What is the ELK Stack?

The ELK Stack is a collection of three open-source products — [Elasticsearch](https://logz.io/category/blog/elasticsearch/), [Logstash](https://logz.io/category/blog/logstash/), and [Kibana](https://logz.io/category/blog/kibana/)— from [Elastic](https://www.elastic.co/). 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?

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, how do Netflix, Facebook, Microsoft, LinkedIn, and Cisco monitor their logs? [With ELK](https://logz.io/product/).

### Why is Log Analysis Becoming More Important?

As more and more IT infrastructures move to public clouds such as [Amazon Web Services](https://logz.io/category/blog/amazon-web-services/) and Microsoft Azure, [public cloud security tools](http://devops.com/2015/12/30/guide-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.

### ****The Challenge of Disparate Log Data****

If you're like most of us, your log data universe is inconsistent, inaccessible, and inconsistent. We understand problems such as these quite well:

**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
* SSH + GREP doesn’t scale or reach

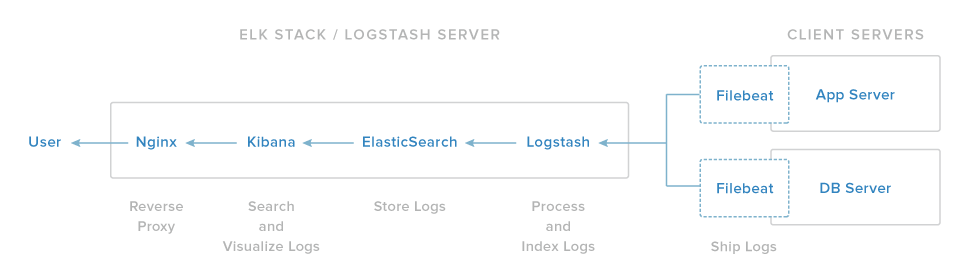
**Accessibility of Log Data —** Much of the data is difficult to locate and manage. Although some of the log data may be highly valuable, many admins face these steep challenges:

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

The ELK stack can help you 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.

Our ELK stack setup has four main components:

* **Logstash**: The server component of Logstash that processes incoming logs
* **Elasticsearch**: Stores all of the logs
* **Kibana**: Web interface for searching and visualizing logs
* **Filebeat**: Installed on client servers that will send their logs to Logstash, Filebeat serves as a log shipping agent that utilizes the *lumberjack* networking protocol to communicate with Logstash



## Elasticsearch

Elasticsearch is often described as a search server. That might be confusing because we usually think of search as something that we do, not something that needs to be served. However, the reality is that search can be quite complex, and search servers have been developed in response to that fact.

**Elasticsearch is a NoSQL database that is based on the Lucene search engine.**. That means it stores data in an unstructured way and that you 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.

**Real-time data and real-time analytics.** The ELK stack gives you the power of real-time data insights, with the ability to perform super-fast 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, multi-tenant.** With Elasticsearch, you can start small and expand it along with your business growth-when you are ready. 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 full-text search capabilities available in any open-source product. The search features come with multi-language support, an extensive query language, geolocation support, and context-sensitive suggestions, and autocompletion.

**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. Understanding these concepts from the outset will tremendously help ease the learning process.

### Near Realtime (NRT)

Elasticsearch is a near real time search platform. What this means is there is a slight latency (normally one second) from the time you index a document until the time it becomes searchable.

### Cluster

A cluster is a collection of one or more nodes (servers) that together holds your entire data and provides federated indexing and search capabilities across all nodes. A cluster is identified by a unique name which by default is "elasticsearch". This name is important because a node can only be part of a cluster if the node is set up to join the cluster by its name.

### Node

A node is a single server that is part of your cluster, stores your data, and participates in the cluster’s indexing and search capabilities. Just like a cluster, a node is identified by a name which by default is a random Universally Unique IDentifier (UUID) that is assigned to the node at startup. You can define any node name you want if you do not want the default. This name is important for administration purposes where you want to identify which servers in your network correspond to which nodes in your Elasticsearch cluster.

A node can be configured to join a specific cluster by the cluster name. By default, each node is set up to join a cluster named elasticsearch which means that if you start up a number of nodes on your network and—assuming they can discover each other—they will all automatically form and join a single cluster named elasticsearch.

In a single cluster, you can have as many nodes as you want. Furthermore, if there are no other Elasticsearch nodes currently running on your network, starting a single node will by default form a new single-node cluster named elasticsearch.

