ObjectRocket
a rackspace company

Elasticsearch Jetpack

Everything you need to know about Elasticsearch and why DBaaS for Elasticsearch will save you time, money, and headaches.

What's Inside

Intro

Few databases have performed as well or have become as truly industry-changing as Elasticsearch. The increase in popularity of Elasticsearch over the past few years is staggering.

Read on to learn everything you need to know about Elasticsearch and why DBaaS for Elasticsearch will save you time, money, and headaches.

What is Elasticsearch?

A distributed, real-time, search engine / document store.

Distributed: Easily scales to multiple servers and TBs of data.

Real-time: Data is searchable as it is added.

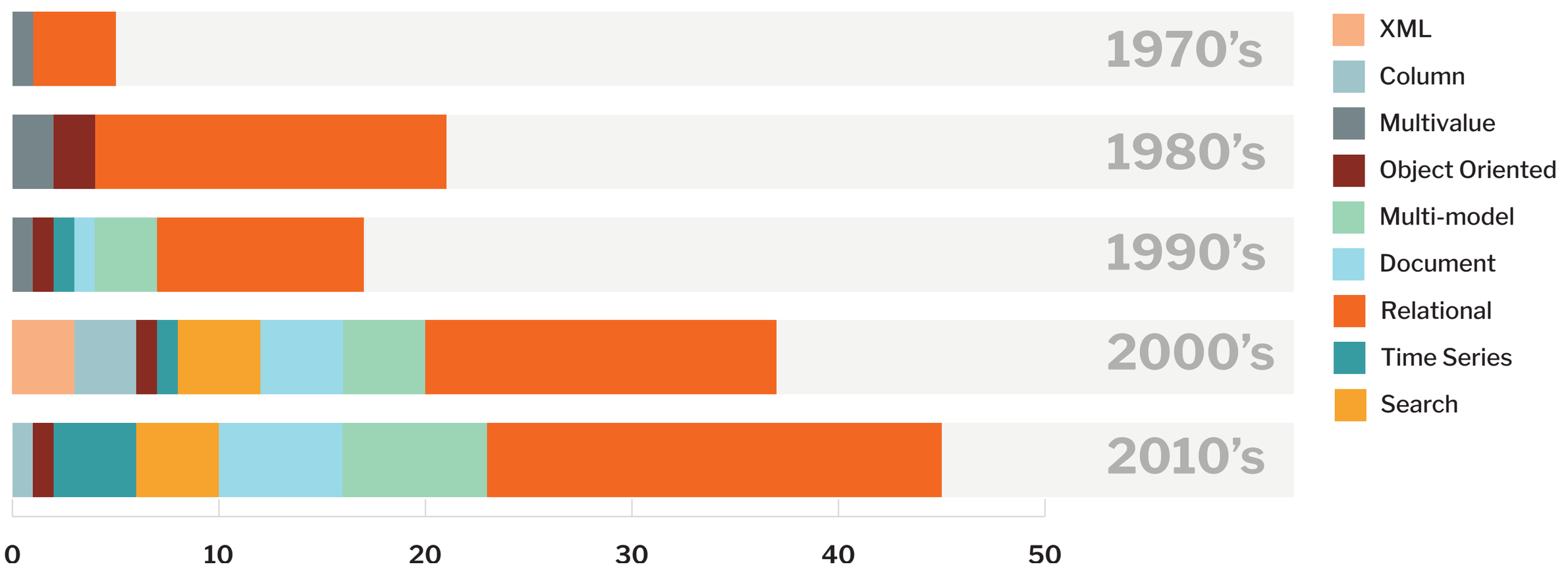
Search engine: Best full text search capabilities of any technology in this space.

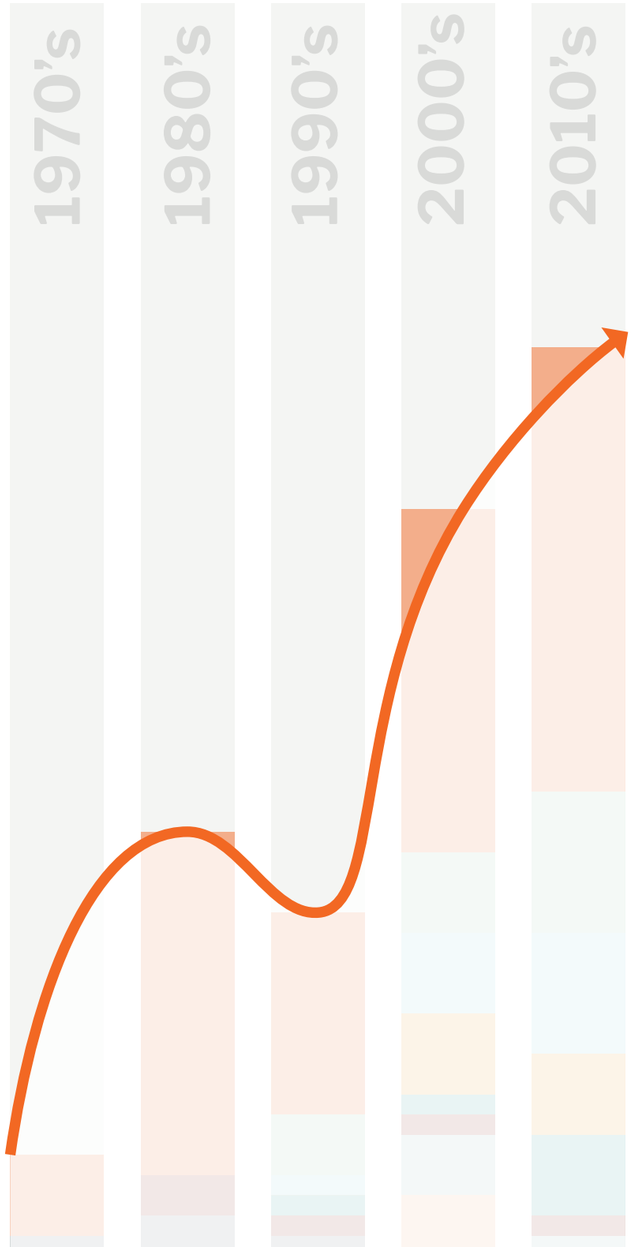
Document store: Less strict schema or SQL-like rigidity. Very JSON-friendly.

Read on to learn about why you should use Elasticsearch.

