## ELK Stack

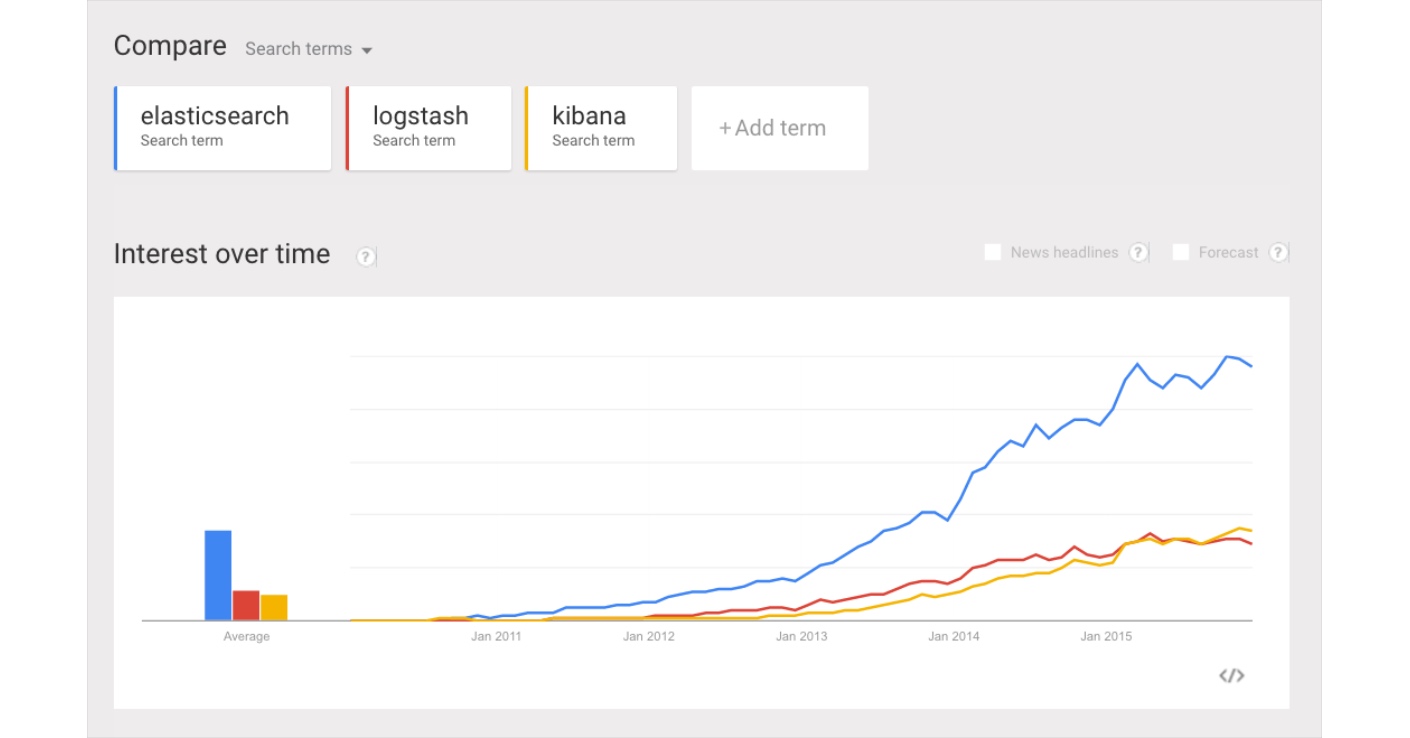
## What is the ELK Stack?



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 one’s environment.

### Why is ELK So Popular?



Google Trends screenshot

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 — especially for smaller companies such as SasS products and tech startups.

For that reason, Splunk has the aforementioned small number of customers while ELK is downloaded more times in a single month than Splunk’s total customer count — and many times over at that. ELK might not have all of the features of Splunk, but it does not need those analytical bells and whistles. ELK is a simple but robust log analysis platform that costs a fraction of the price.

The bigger picture: IT organizations are increasingly favoring open source products in general, and this is why newer proprietary log analysis software platforms such as Sumo Logic, which self-reports only 700 customers, might have a hard time gaining traction today.

After all, how do Netflix, Facebook, Microsoft, LinkedIn, and Cisco monitor their logs? With ELK.

### Why is Log Analysis Becoming More Important?

As more and more IT infrastructures move to public clouds such as Amazon Web Services and Microsoft Azure, public cloud security tools and log analytics platforms are both becoming more and more critical.

In cloud-based infrastructures, 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, NGINX and IIS server logs for technical SEO and web traffic analysis, application logs, and ELB and S3 logs on AWS.

In all of these contexts, DevOps engineers, system administrators, site reliability engineers, and developers can all use logs to make better decisions that are data-informed (and not, as Facebook’s Adam Mosseri says, data-driven). After all, what is being called “big data analytics” is increasingly important for a number of reasons — particularly when it comes to the cloud.

### How to Use the ELK Stack for Log Analysis

As I mentioned above, the ELK Stack is most commonly used in log analysis. However, the implementation and maintenance of a production-grade ELK Stack requires a lot of additional work and many additional products.

### Basics

### Elasticsearch

Elasticsearch is an open-source, broadly-distributable, readily-scalable, enterprise-grade search engine. Accessible through an extensive and elaborate API, Elasticsearch can power extremely fast searches that support your data discovery applications.

Indexing

Elasticsearch is able to achieve fast search responses because, instead of searching the text directly, it searches an index instead.

This is like retrieving pages in a book related to a keyword by scanning the index at the back of a book, as opposed to searching every word of every page of the book.

This type of index is called an inverted index, because it inverts a page-centric data structure (page->words) to a keyword-centric data structure (word->pages).

Elasticsearch uses Apache Lucene to create and manage this inverted index.

How Elasticsearch represents data

In Elasticsearch, a Document is the unit of search and index.

An index consists of one or more Documents, and a Document consists of one or more Fields.

In database terminology, a Document corresponds to a table row, and a Field corresponds to a table column.

Schema

Elasticsearch is schema-free. Whilst you are not required to specify a schema before indexing documents, it is necessary to add mapping declarations if you require anything but the most basic fields and operations.

This is no different from specifying a schema!

The schema declares:

* what fields there are
* which field should be used as the unique/primary key
* which fields are required
* how to index and search each field

In Elasticsearch, an index may store documents of different "mapping types". You can associate multiple mapping definitions for each mapping type. A mapping type is a way of separating the documents in an index into logical groups.

To create a mapping, you will need the Put Mapping API, or you can add multiple mappings when you create an index.

Query DSL

The Query DSL is Elasticsearch's way of making Lucene's query syntax accessible to users, allowing complex queries to be composed using a JSON syntax.

Like Lucene, there are basic queries such as term or prefix queries and also compound queries like the bool query.

## Installing and running Elasticsearch

For the purposes of this tutorial, I'll assume you're on a Linux or Mac environment.

You should also have JDK 6 or above installed.

1. Copy paste the zar archive of the elasticsearch on any specified location of your choice.
2. Unzip it and go to its bin folder.
3. You can simply run it ./elasticsearch command.

Basic Configurations:

1. Initially the elasticsearch runs on port 9200 on your system. You can change the port if this port is already occupied in config/elasticsearch.yml file.
2. Elasticsearch uses 1GB of your current memory space. If you want to increase this space in case of heavy load you can change it using the following command.
3. You can set it via the command line as “export ES\_HEAP\_SIZE=10g”
4. Alternatively, you can pass in the heap size via JVM flags when starting the process, if that is easier for your setup: “ES\_JAVA\_OPTS="-Xms10g -Xmx10g" ./bin/elasticsearch”

You can refer to the following link if you want more knowledge about the heap sizing<https://www.elastic.co/guide/en/elasticsearch/guide/current/heap-sizing.html>.