### Index

An index is a collection of documents that have somewhat similar characteristics. For example, you can 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, you can define as many indexes as you want.

### Type

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.

### Document

A document is a basic unit of information that can be indexed. For example, you can have a document for a single customer, another document for a single product, and yet another for a single order. This document is expressed in [JSON](http://json.org/) (JavaScript Object Notation) which is an ubiquitous internet data interchange format.

Within an index/type, you can store as many documents as you want. Note that although a document physically resides in an index, a document actually must be indexed/assigned to a type inside an index.

### Shards & Replicas

An index can potentially store a large amount of data that can exceed the hardware limits of a single node. For example, a single index of a billion documents taking up 1TB of disk space may not fit on the disk of a single node or may be too slow to serve search requests from a single node alone.

To solve this problem, Elasticsearch provides the ability to subdivide your index into multiple pieces called shards. When you create an index, you can simply define the number of shards that you want. Each shard is in itself a fully-functional and independent "index" that can be hosted on any node in the cluster.

Data In, Data Out

Whatever program we write, the intention is the same: to organize data in a way that serves our purposes. But data doesn’t consist of just random bits and bytes. We build relationships between data elements in order to represent entities, or *things* that exist in the real world. A name and an email address have more meaning if we know that they belong to the same person.

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 person might have three email addresses, while another has none. A Spanish person will probably have two last names, while an English person will probably have only one.

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. So far, so good.

The problem comes when we need to store these entities. Traditionally, we have stored our data in columns and rows in a relational database, the equivalent of using a spreadsheet. All the flexibility gained from using objects 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 spreadsheets, we can instead focus on *using* the data. The flexibility of objects is returned to us.

An *object* is a language-specific, in-memory data structure. To send it across the network or store it, we need to be able to represent it in some standard format. [JSON](http://en.wikipedia.org/wiki/Json) is a way of representing objects in human-readable text. It has become the de facto standard for exchanging data 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 retrieved from any node in the cluster.

Of course, we don’t need to only store data; we must also query it, en masse and at speed. While NoSQL solutions exist that allow us to store objects as documents, they still require us 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.

## How Elasticsearch stores Data

Sharding is important for two primary reasons:

* It allows you to horizontally split/scale your content volume
* It allows you 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 you 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. To this end, Elasticsearch allows you to make one or more copies of your index’s shards into what are called replica shards, or replicas for short.

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 you to scale out your search volume/throughput since searches can be executed on all replicas in parallel.

To summarize, each index can be split into multiple shards. An index can also be replicated zero (meaning no replicas) or more times. Once replicated, each index will have primary shards (the original shards that were replicated from) and replica shards (the copies of the primary shards). The number of shards and replicas can be defined per index at the time the index is created. After the index is created, you may 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 primary shards and another 5 replica shards (1 complete replica) for a total of 10 shards per index.

Each Elasticsearch shard is a Lucene index. There is a maximum number of documents you can have in a single Lucene index. As of [LUCENE-5843](https://issues.apache.org/jira/browse/LUCENE-5843), the limit is 2,147,483,519 (= Integer.MAX\_VALUE - 128) documents. You can monitor shard sizes using the [\_cat/shards](https://www.elastic.co/guide/en/elasticsearch/reference/current/cat-shards.html) api.

\

**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 you to pull data from a wide variety of sources, it also gives you the tools to filter, massage, and shape the data so that it’s easier to work with.

### How to Configure Logstash

# How Logstash Works

The Logstash event processing pipeline has three stages: inputs → filters → outpu ts. 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.

### Plugins

A self-contained software package that implements one of the stages in the Logstash event processing[pipeline](https://www.elastic.co/guide/en/logstash/5.x/glossary.html#glossary-pipeline). The list of available plugins includes [input plugins](https://www.elastic.co/guide/en/logstash/5.x/glossary.html#glossary-input-plugin), [output plugins](https://www.elastic.co/guide/en/logstash/5.x/glossary.html#glossary-output-plugin), [codec plugins](https://www.elastic.co/guide/en/logstash/5.x/glossary.html#glossary-codec-plugin), and [filter plugins](https://www.elastic.co/guide/en/logstash/5.x/glossary.html#glossary-filter-plugin). The plugins are implemented as Ruby [gems](https://www.elastic.co/guide/en/logstash/5.x/glossary.html#glossary-gem) and hosted on [RubyGems.org](https://rubygems.org/). You define the stages of an event processing [pipeline](https://www.elastic.co/guide/en/logstash/5.x/glossary.html#glossary-pipeline) by configuring plugins.