Databases over time

There is no shortage of stories or analyses about the NoSQL movement and how it's driving a lot of expansion and change in the database space. An even more interesting trend is the increase in both database technologies and database types over the past couple of decades. Using DB-Engines and a few other online resources, **we've mapped out a timeline of database technologies so you can see when each was introduced:**





There are two major trends :

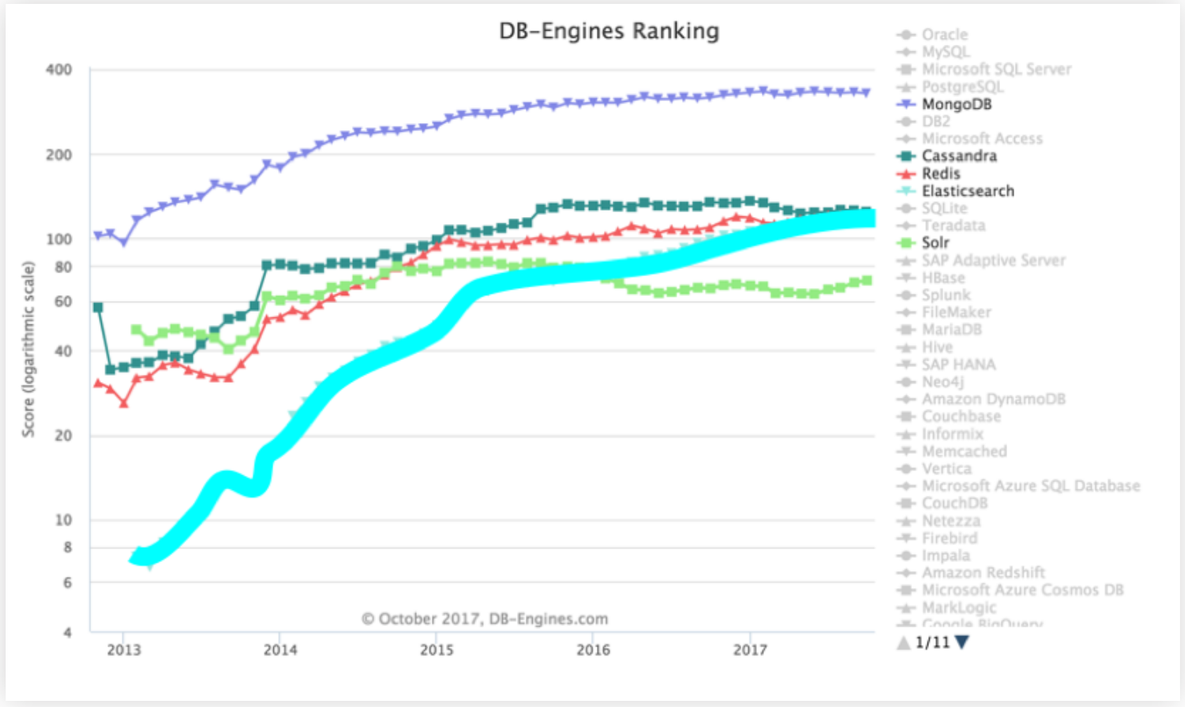
1 The number of databases introduced every year has soared in the last two decades.

2 The number of different categories (or types) of databases has also escalated.

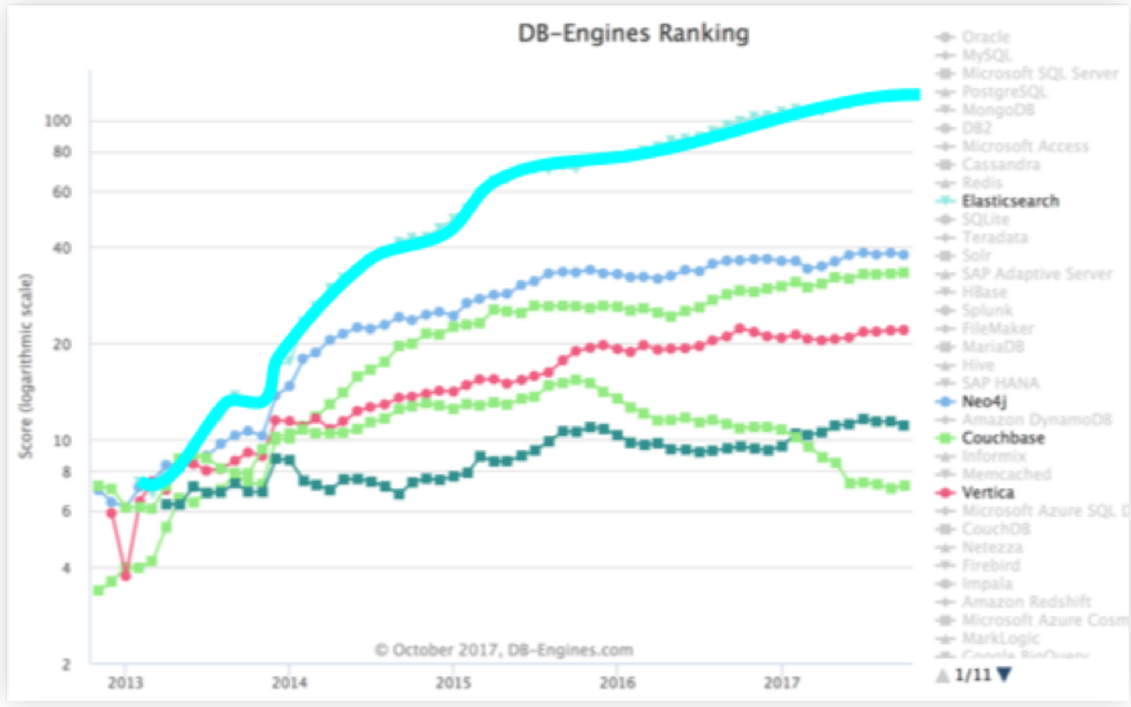
The first of these two trends is not terribly surprising. There have been many evolutionary changes in computing with the internet and open source software over the past 20 years, so one would expect a similar growth in databases. Sure enough, the total number of databases (and open source databases) has continued to grow as people develop new technologies to solve a completely new set of challenges.

This brings us to the second trend. As new challenges have risen with data, smart technologies have been created to fulfill that need. Internet services and big data have driven growth in technologies like Hadoop and NoSQL. Now we see technologies like IoT driving growth in streaming and time-series databases.

Out of this growing mess of databases, a few have outperformed and become truly industry-changing. One of those is Elasticsearch. **The increase in popularity of Elasticsearch over the past few years is staggering and can best be summed up by another few charts from DB-Engines:**