You can check the heap size and change it accordingly

java -XX:+PrintFlagsFinal -version | grep -iE 'HeapSize|PermSize|ThreadStackSize'

## Indexing Data

We're now going to index some data to our Elasticsearch instance. We'll use the example of a blog engine, which has some posts and comments.

curl -XPUT 'http://localhost:9200/blog/user/dilbert' -d '{ "name" : "Dilbert Brown" }'

curl -XPUT 'http://localhost:9200/blog/post/1' -d '

{

"user": "dilbert",

"postDate": "2011-12-15",

"body": "Search is hard. Search should be easy." ,

"title": "On search"

}'

curl -XPUT 'http://localhost:9200/blog/post/2' -d '

{

"user": "dilbert",

"postDate": "2011-12-12",

"body": "Distribution is hard. Distribution should be easy." ,

"title": "On distributed search"

}'

curl -XPUT 'http://localhost:9200/blog/post/3' -d '

{

"user": "dilbert",

"postDate": "2011-12-10",

"body": "Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat" ,

"title": "Lorem ipsum"

}'

To each of these requests, you should have received a response that verifies that the operation was successful, for example:

{"ok":true,"\_index":"blog","\_type":"post","\_id":"1","\_version":1}

Let's verify that all operations were successful.

curl -XGET 'http://localhost:9200/blog/user/dilbert?pretty=true'

curl -XGET 'http://localhost:9200/blog/post/1?pretty=true'

curl -XGET 'http://localhost:9200/blog/post/2?pretty=true'

curl -XGET 'http://localhost:9200/blog/post/3?pretty=true'

Note that there are 2 main ways of adding data to Elasticsearch:

1. json over HTTP
2. Native client

## Searching

Let's see if we can retrieve the documents we just added via search.

Find all blog posts by Dilbert:

curl 'http://localhost:9200/blog/post/\_search?q=user:dilbert&pretty=true'

This returns the following JSON result:

{

"took" : 85,

"timed\_out" : false,

"\_shards" : {

"total" : 5,

"successful" : 5,

"failed" : 0

},

"hits" : {

"total" : 3,

"max\_score" : 1.0,

"hits" : [ {

"\_index" : "blog",

"\_type" : "post",

"\_id" : "1",

"\_score" : 1.0, "\_source" :

{

"user": "dilbert",

"postDate": "2011-12-15",

"body": "Search is hard. Search should be easy." ,

"title": "On search"

}

}, {

"\_index" : "blog",

"\_type" : "post",

"\_id" : "2",

"\_score" : 0.30685282, "\_source" :

{

"user": "dilbert",

"postDate": "2011-12-12",

"body": "Distribution is hard. Distribution should be easy." ,

"title": "On distributed search"

}

}, {

"\_index" : "blog",

"\_type" : "post",

"\_id" : "3",

"\_score" : 0.30685282, "\_source" :

{

"user": "dilbert",

"postDate": "2011-12-10",

"body": "Lorem ipsum dolor sit amet, consectetuer adipiscing elit, sed diam nonummy nibh euismod tincidunt ut laoreet dolore magna aliquam erat volutpat" ,

"title": "Lorem ipsum"

}

} ]

}

All posts which don't contain the term search:

curl 'http://localhost:9200/blog/post/\_search?q=-title:search&pretty=true'

Retrieve the title of all posts which contain search and not distributed:

curl 'http://localhost:9200/blog/post/\_search?q=+title:search%20-title:distributed&pretty=true&fields=title'

A range search on postDate:

curl -XGET 'http://localhost:9200/blog/\_search?pretty=true' -d '

{

"query" : {

"range" : {

"postDate" : { "from" : "2011-12-10", "to" : "2011-12-12" }

}

}

}'

The usual Lucene query syntax is available either through the JSON query language, or through the query parser.

## Shutdown

To shutdown Elasticsearch, from the terminal where you launched elasticsearch, hit Ctrl+C. This will shut down ElasticSearch cleanly.

Elasticsearch is fairly robust, so even in situations of OS or disk crashes, it is unlikely that ElasticSearch's index will become corrupted.

Kibana

Kibana is an open source data visualization plugin for [Elasticsearch](https://en.wikipedia.org/wiki/Elasticsearch). It provides visualization capabilities on top of the content indexed on an Elasticsearch cluster. Users can create bar, line and scatter plots, or pie charts and maps on top of large volumes of data.

## Setting your index(es)

## If you start up Kibana for the first time you will be asked to configure an index pattern. The first important step is to select whether you want to handle time-based events or any other data set that contains a timestamp in each document or if you want to work with "static" data like the bank sample data.

## In elasticsearch it is common to store time-based events in multiple indexes to facilitate search and allow for memory optimization. With one index per day, you could then have a naming pattern for the indexes like *twitter-2015.01.15*, *twitter-2015.01.26*, and so on. If you store your data like that (which is not the case for the sample data) you should mark the "Use event times to create index names" checkbox. In that case you can specify the index pattern as follows: *[twitter-]YYYY.MM.DD*. Also, you will have to select if you use hourly, daily, weekly, monthly or yearly indexes.

## Configure the sample index

## After creating the index pattern, the index screen will show you a list of all fields that exist in the document with further information about their type, whether they are indexed and whether their contents have been analysed.

## Discovering Your Data

## Click Discover in the side navigation to display Kibana’s data discovery functions:

## images/tutorial-discover.png

## In the query bar, you can enter an [Elasticsearch query](https://www.elastic.co/guide/en/elasticsearch/reference/5.0/query-dsl-query-string-query.html#query-string-syntax) to search your data. You 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.

## You can construct searches by using the field names and the values you’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.

## images/tutorial-discover-2.png

## By default, all fields are shown for each matching document. To choose which document fields to display, hover over the Available Fields list and click the add button next to each field you want include. For example, if you add just the account\_number, the display changes to a simple list of five account numbers:

## images/tutorial-discover-3.png

## Setting the Time Filter

## The time filter restricts the search results to a specific time period. You can set a time filter if your index contains time-based events and a time-field is configured for the selected index pattern.

## By default the time filter is set to the last 15 minutes. You can use the Time Picker to change the time filter or select a specific time interval or time range in the histogram at the top of the page.

## To set a time filter with the Time Picker:

## Click Time Picker images/time-picker.jpg in the Kibana toolbar.

## To set a quick filter, click one of the shortcut links.

## Time filter shortcuts

## To specify a time filter relative to the current time, click Relative and specify the start time as a number of seconds, minutes, hours, days, months, or years ago.

## Relative time filter

## To specify both the start and end times for the time filter, click Absolute and select a start and end date. You can adjust the time by editing the To and From fields.

## Searching Your Data

## You can search the indices that match the current index pattern by entering your search criteria in the Query bar. You can perform a simple text search, use the Lucene [query syntax](https://lucene.apache.org/core/2_9_4/queryparsersyntax.html), or use the full JSON-based [Elasticsearch Query DSL](https://www.elastic.co/guide/en/elasticsearch/reference/5.0/query-dsl.html).

When you submit a search request, the histogram, Documents table, and Fields list are updated to reflect the search results. The total number of hits (matching documents) is shown in the toolbar. The Documents table shows the first five hundred hits. By default, the hits are listed in reverse chronological order, with the newest documents shown first. You can reverse the sort order by clicking the Time column header. You can also sort the table by the values in any indexed field. For more information, see [Sorting the Documents Table](https://www.elastic.co/guide/en/kibana/current/document-data.html#sorting).

To search your data, enter your search criteria in the Query bar and press Enter or click Search images/search-button.jpg to submit the request to Elasticsearch.

* To perform a free text search, simply enter a text string. For example, if you’re searching web server logs, you could enter safari to search all fields for the term safari.
* To search for a value in a specific field, prefix the value with the name of the field. For example, you could enter status:200 to find all of the entries that contain the value 200 in the status field.
* To search for a range of values, you can use the bracketed range syntax, [START\_VALUE TO END\_VALUE]. For example, to find entries that have 4xx status codes, you could enter status:[400 TO 499].
* To specify more complex search criteria, you can use the Boolean operators AND, OR, andNOT. For example, to find entries that have 4xx status codes and have an extension of php or html, you could enter status:[400 TO 499] AND (extension:php OR extension:html).