### 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
* **beats**: processes events sent by [Filebeat](https://www.elastic.co/downloads/beats/filebeat" \t "_top).

For more information about the available inputs, see [Input Plugins](https://www.elastic.co/guide/en/logstash/current/input-plugins.html).

### 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!)

For more information about the available filters, see [Filter Plugins](https://www.elastic.co/guide/en/logstash/current/filter-plugins.html).

### 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.
* **stdout**: send event data to graphite, a popular open source tool for storing and graphing metrics. http://graphite.readthedocs.io/en/latest/

### 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).

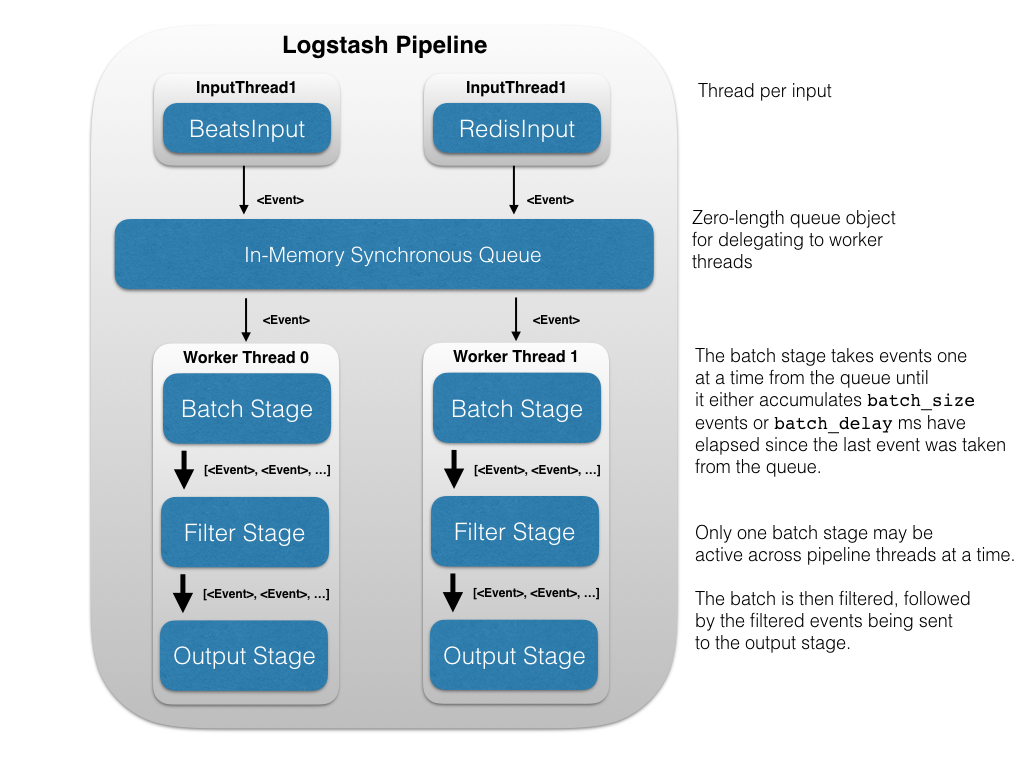
* **json**: encode or decode data in the JSON format.
* **multiline**: merge multiple-line text events such as java exception and stacktrace messages into a single event.

shipper

An instance of Logstash that send events to another instance of Logstash, or some other application.

worker

The filter thread model used by Logstash, where each worker receives an [event](https://www.elastic.co/guide/en/logstash/5.x/glossary.html#glossary-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.



Rather than normalizing with time-sucking ETL (Extract, Transform, and Load), we recommend that you switch over to the fast track. Instead, you could spend much less time training Logstashto 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, you can grab hold of a very powerful feature-set with a very little effort on your part.

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—BAM!—you get super-fast queries against your mountains of data.

## Execution Model[edit](https://github.com/elastic/logstash/edit/5.x/docs/static/life-of-an-event.asciidoc)

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](https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/SynchronousQueue.html" \t "_top). 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 are configurable (see [Tuning and Profiling Logstash Performance](https://www.elastic.co/guide/en/logstash/current/tuning-logstash.html)).