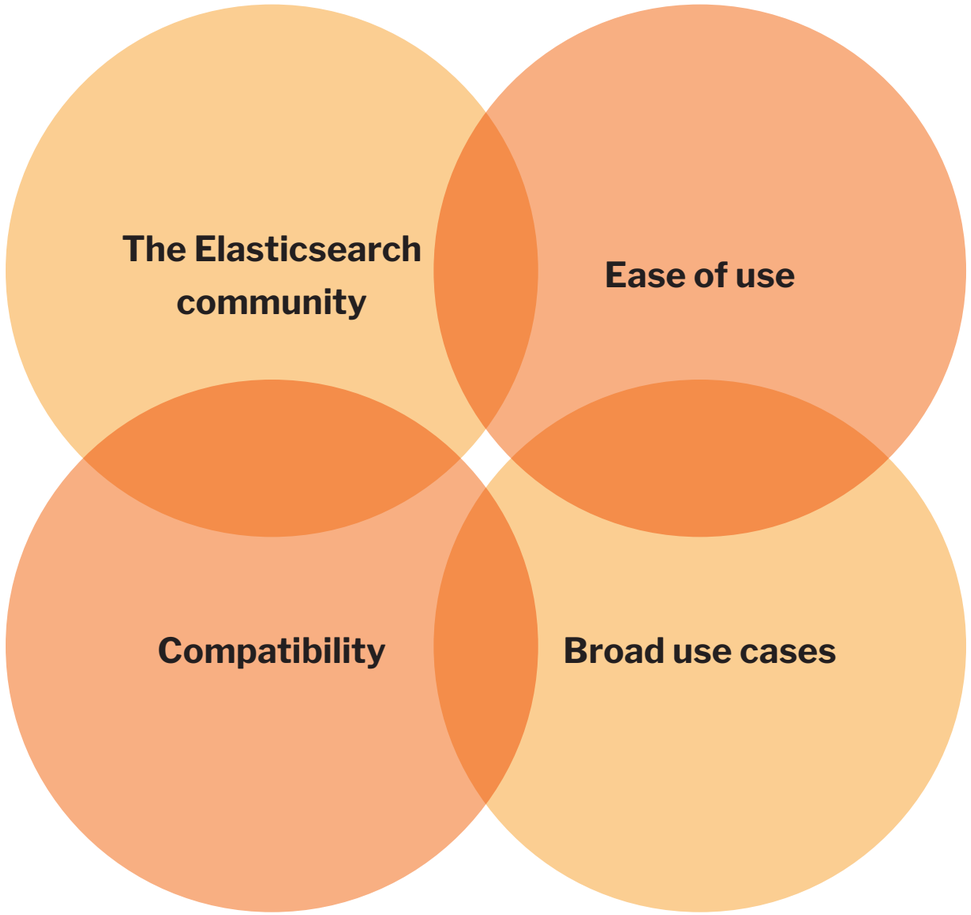
In the first image, you can see that Elasticsearch has risen from relative obscurity in 2013 to surpassing it's closest rival.



Now you can see that it is jumping into the top 3 or 4 non-relational data stores. If you compare that against other technologies that had roughly the same popularity in 2013 (the second image), you will notice how much Elasticsearch has surpassed them.

Elasticsearch strikes back

How did Elasticsearch outperform competitors and other rising data stores in this time period?
This rapid adoption can be attributed to a few main areas:



These areas all build on each other. The great focus that the product and its creators have placed on making it easy to bootstrap and use has led to a quickly growing community of people and products. This active community has created a massive number of clients, plugins, and additional technologies to ensure that Elasticsearch is compatible with everything. The fact that Elasticsearch is compatible with so many technologies has allowed the use cases for Elasticsearch to rapidly expand.

Looking at the timeline of the Elasticsearch community's launches and acquisitions over the past few years, you see a core focus on search, then expansion into logging with Logstash, then added visualization with Elasticsearch Kibana, expansion into analytics with aggregations, then into monitoring with Beats. That is a huge amount of use case expansion in a small period of time.

All of these features have been getting more established and more robust over time. Meanwhile Elastic.co, as a company, has really focused on beefing up a lot of these capabilities with their acquisitions of Found, Packetbeat, Prelert, Opbeat, and most recently, Swifttype. This doubling down and moving up the stack in the key use case areas is key to Elastic's success.

Elasticsearch Implementation Methods

All of this expansion has created a few interesting adoption models for Elasticsearch. Let's review some of these adoption models.

Adoption Models

Main datastore

“I want to build an app/feature, so I choose Elasticsearch as the datastore.”

This first model is pretty tried and true, and reflects the usual method of database adoption.

Complementary technology

“I want to add visualization to SQL/ MongoDB/etc, add indexing to Hadoop, add processing/storage to Kafka, etc.”

This model covers a fairly untapped market. There have been many companies selling enhancements to databases, but none come to mind that can handle so many disparate uses with the same technology.

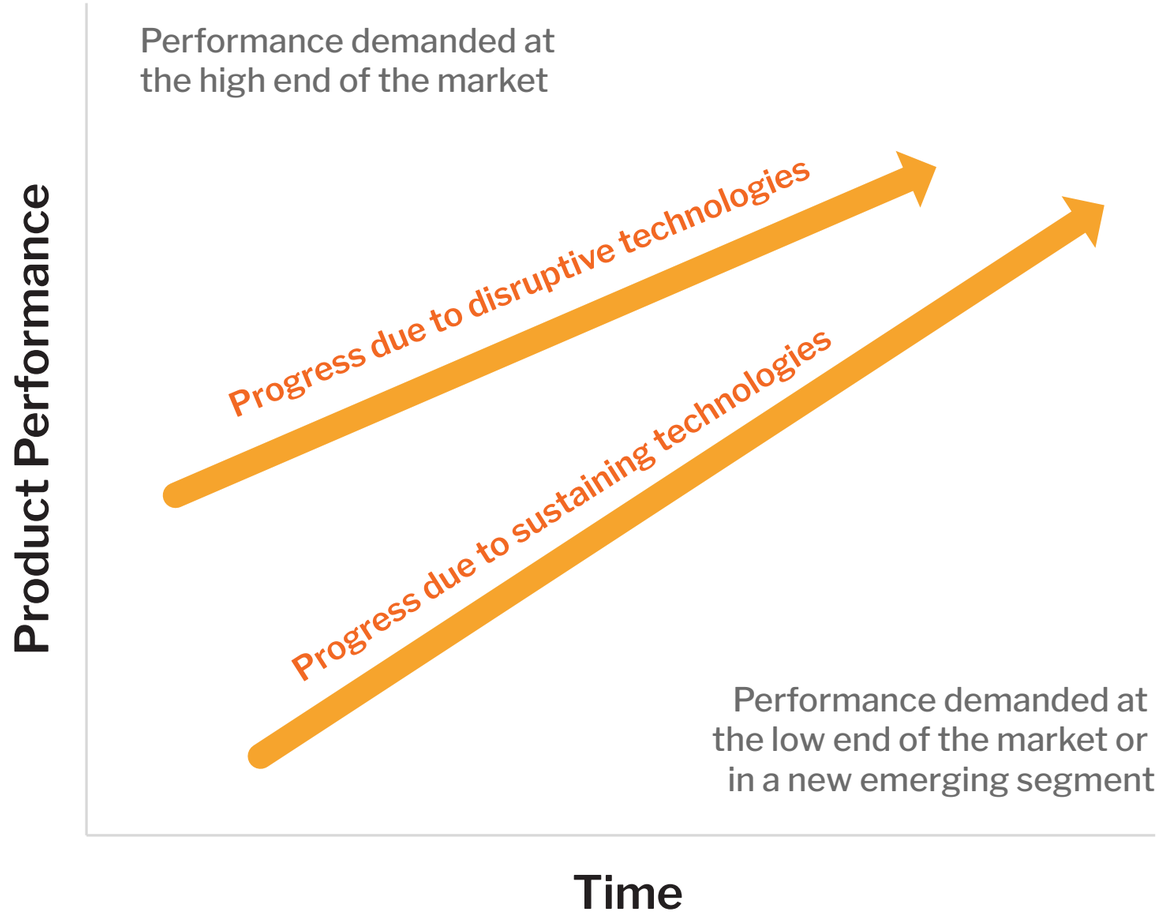
Additive technology

“I’m already shoving my logs into Elasticsearch, why not add metrics, monitoring, etc.”