### Saving a Search

Saving searches enables you to reload them into Discover and use them as the basis for visualizations. Saving a search saves both the search query string and the currently selected index pattern.

To save the current search:

1. Click Save in the Kibana toolbar.
2. Enter a name for the search and click Save.

You can import, export and delete saved searches from Management/Kibana/Saved Objects.

Opening a Saved Search

To load a saved search into Discover:

1. Click Open in the Kibana toolbar.
2. Select the search you want to open.

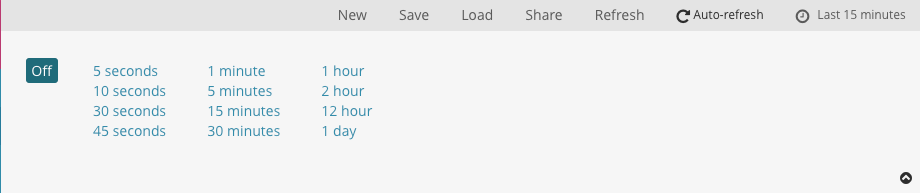
If the saved search is associated with a different index pattern than is currently selected, opening the saved search also changes the selectedindex pattern.

### Changing Which Indices You’re Searching

When you submit a search request, the indices that match the currently-selected index pattern are searched. The current index pattern is shown below the toolbar. To change which indices you are searching, click the index pattern and select a different index pattern.

### Refreshing the Search Results

As more documents are added to the indices you’re searching, the search results shown in Discover and used to display visualizations get stale. You can configure a refresh interval to periodically re submit your searches to retrieve the latest results. To enable auto refresh:

1. Click the Time Picker Time Picker in the Kibana toolbar.
2. Click Auto refresh.
3. Choose a refresh interval from the list.
4. 

When auto refresh is enabled, the refresh interval is displayed next to the Time Picker, along with a Pause button. To temporarily disable auto refresh, click Pause.

## Absolute time filter

## Click the caret in the bottom right corner to close the Time Picker.

## To set a time filter from the histogram, do one of the following:

## Click the bar that represents the time interval you want to zoom in on.

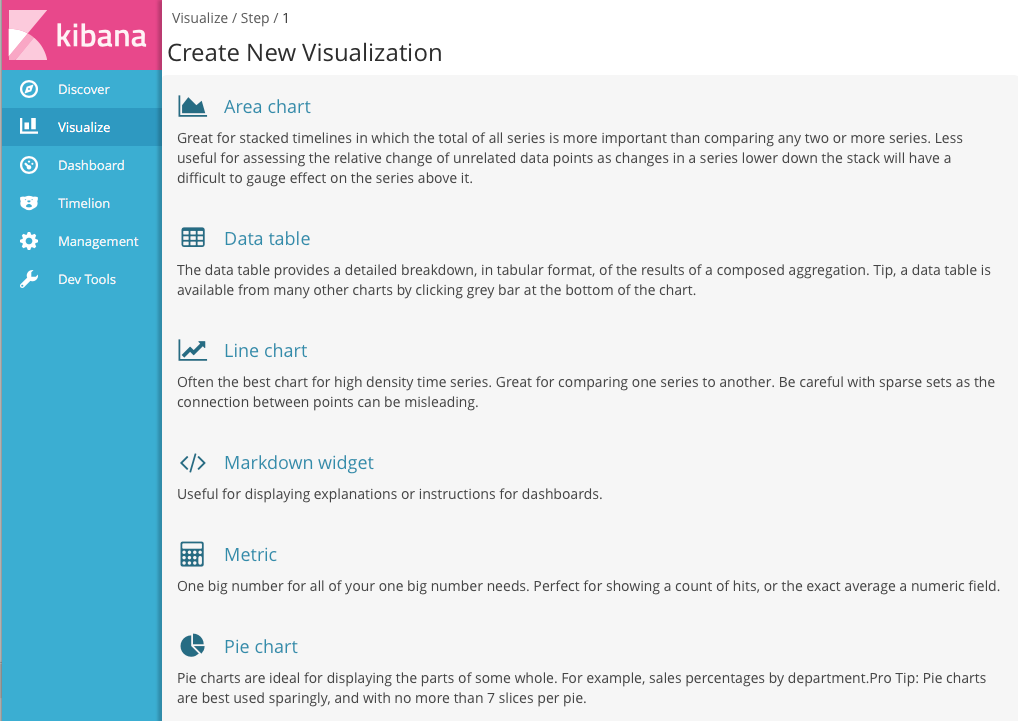
## Click and drag to view a specific timespan. You must start the selection with the cursor over the background of the chart—the cursor changes to a plus sign when you hover over a valid start point.

## You can use the browser Back button to undo your changes.

## The displayed time range and interval are shown on the histogram. By default, the interval is set automatically based on the time range. To use a different interval, click the link and select an interval.

## Visualizing Your Data

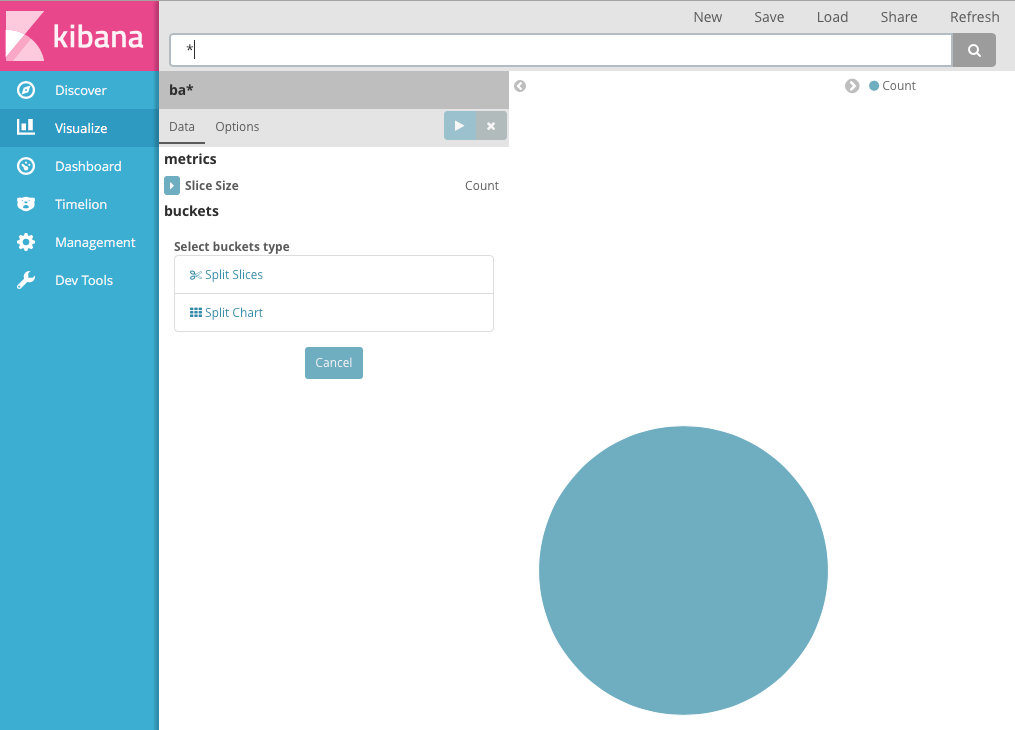
## To start visualizing your data, click Visualize in the side navigation:



The Visualize tools enable you to view your data in several ways. For example, let’s use that venerable visualization, the pie chart, to get some insight into the account balances in the sample bank account data.

To get started, click Pie chart in the list of visualizations. You can build visualizations from saved searches, or enter new search criteria. To enter new search criteria, you first need to select an index pattern to specify what indices to search. We want to search the account data, so select the ba\*index pattern.