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. See [Persistent Queues](https://www.elastic.co/guide/en/logstash/current/persistent-queues.html) for more information.

# Data Resiliency[edit](https://github.com/elastic/logstash/edit/master/docs/static/resiliency.asciidoc)

As data flows through the event processing pipeline, Logstash may encounter situations that prevent it from delivering events to the configured output. For example, the data might contain unexpected data types, or Logstash might terminate abnormally.

To guard against data loss and ensure that events flow through the pipeline without interruption, Logstash provides the following data resiliency features.

* [Persistent Queues](https://www.elastic.co/guide/en/logstash/5.x/persistent-queues.html) protect against data loss by storing events in an internal queue on disk.
* [Dead Letter Queues](https://www.elastic.co/guide/en/logstash/5.x/dead-letter-queues.html) provide on-disk storage for events that Logstash is unable to process. You can easily reprocess events in the dead letter queue by using the dead\_letter\_queueinput plugin.

These resiliency features are disabled by default. To turn on these features, you must explicitly enable them in the Logstash [settings file](https://www.elastic.co/guide/en/logstash/5.x/logstash-settings-file.html).

# Kibana

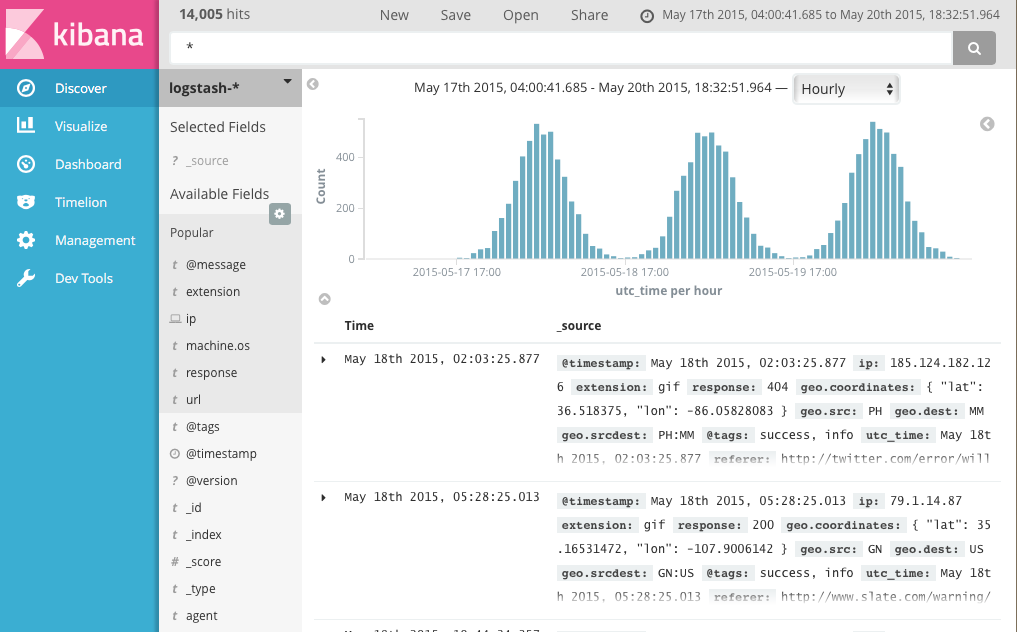
Kibana is an open source analytics and visualization platform designed to work with Elasticsearch. You use Kibana to search, view, and interact with data stored in Elasticsearch indices. You can easily perform advanced data analysis and visualize your data in a variety of charts, tables, and maps.

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 a snap. You can install Kibana and start exploring your Elasticsearch indices in minutes — no code, no additional infrastructure required.

Discovering Your Data[edit](https://github.com/elastic/kibana/edit/5.5/docs/getting-started/tutorial-discovering.asciidoc)

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



In the query bar, you can enter an [Elasticsearch query](https://www.elastic.co/guide/en/elasticsearch/reference/5.x/query-dsl-query-string-query.html" \l "query-string-syntax" \t "_top) 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.

## Visualizing Your Data[edit](https://github.com/elastic/kibana/edit/5.5/docs/getting-started/tutorial-visualizing.asciidoc)

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 the big blue **Create a visualization** button in the center of the screen.

When you define an index pattern, indices that match that pattern must exist in Elasticsearch. Those indices must contain data.