This final, additive technology path, is the most disruptive thing Elasticsearch does. Elasticsearch can get in the door as either an add-on, or small part of the app. From there, it grows exponentially as people get comfortable with it and see how flexible it is.

The Innovator's Dilemma

Elasticsearch challenges many other players in the market. The technology very closely matches the disruptive technology model from "The Innovator's Dilemma". We see Elasticsearch meeting the needs of many basic uses and quickly working up the stack to add capabilities that nip at the heels of bigger players in the monitoring, log analysis, analytics, and enterprise search markets. Not only is Elasticsearch doing this in one use case, it's doing it in multiple areas.



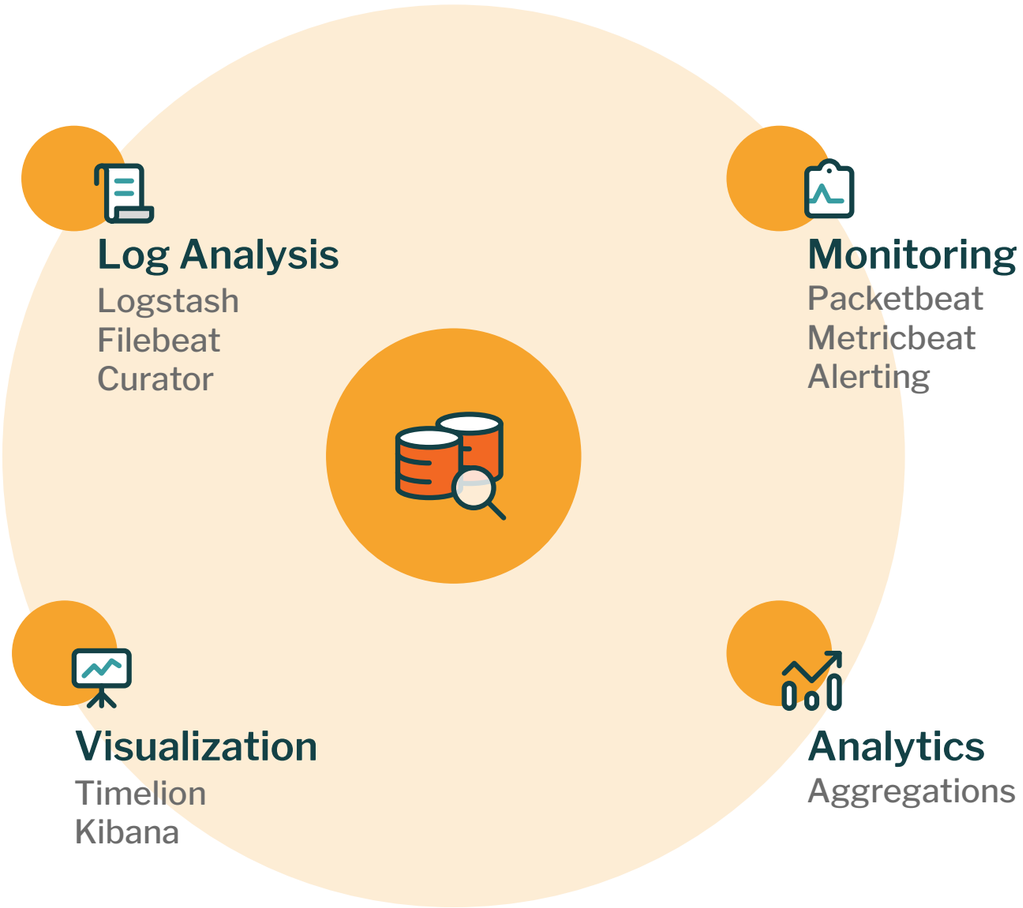
Innovator's Dilemma.
Disruptive vs. Sustaining Technologies: Accessed 10/24/17.
http://web.mit.edu/6.933/www/Fall2000/teradyne/clay.html

Who Uses Elasticsearch?

Customers across industries enjoy Elasticsearch.

Elasticsearch is becoming a disruptive “utility” database.

Elasticsearch excels in a very niche area: search. The key to the growth is their expansion of use cases outward over time. It all started with Logstash and Kibana and now they’re doubling down and seriously improving analytics and monitoring capabilities. One of the most compelling aspects of Elasticsearch is its ability to cover up the shortcomings of other databases and its ability to play as an add-on vs. a replacement. (For example, search for Hadoop, visualization for Mongo, etc.)



Top 5 Elasticsearch Use Cases

Other than “You Know, for Search”, the uses of Elasticsearch continue to grow and change over time. We at ObjectRocket have been offering hosted Elasticsearch on the ObjectRocket platform for a while now and have been able to see some clear trends among our customers and how they’re using the product.

Read on to read about the 5 use cases that we see on our platform.

1

Logging and Log Analysis

For anyone familiar with Elasticsearch, this one should be no surprise. The ecosystem built up around Elasticsearch has made it one of the easiest to implement and scale logging solutions. Many of the users on our platform are no different and have taken advantage of this to either add logging to their main use case, or are using us purely for logging.

From Beats, to Logstash, to Ingest Nodes, Elasticsearch gives you plenty of options for grabbing data wherever it lives and getting it indexed. From there, tools like Kibana give you the ability to create rich dashboards and analysis, while Curator allows you to put the retention period on autopilot.

2

Scraping and Combining Public Data

Like log data, the Elastic Stack has plenty of tools to make grabbing and indexing remote data easy. Also, like most document stores, the lack of a strict schema gives Elasticsearch the flexibility to take in multiple different sources of data and still keep it all manageable and searchable. A cool example of this that you can check out is our Twitter connector, which allows you to set up hashtags to watch on Twitter and then grab all tweets with those hashtags and analyze them in Kibana. We built that product on core Elastic Stack components and added some additional pieces to help it scale.

3

Full Text Search

It's also no surprise that full text search, as the core capability of Elasticsearch, is high on this list. The surprising part is the applications of this among our customer set, which go well beyond traditional Enterprise search or E-commerce. From fraud detection/security to collaboration and beyond, our users have shown that Elasticsearch's search capabilities are powerful, flexible, and include a great number of tools to make search easier; Elasticsearch has its own query DSL as well as built in capabilities for auto-complete, "Did you mean" responses, and more.

4

Event Data and Metrics

Elasticsearch also operates really well on time-series data like metrics and application events. This is another area where the huge Beats ecosystem allows you to easily grab data for common applications. Whatever technologies you use, there's a pretty good chance that Elasticsearch has the components to grab metrics and events out of the box... and in the rare case that it can't, adding that capability is really easy.

5

Visualizing Data

With tons of charting options, a tile service for geo-data, and TimeLion for time-series data, Kibana is an amazingly powerful and easy to use visualization tool. For every use case above there is some visual component handled by Kibana. Once you're comfortable with the various data ingest tools, you'll find that Elasticsearch + Kibana will become your go-to tool for visualizing data that you're trying to wrap your head around.

Conclusion

We haven't included every use case. Elasticsearch and the rest of the Elastic Stack have proven to be extremely versatile. There are multiple ways to integrate Elasticsearch into what you're doing today and gain extra insight. That's the coolest part of Elasticsearch: the ability to enhance the technologies you're already using rather than just another database to store your data.