The default search matches all documents. Initially, a single "slice" encompasses the entire pie:

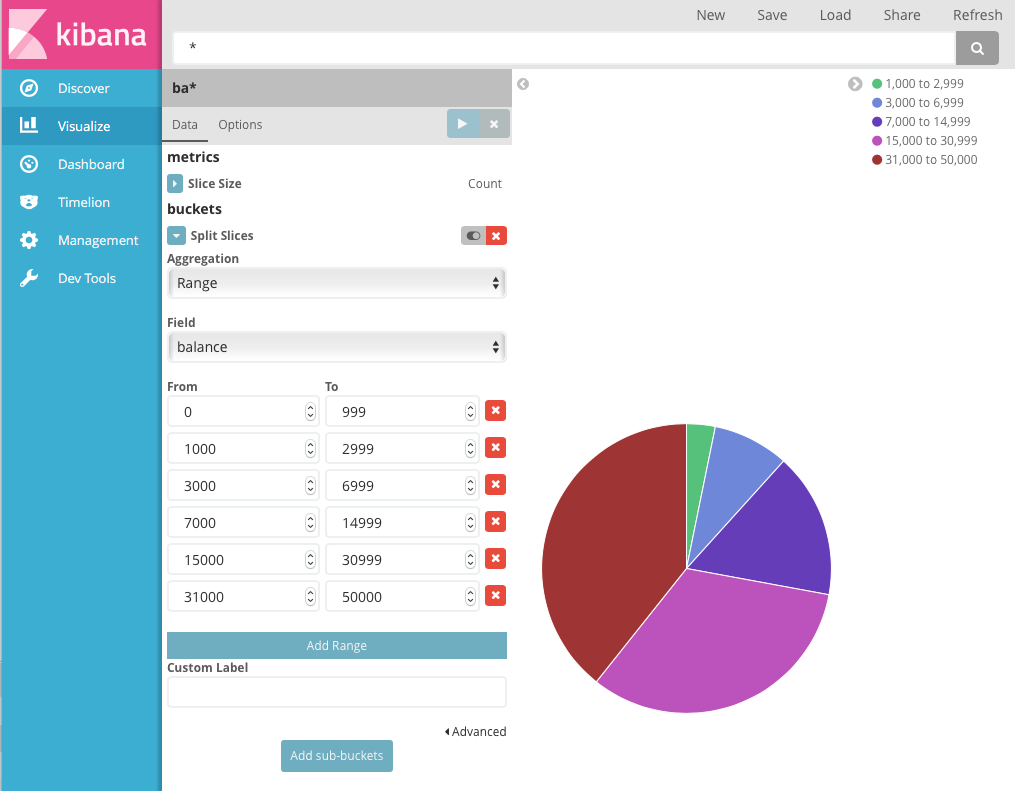


To specify what slices to display in the chart, you use an Elasticsearch [bucket aggregation](https://www.elastic.co/guide/en/elasticsearch/reference/5.0/search-aggregations.html). A bucket aggregation simply sorts the documents that match your search criteria into different categories, aka *buckets*. For example, the account data includes the balance of each account. Using a bucket aggregation, you can establish multiple ranges of account balances and find out how many accounts fall into each range.

To define a bucket for each range:

1. Click the Split Slices buckets type.
2. Select Range from the Aggregation list.
3. Select the balance field from the Field list.
4. Click Add Range four times to bring the total number of ranges to six.
5. Define the following ranges:
6. 0 999  
   1000 2999  
   3000 6999  
   7000 14999  
   15000 30999  
   31000 50000
7. Click Apply changes images/apply-changes-button.png to update the chart.

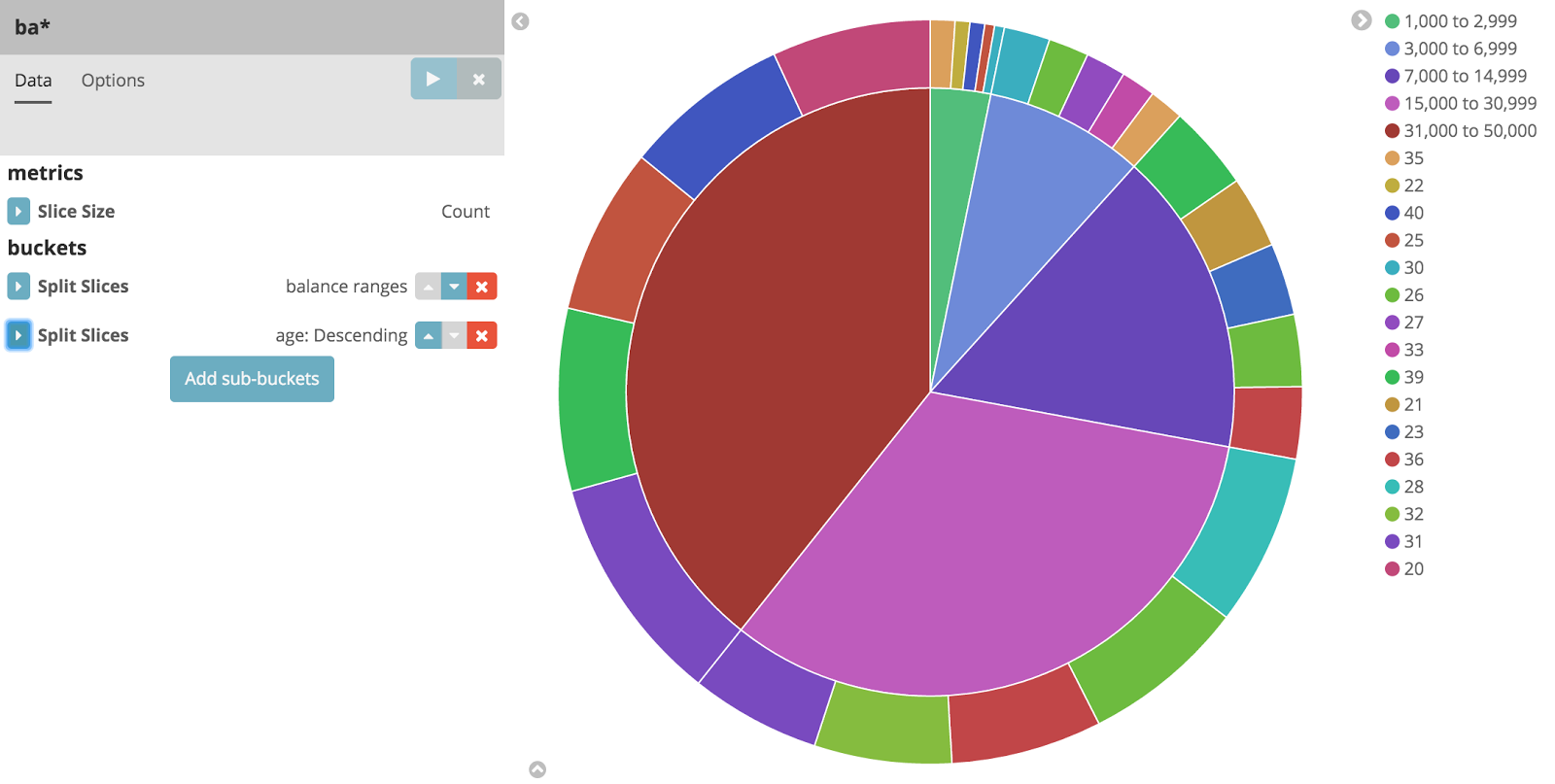
Now you can see what proportion of the 1000 accounts fall into each balance range.



