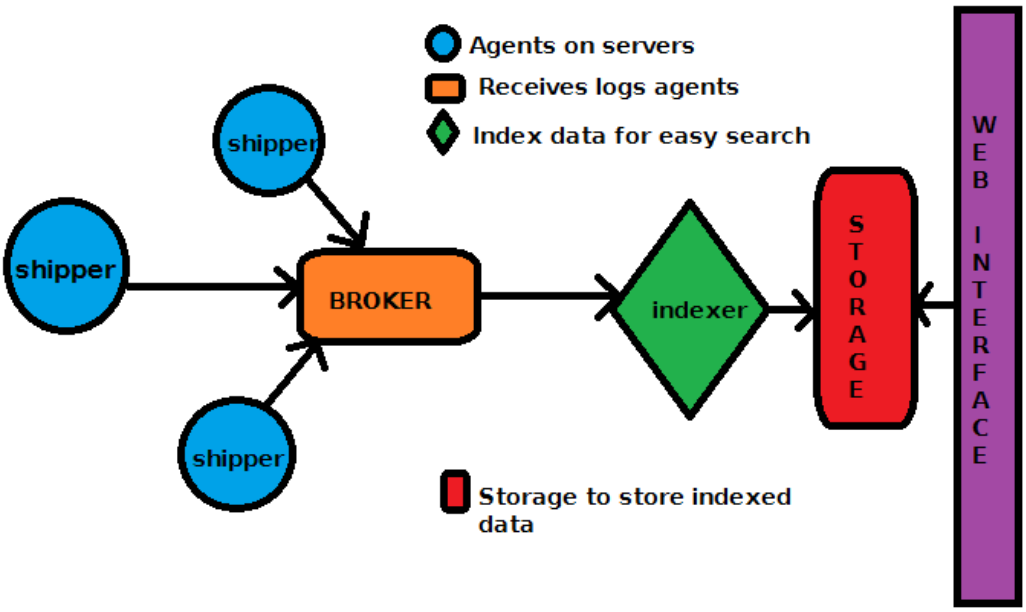
However, Elasticsearch searching and scalability can be improved if shards and indexes are assigned in an appropriate manner, as they are the core behind how elasticsearch works.

For example, specifying types within index can increase the search speed as we have logically partitioned the data. For example, if there is data oncontinents and our index continent name, if we mention type name as country, then it is obviously more easy to search data related to India which is not confined to India type within Asia index. Rather than searching in whle Asia index.

As more nodes are incorporated into the elasticsearch cluster, more performance evaluations will be done to improve the performance.

CONCLUSION

Thus, the overall workflow of ELK stack can be visualized as:



Hence, the process of Post Trade Analytics which earlier required downloading log files on invidual systems and analysing them by applying algorithms has been simplified such that the overall setup is not only distributed, but the data to be analysed can also be visualized using kibana which is very easily integrable with the data storage and search engine, Elasticsearch.

REFERENCES

1) https://www.elastic.co/guide/en/elasticsearch/reference/current

2) https://www.elastic.co/guide/en/logstash/reference/current

3) https://www.elastic.co/guide/en/kibana/reference/current

4) https://www.spark.apache.org/docs/latest/sql-programming-guide

5) https://www.researchgate.net/publication/305675550_Geo-identification_of_web_users_through_logs_using_ELK_stack

6) https://people.ischool.berkeley.edu/~hearst/papers/lisa2014-final-version.pdf

FUTURE SCOPE

After succesfully deploying the ELK Stack for analysis of logs, i would like to take it one step ahead. There is a Machine Learning library integrated in the latest version of kibana, which helps analyze time related anamolies, and thus notify if any of the trades or orders as per logs data is varying compared to the other ones.

This would give more reasonable insight as this anamoly predicted based on on the combination of various unsupervised learning algorithms in this ML library can help identify the past mistakes, so as to make more money in the future.

SUGGESTIONS FROM BOARD MEMBERS