ObjectRocket can help you set up a new Elasticsearch instance or help you configure your current one for the support your applications need.

SCHEDULE A CONSULTATION

Where Elasticsearch and Solr Differ

Now that businesses have shifted toward “big data,” people want access to more information than ever before, which means more new databases. It’s become absolutely crucial for businesses to harness “big data” to make informed decisions about customers and their needs based on data analysis.

One of the challenges businesses face when using databases is getting the information they need out of it. All of the data in the world isn’t going to do much to help you unless you can draw insights from your data and effectively search that data. That’s where tools like Elasticsearch and Solr come in. Read on to find out which of these tools is best suited to your needs.

It all starts with Apache Lucene

Solr and Elasticsearch share a common heritage; Both were created to provide a high-level search engine built on Apache Lucene. Lucene is an extremely powerful search library, but is difficult to use for newcomers and doesn't provide a stand-alone search application with REST APIs and more. Elasticsearch and Solr fill in those gaps and provide a whole lot more. Both offer an effective way to retrieve the information you need from your database, without having to understand all of the ins and outs of Lucene itself. Most of the work is done for you, so searching for information becomes really simple.

Essentially, these tools take index data as it's placed inside them, making it easier to retrieve or reference that information. Both Solr and Elasticsearch are popular tools with large, active communities (so there's plenty of help to be had on sites like Stack Overflow in the event something is amiss). The most common use for Solr and Elasticsearch is for enterprise search and providing search, wherever it's needed, like search on a company's website.

The Elasticsearch advantage

If all you're looking for is a search function you can integrate into your website, then either Elasticsearch or Solr will do. The difference is only evident when you try to scale or do more with the tool you've chosen.

Built-in Clustering

One of the initial advantages of Elasticsearch is the built-in clustering. Elasticsearch makes the creation, scaling, and management of the cluster a lot easier because it's all neatly rolled into the product. This has been a focus of Elasticsearch from the beginning. Elasticsearch clusters provide a lot more capabilities to manage and scale the cluster than Solr does. Solr requires external tools, like Zookeeper, to manage the cluster. Plus Solr didn't really offer the scalability of Elasticsearch until SolrCloud was launched and it just hasn't caught up.

Beyond Search

When you look at the functionality of both products, Solr's features don't really go far beyond the search function—that's its primary focus. So if you're hoping to do something with the data you pull into a search, Solr isn't the tool you want.

Elasticsearch has spent the last few years expanding the use cases. An entire ecosystem has formed around Elasticsearch that provides additional capabilities, called the Elastic Stack. The Elastic Stack includes tools for grabbing data (Beats), processing/transforming data (Logstash), visualizing data (Kibana), and more. In other words, Elasticsearch can provide functionality that's normally reserved for expensive business intelligence suites.

In the end, Elasticsearch took the costs involved in developing a tool in-house or paying for a third-party Business Intelligence (BI) tool, and threw it aside. **For anyone in need of BI and looking to save tens of thousands, it's a great option.**

Clustered Elasticsearch – Indexing, Shard, and Replica Best Practices

Some of the most common sources of support tickets we see on the ObjectRocket for Elasticsearch platform are related to indexing, shard count, and replication decisions.

Read on for best practices for indexing, sharding, and replicas for Elasticsearch.

When you're first starting out, Elasticsearch is awesome at spreading data across your cluster with the default settings. Once your cluster begins to grow, the defaults can get you in trouble. **Let's go over some of the basics of sharding and provide some best practices for indexing and shard count.**

An intro to Elasticsearch sharding

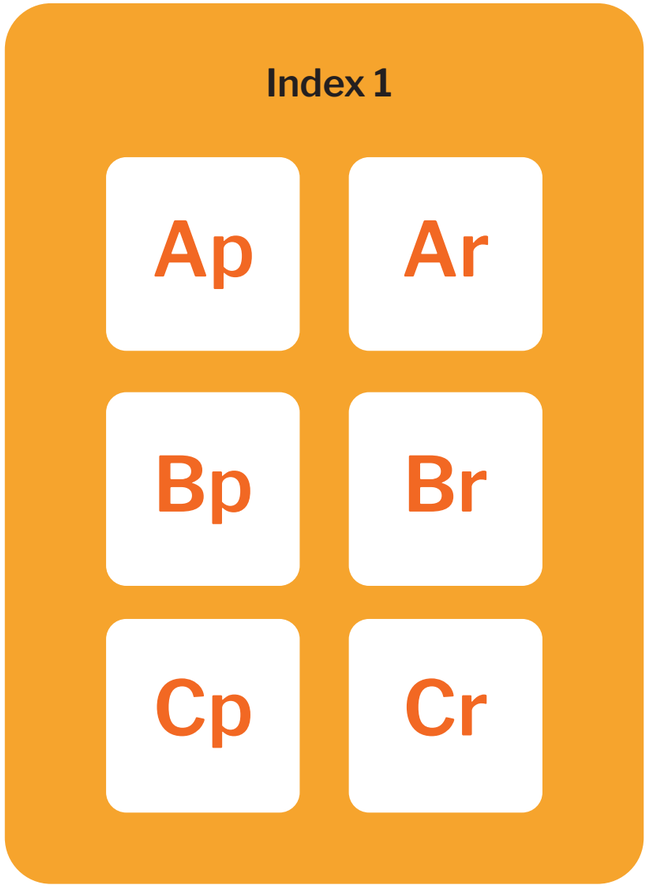
The basic concept of sharding is splitting up the your data into a number of chunks so that searches can operate on multiple parts in parallel. In order to facilitate clustering and parallelization of index functions, each index in your Elasticsearch instance is sliced up; These slices are called shards. Let's look at some of the key behaviors for shards.

The rules

Each shard is replicated based on the number_of_replicas setting for the index. E.g. For a number_of_replicas setting of one, there will two copies of each shard: one primary shard + one replica shard. The primary shard is the main shard and used for indexing/write and search/read operations, while the replicas are used only for search/read operations and for recovery if a primary fails.

- Replica shards must reside on a different host than their primary.
- By default, shards are automatically spread across the number of hosts in the cluster, but multiple primary shards can be placed on the same physical host. There are a number of Elasticsearch settings to modify this behavior (e.g. rebalancing, where shards are allocated, etc.).
- Shards can not be divided further. Each individual shard must reside on only one host.
- The number of shards can be set during index creation or you can use a global default. Once the index is created, the number of shards cannot be changed without reindexing.
- The number of replicas per index can be set either during index creation or a global default can be used. This CAN be changed after the index is created.

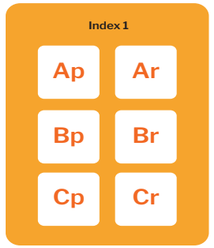
Now that we've set some ground rules, let's look at a small example.



Here's an index created with a shard count of three and a replica setting of one. As you can see in the diagram above, Elasticsearch will create 6 shards for you: Three primary shards (Ap,Bp, and Cp above), and three replica shards (Ar, Br, and Cr).