Let’s take a look at another dimension of the data: the account holder’s age. By adding another bucket aggregation, you can see the ages of the account holders in each balance range:

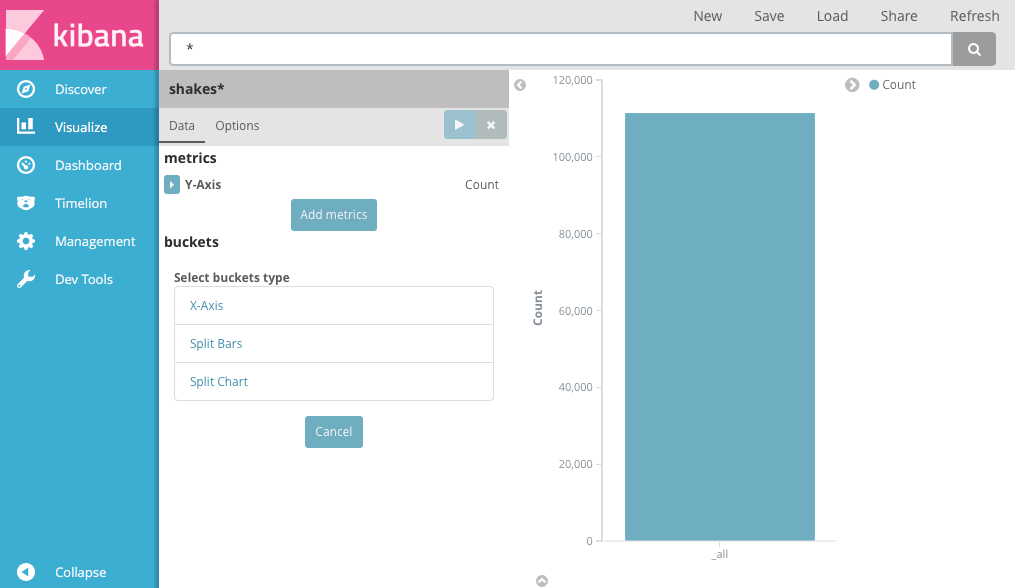
1. Click Add sub-buckets below the buckets list.
2. Click Split Slices in the buckets type list.
3. Select Terms from the aggregation list.
4. Select age from the field list.
5. Click Apply changes images/apply-changes-button.png.

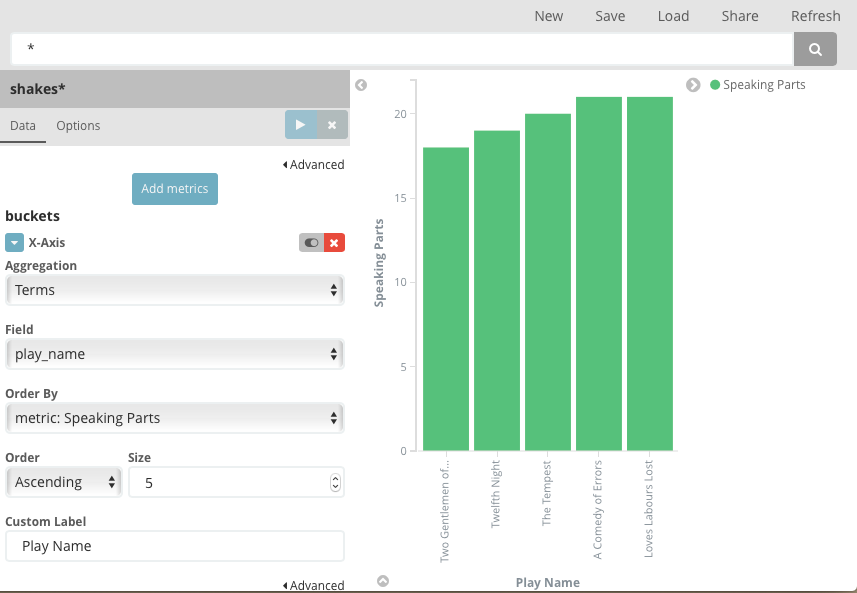
Now you can see the breakdown of the account holders' ages displayed in a ring around the balance ranges.



To save this chart so we can use it later, click Save and enter the name *Pie Example*.

Next, we’re going to look at data in the Shakespeare data set. Let’s find out how the plays compare when it comes to the number of speaking parts and display the information in a bar chart:

1. Click New and select Vertical bar chart.
2. Select the shakes\* index pattern. Since you haven’t defined any buckets yet, you’ll see a single big bar that shows the total count of documents that match the default wildcard query.
3. 
4. To show the number of speaking parts per play along the y-axis, you need to configure the Y-axis [metric aggregation](https://www.elastic.co/guide/en/elasticsearch/reference/5.0/search-aggregations.html). A metric aggregation computes metrics based on values extracted from the search results. To get the number of speaking parts per play, select the Unique Count aggregation and choose speaker from the field list. You can also give the axis a custom label, *Speaking Parts*.
5. To show the different plays long the x-axis, select the X-Axis buckets type, select Terms from the aggregation list, and choose play\_name from the field list. To list them alphabetically, select Ascending order. You can also give the axis a custom label, *Play Name*.
6. Click Apply changes images/apply-changes-button.png to view the results.

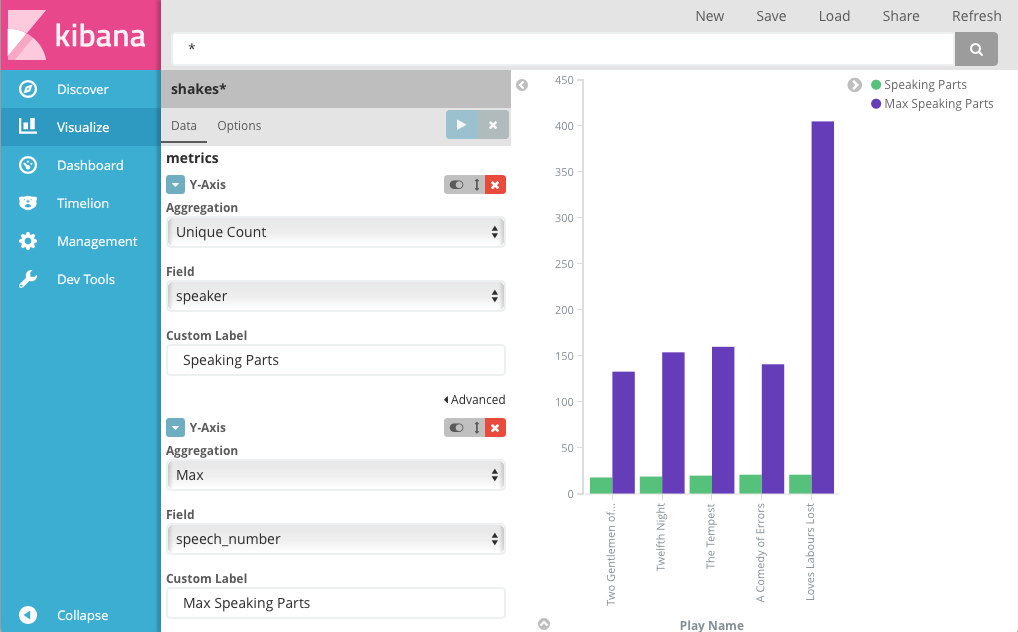


Notice how the individual play names show up as whole phrases, instead of being broken down into individual words. This is the result of the mapping we did at the beginning of the tutorial, when we marked the play\_name field as *not analyzed*.

Hovering over each bar shows you the number of speaking parts for each play as a tooltip. To turn tooltips off and configure other options for your visualizations, select the Visualization builder’s Options tab.

Now that you have a list of the smallest casts for Shakespeare plays, you might also be curious to see which of these plays makes the greatest demands on an individual actor by showing the maximum number of speeches for a given part.