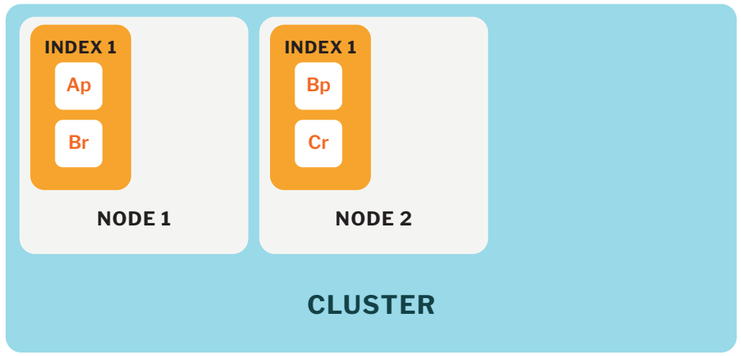
Elasticsearch will ensure that the replicas and primaries will be placed on physically different hosts, but multiple primary shards can and will be allocated to the same host. Now that we're talking hosts, let's dive into how shards are allocated to your hosts.

Shard allocation and clustered Elasticsearch

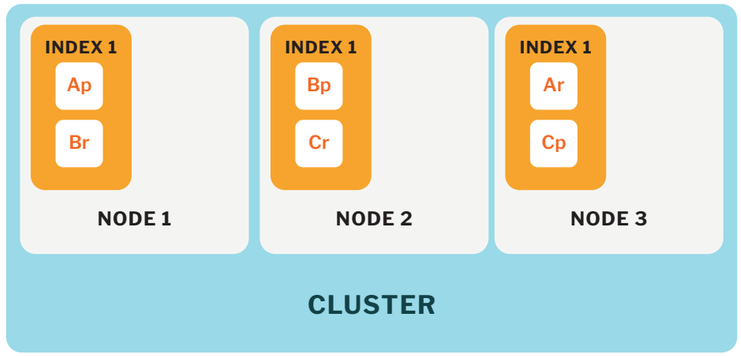


By default, Elasticsearch will attempt to allocate shards across all available hosts. At ObjectRocket, each cluster is made up of master nodes, client nodes, and data nodes. It's the data nodes in our architecture that form the "buckets" that the shards can be assigned to.

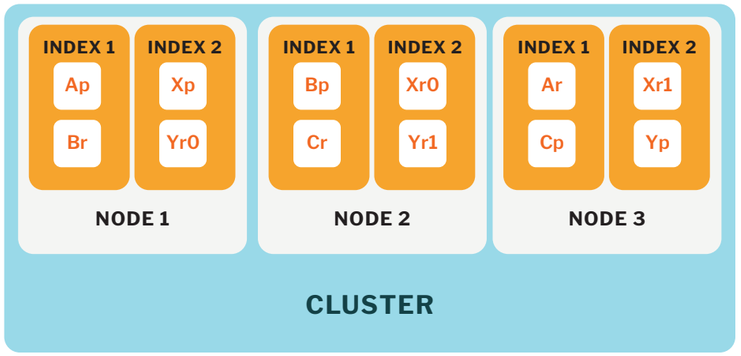
Keep in mind that the examples here show just one possible allocation, the only thing that's definite is that a replica will always be placed on a different data node than its primary.



1 Using our example above, let's take those six shards and assign them to an ObjectRocket for Elasticsearch cluster with 2 data nodes (the minimum). For each shard, the primary will land on one data node, while the replica is guaranteed to be on the other node.



2 Now, let's extend this example and add a third data node. What you see is that two shards will be moved to the new data node. You're now left with 2 shards on each node.



3 Finally, let's add a new index to this cluster with a a shard count of two and the number of replicas set to two. What you're left with is two new primaries (Xp and Yp) and four replicas (Xr0, Xr1, Yr0, Yr1), that could be spread across the cluster as seen in the picture.

That's it. Elasticsearch does all of the hard work for you, but there are some pitfalls to look out for.

PITFALL #1 Massive Indexes and Massive Shards

The most common (and easiest to mitigate) issue in Elasticsearch is a massive index with massive shards. We see this a lot. A user starts out with a very manageable single index and as their application grows, so does their index. This then leads to huge shards because shard size is directly related to the amount of data in the cluster.

The first issue this causes is poor efficiency in cluster utilization. As the shards get larger and larger, they get harder to place on a data node since you'll need a large block of free space on a data node to place a shard there. This leads to nodes with a lot of unused space. For example, if I have 8GB data nodes but each shard is 6GB, I'll be stranding 2GB on each of my data nodes.

The second issue is "hot spotting". If your data is consolidated into few shards then complex queries will not have the opportunity of being split across a larger number of nodes and executing in parallel.

Don't be stingy with indexes

The first and easiest solution is to use multiple indexes. Spreading your data across multiple indexes will increase the number of shards in the cluster and help spread the data a little more evenly. In addition to just an easier game of "Tetris" when Elasticsearch places shards, multiple indexes are easier to curate. Also, the alias capabilities in Elasticsearch can still make multiple instances look like a single index to your app.

Most of the Elastic Stack will create daily indexes by default, which is a good practice. You can then use aliases to limit the scope of searches to specific date ranges, curator to remove old indexes as they age, and modify index settings as your data grows without having to reindex the old data.

Increase shard count as your index size increases

In addition to adding indexes more frequently, you can also increase the shard count as your index sizes start to increase. Once you see shard sizes starting to get a little too large, you can update your index template (or whatever you use to create new indexes) to use more shards for each index. However, this only helps if you're regularly creating new indexes, which is why this recommendation is listed second. Otherwise, you'll have to reindex to modify shard count, which is not impossible, but a little more work than managing multiple indexes.

Our rule of thumb here is if a shard is larger than 40% of the size of a data node, that shard is probably too big. In this case, we recommend reindexing to an index with more shards, or moving up to a larger plan size (more capacity per data node).

PITFALL #2 Too many indexes/shards

The inverse of pitfall #1 is far too many indexes or shards. After reading the previous section, you may just say “Fine, I’ll just put every doc in its own index and create a million shards”. The problem is that indexes and shards have overhead. That overhead manifests itself in storage/memory resources as well as in processing performance.

Since the cluster must maintain the state of all shards and where they’re located, a massive number of shards becomes a larger bookkeeping operation which will have an impact on memory usage. Also, since queries will need to be split more ways, there will be a lot more time spent in scatter/gather for queries.

This one is a little harder to give exact guidance on, since it is highly dependent on the size of the cluster, use case, and a few other factors, but in general we can mitigate this with a few recommendations.

Shards should be no larger than 50GB
In general, 25GB is our target for shard size. 50GB is where we have the conversation with our customers about reindexing. This has as much to do with the performance of the shard itself as it does with the process of moving that shard when you need to. Whenever rebalancing, shards may need to be moved to a different node in the cluster. Moving 50GB of data can take a significant amount of time and then you’ve got that capacity tied up on two nodes during that entire process.

Keep shard size less than 40% of data node size
The second metric we look at for shard size is what percentage of the data node capacity a shard takes up. On the ObjectRocket service, we offer different plan sizes that are related to the amount of storage on the data nodes. We try to size the cluster and the shards to ensure that each of the largest shards don’t take up more than 40% of a data node’s capacity. In a cluster with a number of indexes at a mix of sizes, this is fairly effective, but in a cluster with a single or very few indexes that are very large, we are even more aggressive and try to keep this below 30%.