1. Click Add metrics to add a Y-axis aggregation.
2. Choose the Max aggregation and select the speech\_number field.
3. Click Options and change the Bar Mode to grouped.
4. Click Apply changes images/apply-changes-button.png. Your chart should now look like this:

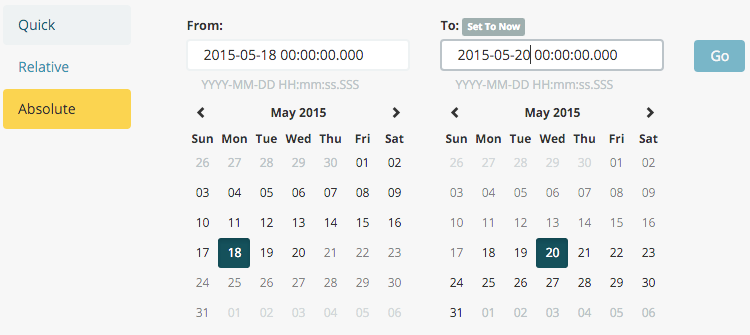


As you can see, *Love’s Labours Lost* has an unusually high maximum speech number, compared to the other plays, and might therefore make more demands on an actor’s memory.

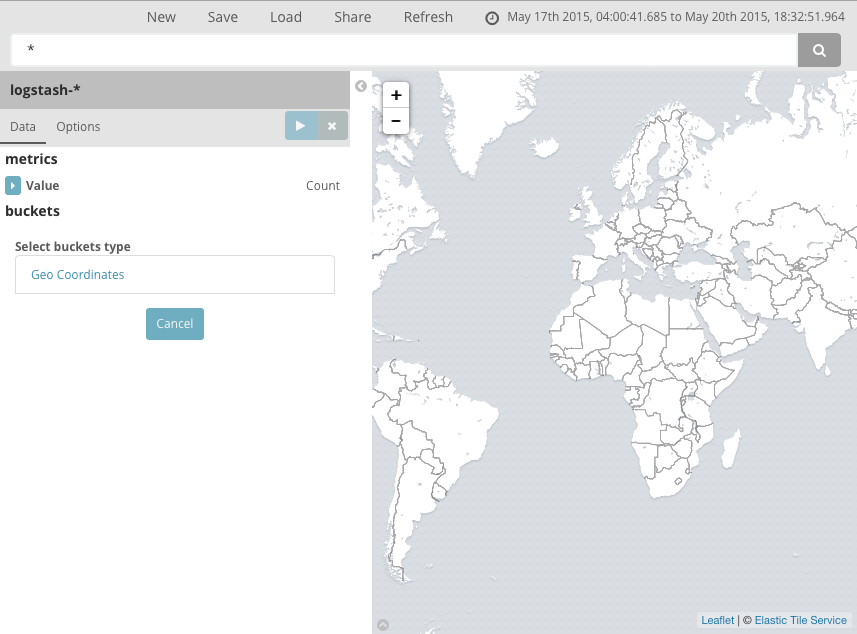
Note how the Number of speaking parts Y-axis starts at zero, but the bars don’t begin to differentiate until 18. To make the differences stand out, starting the Y-axis at a value closer to the minimum, go to Options and select Scale Y-Axis to data bounds.

Save this chart with the name *Bar Example*.

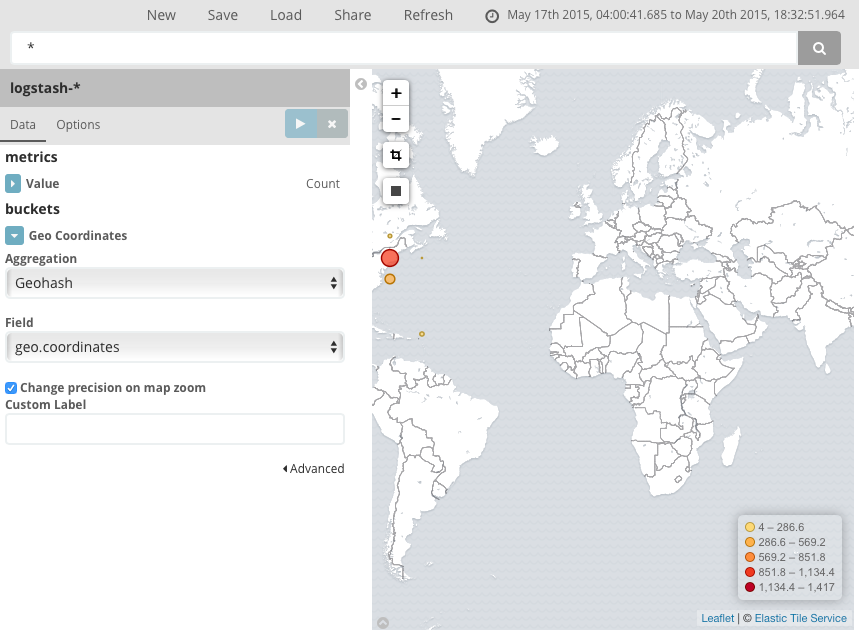
Next, we’re going to use a tile map chart to visualize geographic information in our log file sample data.

1. Click New.
2. Select Tile map.
3. Select the logstash-\* index pattern.
4. Set the time window for the events we’re exploring:
5. Click the time picker in the Kibana toolbar.
6. Click Absolute.
7. Set the start time to May 18, 2015 and the end time to May 20, 2015.
8. 
9. Once you’ve got the time range set up, click the Go button and close the time picker by clicking the small up arrow in the bottom right corner.

You’ll see a map of the world, since we haven’t defined any buckets yet:



To map the geo coordinates from the log files select Geo Coordinates as the bucket and click Apply changes images/apply-changes-button.png. Your chart should now look like this:



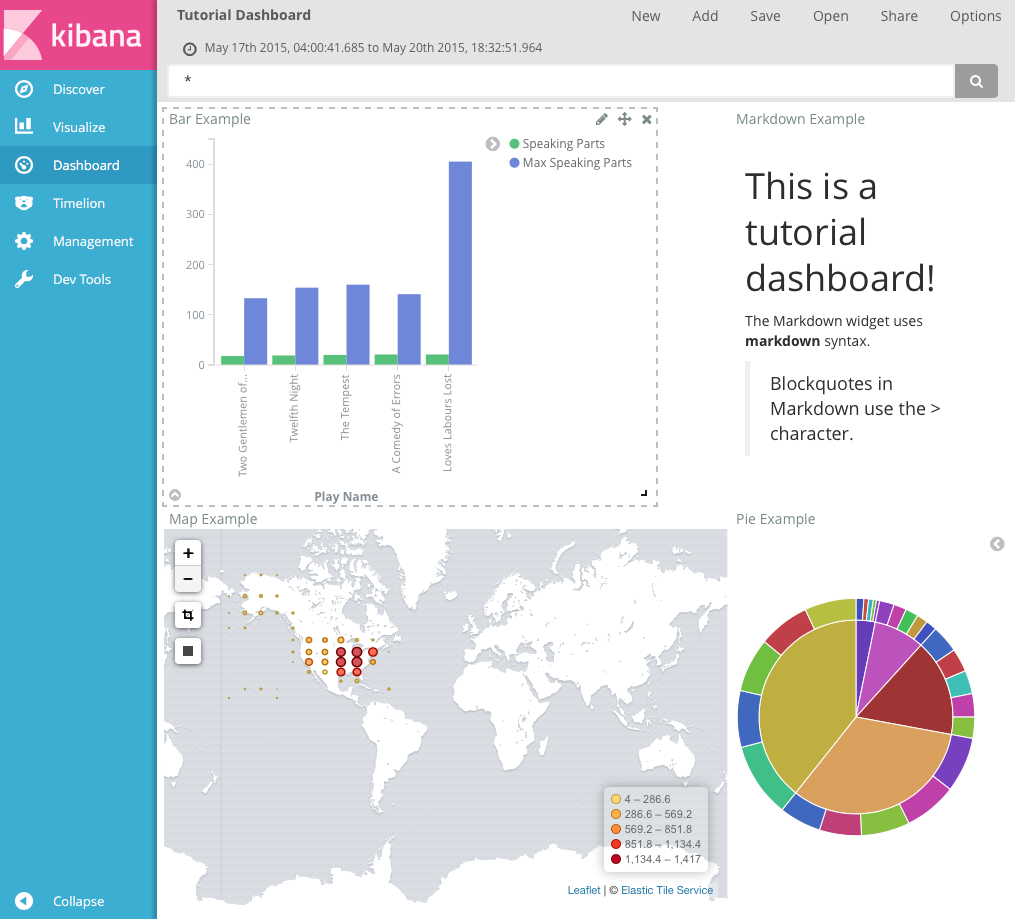
You can navigate the map by clicking and dragging, zoom with the images/viz-zoom.png buttons, or hit the Fit Data Bounds images/viz-fit-bounds.png button to zoom to the lowest level that includes all the points. You can also include or exclude a rectangular area by clicking the Latitude/Longitude Filter images/viz-lat-long-filter.png button and drawing a bounding box on the map. Applied filters are displayed below the query bar. Hovering over a filter displays controls to toggle, pin, invert, or delete the filter.

## Putting it all Together with Dashboards

A dashboard is a collection of visualizations that you can arrange and share. To build a dashboard that contains the visualizations you saved during this tutorial:

1. Click Dashboard in the side navigation.
2. Click Add to display the list of saved visualizations.
3. Click *Markdown Example*, *Pie Example*, *Bar Example*, and *Map Example*, then close the list of visualizations by clicking the small up-arrow at the bottom of the list.

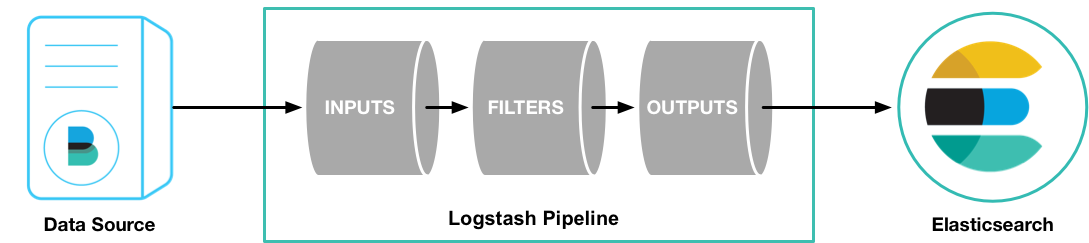
Hovering over a visualization displays the container controls that enable you to edit, move, delete, and resize the visualization. Your sample dashboard should end up looking roughly like this:



To get a link to share or HTML code to embed the dashboard in a web page, save the dashboard and click Share.

Logstash

Logstash is an open source tool for collecting, parsing, and storing logs for future use. A Logstash pipeline has two required elements, input and output, and one optional element,filter. The input plugins consume data from a source, the filter plugins modify the data as you specify, and the output plugins write the data to a destination.



# How Logstash Works

The Logstash event processing pipeline has three stages: inputs → filters → outputs. Inputs generate events, filters modify them, and outputs ship them elsewhere. Inputs and outputs support codecs that enable you to encode or decode the data as it enters or exits the pipeline without having to use a separate filter.

### Inputs

You use inputs to get data into Logstash. Some of the more commonly-used inputs are:

* file: reads from a file on the filesystem, much like the UNIX command tail -0F
* syslog: listens on the well-known port 514 for syslog messages and parses according to the RFC3164 format
* redis: reads from a redis server, using both redis channels and redis lists. Redis is often used as a "broker" in a centralized Logstash installation, which queues Logstash events from remote Logstash "shippers".
* beats: processes events sent by Filebeat.

### Filters

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

* grok: parse and structure arbitrary text. Grok is currently the best way in Logstash to parse unstructured log data into something structured and queryable. With 120 patterns built-in to Logstash, it’s more than likely you’ll find one that meets your needs!
* mutate: perform general transformations on event fields. You 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!)

### Outputs

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 you’re planning to save your data in an efficient, convenient, and easily queryable format… Elasticsearch is the way to go. Period. Yes, we’re biased :)
* file: write event data to a file on disk.
* graphite: send event data to graphite, a popular open source tool for storing and graphing metrics.
* statsd: send event data to statsd, a service that "listens for statistics, like counters and timers, sent over UDP and sends aggregates to one or more pluggable backend services". If you’re already using statsd, this could be useful for you!

### Codecs

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, msgpack, and plain (text).

* json: encode or decode data in the JSON format.
* multiline: merge multiple-line text events such as java exception and stacktrace messages into a single event.

## Logstash Configuration Examples

The following examples illustrate how you can configure Logstash to filter events, process Apache logs and syslog messages, and use conditionals to control what events are processed by a filter or output.

#### Configuring Filters

Filters are an in-line processing mechanism that provide the flexibility to slice and dice your data to fit your needs. Let’s take a look at some filters in action. The following configuration file sets up the grok and date filters.

input { stdin { } }  
  
filter {  
 grok {  
 match => { "message" => "%{COMBINEDAPACHELOG}" }  
 }  
 date {  
 match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]  
 }  
}  
  
output {  
 elasticsearch { hosts => ["localhost:9200"] }  
 stdout { codec => rubydebug }  
}

Run Logstash with this configuration:

bin/logstash -f logstash-filter.conf

Now, paste the following line into your terminal so it will be processed by the stdin input:

127.0.0.1 - - [11/Dec/2013:00:01:45 -0800] "GET /xampp/status.php HTTP/1.1" 200 3891 "http://cadenza/xampp/navi.php" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0"

You should see something returned to stdout that looks like this:

{  
 "message" => "127.0.0.1 - - [11/Dec/2013:00:01:45 -0800] \"GET /xampp/status.php HTTP/1.1\" 200 3891 \"http://cadenza/xampp/navi.php\" \"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0\"",  
 "@timestamp" => "2013-12-11T08:01:45.000Z",  
 "@version" => "1",  
 "host" => "cadenza",  
 "clientip" => "127.0.0.1",  
 "ident" => "-",  
 "auth" => "-",  
 "timestamp" => "11/Dec/2013:00:01:45 -0800",  
 "verb" => "GET",  
 "request" => "/xampp/status.php",  
 "httpversion" => "1.1",  
 "response" => "200",  
 "bytes" => "3891",  
 "referrer" => "\"http://cadenza/xampp/navi.php\"",  
 "agent" => "\"Mozilla/5.0 (Macintosh; Intel Mac OS X 10.9; rv:25.0) Gecko/20100101 Firefox/25.0\""  
}

As you can see, Logstash (with help from the grok filter) was able to parse the log line (which happens to be in Apache "combined log" format) and break it up into many different discrete bits of information. This is extremely useful once you start querying and analyzing our log data. For example, you’ll be able to easily run reports on HTTP response codes, IP addresses, referrers, and so on. There are quite a few grok patterns included with Logstash out-of-the-box, so it’s quite likely if you need to parse a common log format, someone has already done the work for you. For more information, see the list of Logstash grok patterns on GitHub.

The other filter used in this example is the date filter. This filter parses out a timestamp and uses it as the timestamp for the event (regardless of when you’re ingesting the log data). You’ll notice that the @timestamp field in this example is set to December 11, 2013, even though Logstash is ingesting the event at some point afterwards. This is handy when backfilling logs. It gives you the ability to tell Logstash "use this value as the timestamp for this event".

#### Processing Apache Logs

Let’s do something that’s actually useful: process apache2 access log files! We are going to read the input from a file on the localhost, and use a conditional to process the event according to our needs. First, create a file called something like *logstash-apache.conf* with the following contents (you can change the log’s file path to suit your needs):

input {  
 file {  
 path => "/tmp/access\_log"  
 start\_position => "beginning"  
 }  
}  
  
filter {  
 if [path] =~ "access" {  
 mutate { replace => { "type" => "apache\_access" } }  
 grok {  
 match => { "message" => "%{COMBINEDAPACHELOG}" }  
 }  
 }  
 date {  
 match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]  
 }  
}  
  
output {  
 elasticsearch {  
 hosts => ["localhost:9200"]  
 }  
 stdout { codec => rubydebug }  
}

Then, create the input file you configured above (in this example, "/tmp/access\_log") with the following log entries (or use some from your own webserver):

71.141.244.242 - kurt [18/May/2011:01:48:10 -0700] "GET /admin HTTP/1.1" 301 566 "-" "Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.9.2.3) Gecko/20100401 Firefox/3.6.3"  
134.39.72.245 - - [18/May/2011:12:40:18 -0700] "GET /favicon.ico HTTP/1.1" 200 1189 "-" "Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 5.1; Trident/4.0; .NET CLR 2.0.50727; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; InfoPath.2; .NET4.0C; .NET4.0E)"  
98.83.179.51 - - [18/May/2011:19:35:08 -0700] "GET /css/main.css HTTP/1.1" 200 1837 "http://www.safesand.com/information.htm" "Mozilla/5.0 (Windows NT 6.0; WOW64; rv:2.0.1) Gecko/20100101 Firefox/4.0.1"

Now, run Logstash with the -f flag to pass in the configuration file:

bin/logstash -f logstash-apache.conf

Now you should see your apache log data in Elasticsearch! Logstash opened and read the specified input file, processing each event it encountered. Any additional lines logged to this file will also be captured, processed by Logstash as events, and stored in Elasticsearch. As an added bonus, they are stashed with the field "type" set to "apache\_access" (this is done by the type ⇒ "apache\_access" line in the input configuration).

In this configuration, Logstash is only watching the apache access\_log, but it’s easy enough to watch both the access\_log and the error\_log (actually, any file matching \*log), by changing one line in the above configuration:

input {  
 file {  
 path => "/tmp/\*\_log"  
...

When you restart Logstash, it will process both the error and access logs. However, if you inspect your data (using elasticsearch-kopf, perhaps), you’ll see that the access\_log is broken up into discrete fields, but the error\_log isn’t. That’s because we used a grok filter to match the standard combined apache log format and automatically split the data into separate fields. Wouldn’t it be nice if we could control how a line was parsed, based on its format? Well, we can…

Note that Logstash did not reprocess the events that were already seen in the access\_log file. When reading from a file, Logstash saves its position and only processes new lines as they are added. Neat!

#### Using Conditionals

You use conditionals to control what events are processed by a filter or output. For example, you could label each event according to which file it appeared in (access\_log, error\_log, and other random files that end with "log").

input {  
 file {  
 path => "/tmp/\*\_log"  
 }  
}  
  
filter {  
 if [path] =~ "access" {  
 mutate { replace => { type => "apache\_access" } }  
 grok {  
 match => { "message" => "%{COMBINEDAPACHELOG}" }  
 }  
 date {  
 match => [ "timestamp" , "dd/MMM/yyyy:HH:mm:ss Z" ]  
 }  
 } else if [path] =~ "error" {  
 mutate { replace => { type => "apache\_error" } }  
 } else {  
 mutate { replace => { type => "random\_logs" } }  
 }  
}  
  
output {  
 elasticsearch { hosts => ["localhost:9200"] }  
 stdout { codec => rubydebug }  
}

This example labels all events using the type field, but doesn’t actually parse the error or random files. There are so many types of error logs that how they should be labeled really depends on what logs you’re working with.

Similarly, you can use conditionals to direct events to particular outputs. For example, you could:

* alert nagios of any apache events with status 5xx
* record any 4xx status to Elasticsearch
* record all status code hits via statsd

To tell nagios about any http event that has a 5xx status code, you first need to check the value of the type field. If it’s apache, then you can check to see if the status field contains a 5xx error. If it is, send it to nagios. If it isn’t a 5xx error, check to see if the status field contains a 4xx error. If so, send it to Elasticsearch. Finally, send all apache status codes to statsd no matter what the status field contains:

output {  
 if [type] == "apache" {  
 if [status] =~ /^5\d\d/ {  
 nagios { ... }  
 } else if [status] =~ /^4\d\d/ {  
 elasticsearch { ... }  
 }  
 statsd { increment => "apache.%{status}" }  
 }  
}

#### Processing Syslog Messages

Syslog is one of the most common use cases for Logstash, and one it handles exceedingly well (as long as the log lines conform roughly to RFC3164). Syslog is the de facto UNIX networked logging standard, sending messages from client machines to a local file, or to a centralized log server via rsyslog. For this example, you won’t need a functioning syslog instance; we’ll fake it from the command line so you can get a feel for what happens.

First, let’s make a simple configuration file for Logstash + syslog, called *logstash-syslog.conf*.

input {  
 tcp {  
 port => 5000  
 type => syslog  
 }  
 udp {  
 port => 5000  
 type => syslog  
 }  
}  
  
filter {  
 if [type] == "syslog" {  
 grok {  
 match => { "message" => "%{SYSLOGTIMESTAMP:syslog\_timestamp} %{SYSLOGHOST:syslog\_hostname} %{DATA:syslog\_program}(?:\[%{POSINT:syslog\_pid}\])?: %{GREEDYDATA:syslog\_message}" }  
 add\_field => [ "received\_at", "%{@timestamp}" ]  
 add\_field => [ "received\_from", "%{host}" ]  
 }  
 date {  
 match => [ "syslog\_timestamp", "MMM d HH:mm:ss", "MMM dd HH:mm:ss" ]  
 }  
 }  
}  
  
output {  
 elasticsearch { hosts => ["localhost:9200"] }  
 stdout { codec => rubydebug }  
}

Run Logstash with this new configuration:

bin/logstash -f logstash-syslog.conf

Normally, a client machine would connect to the Logstash instance on port 5000 and send its message. For this example, we’ll just telnet to Logstash and enter a log line (similar to how we entered log lines into STDIN earlier). Open another shell window to interact with the Logstash syslog input and enter the following command:

telnet localhost 5000

Copy and paste the following lines as samples. (Feel free to try some of your own, but keep in mind they might not parse if the grok filter is not correct for your data).

Dec 23 12:11:43 louis postfix/smtpd[31499]: connect from unknown[95.75.93.154]  
Dec 23 14:42:56 louis named[16000]: client 199.48.164.7#64817: query (cache) 'amsterdamboothuren.com/MX/IN' denied  
Dec 23 14:30:01 louis CRON[619]: (www-data) CMD (php /usr/share/cacti/site/poller.php >/dev/null 2>/var/log/cacti/poller-error.log)  
Dec 22 18:28:06 louis rsyslogd: [origin software="rsyslogd" swVersion="4.2.0" x-pid="2253" x-info="http://www.rsyslog.com"] rsyslogd was HUPed, type 'lightweight'.

Now you should see the output of Logstash in your original shell as it processes and parses messages!

{  
 "message" => "Dec 23 14:30:01 louis CRON[619]: (www-data) CMD (php /usr/share/cacti/site/poller.php >/dev/null 2>/var/log/cacti/poller-error.log)",  
 "@timestamp" => "2013-12-23T22:30:01.000Z",  
 "@version" => "1",  
 "type" => "syslog",  
 "host" => "0:0:0:0:0:0:0:1:52617",  
 "syslog\_timestamp" => "Dec 23 14:30:01",  
 "syslog\_hostname" => "louis",  
 "syslog\_program" => "CRON",  
 "syslog\_pid" => "619",  
 "syslog\_message" => "(www-data) CMD (php /usr/share/cacti/site/poller.php >/dev/null 2>/var/log/cacti/poller-error.log)",  
 "received\_at" => "2013-12-23 22:49:22 UTC",  
 "received\_from" => "0:0:0:0:0:0:0:1:52617",  
 "syslog\_severity\_code" => 5,  
 "syslog\_facility\_code" => 1,  
 "syslog\_facility" => "user-level",  
 "syslog\_severity" => "notice"  
}