The idea here is to make sure that you’re not stranding capacity on a data node. If your shards are 45% the size of the data node, for example, you’ll need a data node at roughly half utilization to be able to place that shard. That’s a lot of spare capacity to leave lying around!

Conclusion

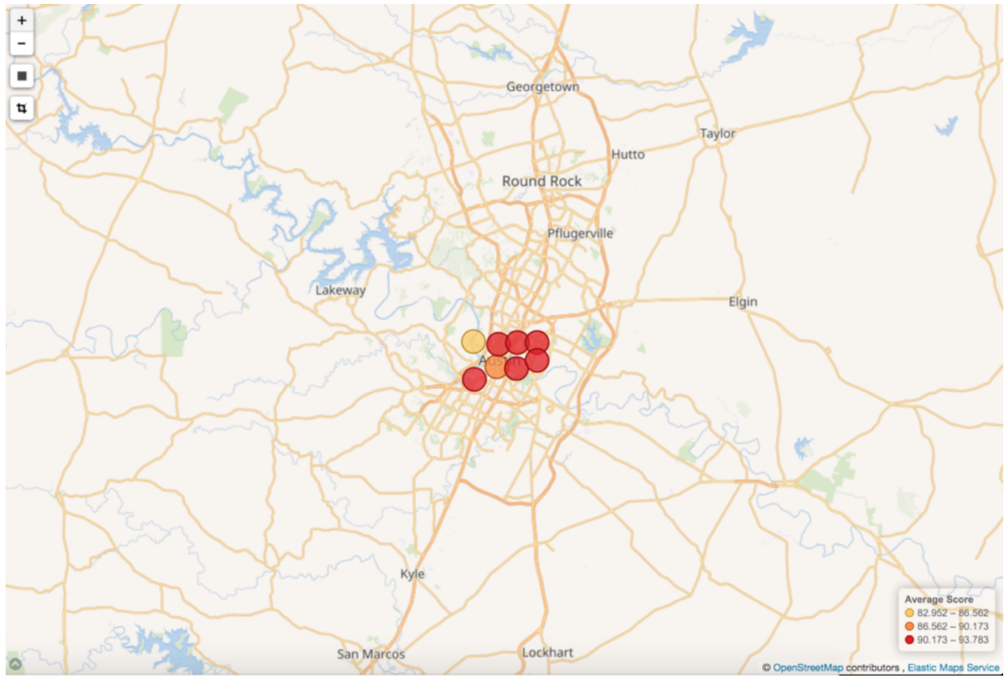
Selecting the right shard and indexing settings can be a moving target, but by planning ahead, making some good decisions up front and tuning as you go, you can keep your cluster healthy and running optimally. We help businesses refine their Elasticsearch instances all the time. Contact us for an Elasticsearch with Kibana consultation with our Elasticsearch DBAs.

Third Party Map and Tile Services with Kibana

Out of the box, Kibana includes the ability to display geo-data on maps provided by Elastic's tile service. This provides a great introduction to what Kibana can do, but the maximum zoom level is limited if you don't have access to X-Pack.

Read on to learn about your third party options.

Using Elastic's tile service, the maximum zoom for ObjectRocket's home base of Austin, Texas is:



The good news is that it's easy enough to configure Kibana to use another tile service or WMS compliant mapping server.

Why Use a Different Mapping Service?

The Elastic tile service is a great benefit to all users, but there may be times you want to diverge from the default service. Here are a few:

- You want to zoom further to street-level data
- You want to add new layers or provide a different map style
- You want a fully custom map, like the inside of a building, for example

What these all boil down to is finding a server that provides the data you want and then configuring Kibana to use that server.

Unfortunately, Kibana can cause a bit of confusion because it can leverage two completely different types of map services. I'll introduce those before we jump into how to use them. By default, Kibana draws maps from a Tile Map Service, or TMS. Tile services chop up maps into square tiles that can be accessed by their coordinates and zoom level. This is what Elastic provides with their service and this is configured globally in the kibana.yml configuration file. An alternate way to display maps in Kibana is to use a WMS-compliant Map Service, or WMS. A web map service operates using a different protocol and generates maps from data in a GIS database. This can be set from within the Kibana UI.

Using Third Party Map Services

There are a number of commercial map services out there, as well as free data sources for creating maps like OpenStreetMap. For testing purposes, there are free tile services based on OpenStreetMap, like Stamen, that you can test with. On the WMS side, there are also some free map servers for testing, like the National Map in the US. Finally, there are options for setting up your own mapping service, which I'll cover later.

Setting up Kibana to use a different tile service

Once again, by default mapping for Kibana is based off of a tile service. Stamen, which is mentioned above, is one of my favorite free to test services and offers a number of sharp map styles, so we'll test with that.

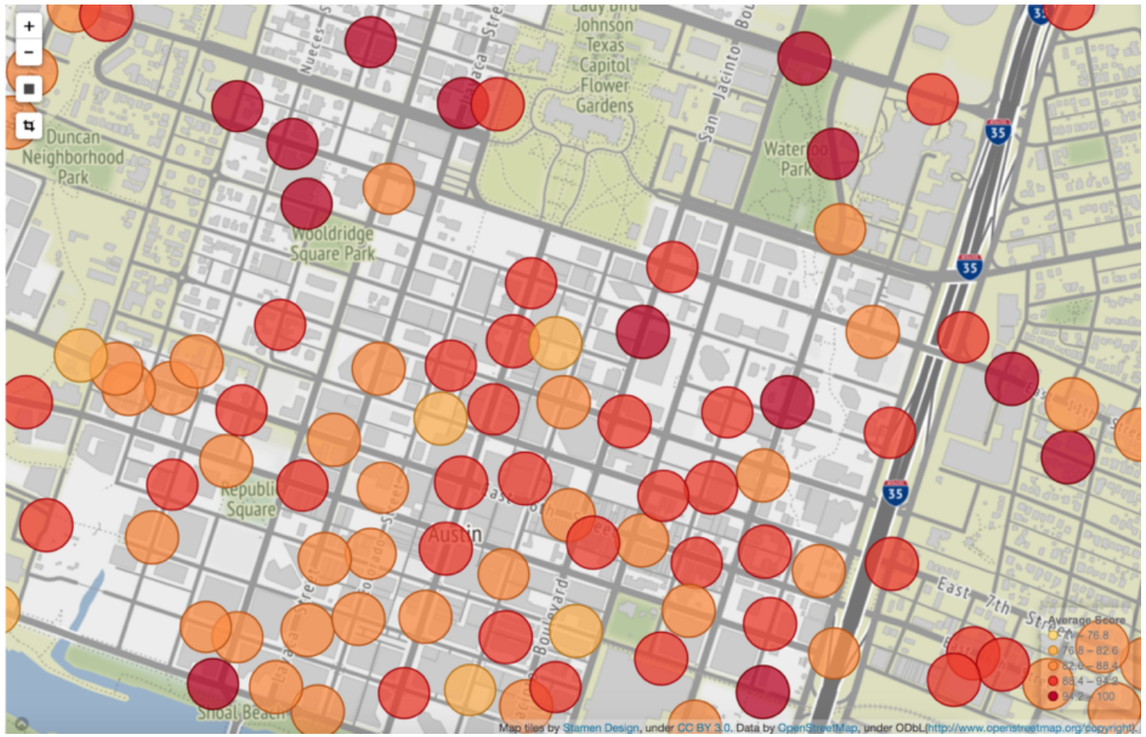
You'll need access to your kibana.yml file to make this change, so if you're on a hosted service, like ObjectRocket for Elasticsearch, you can use a local kibana install to test it out first. Once you've located the kibana.yml file, you'll add the following entries:

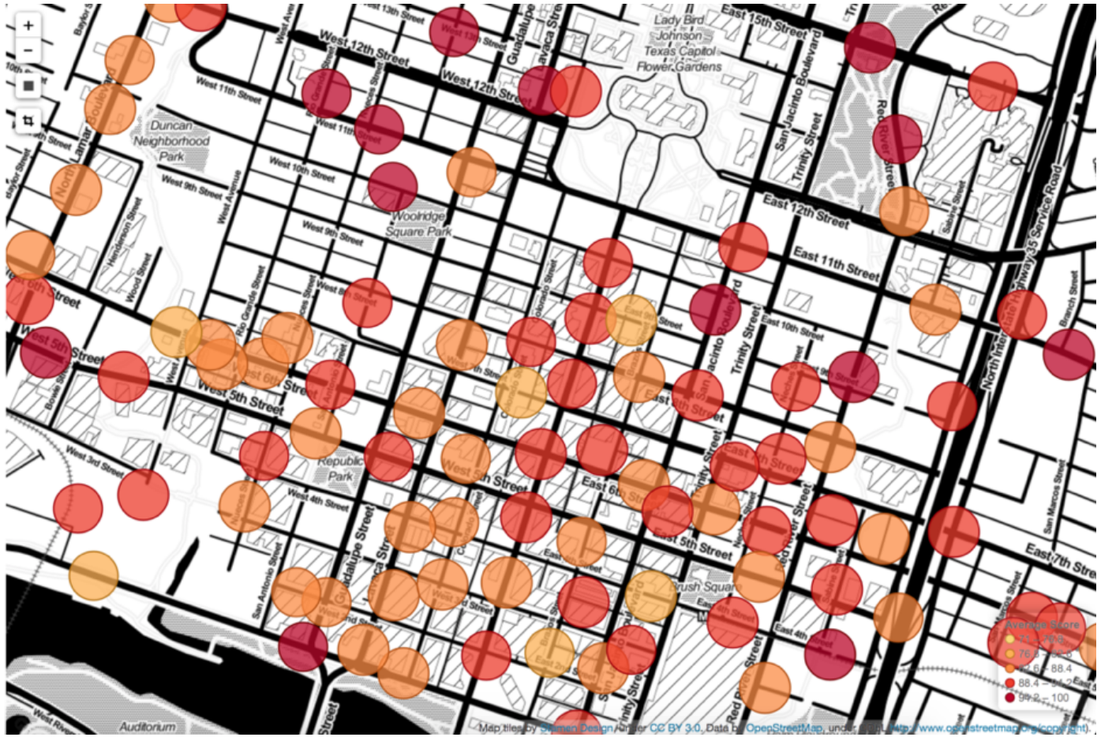
```
tilemap.url: "https://stamen-tiles.a.ssl.fastly.net/terrain/{z}/{x}/{y}.jpg"

tilemap.options.maxZoom: 20

tilemap.options.attribution: 'Map tiles by [Stamen Design](http://stamen.
com), under [CC BY 3.0](http://creativecommons.org/licenses/by/3.0). Data by
[OpenStreetMap](http://openstreetmap.org), under ODbL(http://www.openstreetmap.org/
copyright).'
```

The first setting is the url for the tile service and follows the usual endpoint/{z}/{x}/{y}.jpg/png format that standard tile services use. Note that Stamen has a number of cool styles, like 'toner' and 'watercolor', that you can use by replacing 'terrain' in the url. The second setting is the maximum number of zoom settings. Some services don't advertise this, so it will just require a little trial and error. Finally, and very importantly is properly attributing the maps to the creator. The attribution markdown will be displayed in the lower right corner of the map visualization.

Once that's set and you restart Kibana, you'll now see that you can zoom a lot further and get some pretty cool stylized maps.

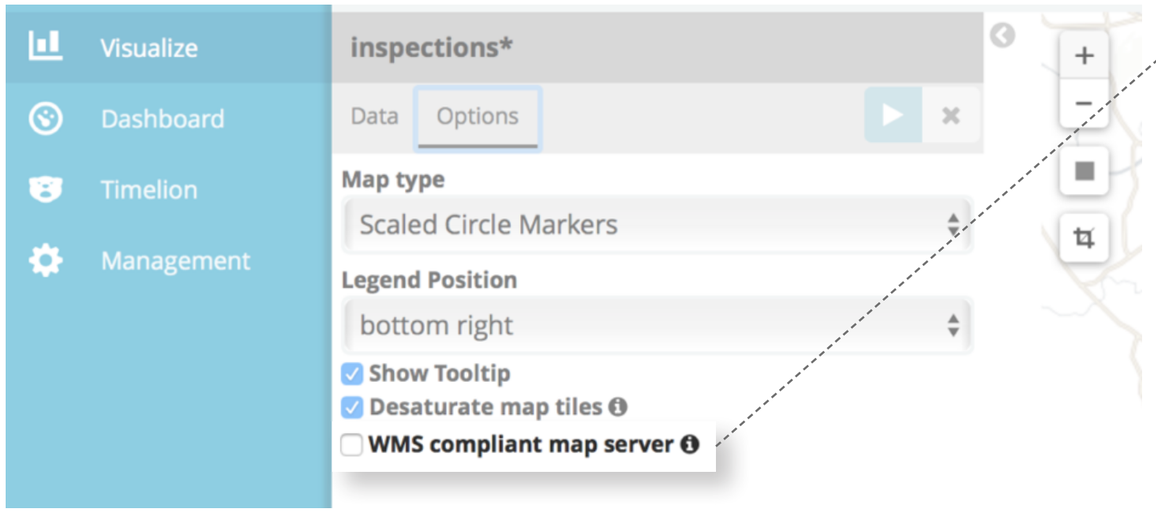
Compare the picture on right to the one on the left:





Connecting to a WMS Map from Kibana

The configuration is a little different for a WMS map and must be set within Kibana itself, rather than from the configuration file. For this example, I'll use the United States Geological Services' National Map. Specifically transportation maps so we can play with the different layers available.



1 First, you'll need to load up a coordinate map visualization in Kibana and then click the "**options**" button. From there, you're going to select "**WMS compliant map server**."

2 Once you select that, a bunch of new settings will display. You'll need to fill in the **URL for the WMS server**, which layers to use, which version of the WMS standard the server is running, image type to load, and what styles to use. These settings are described in the Kibana documentation.

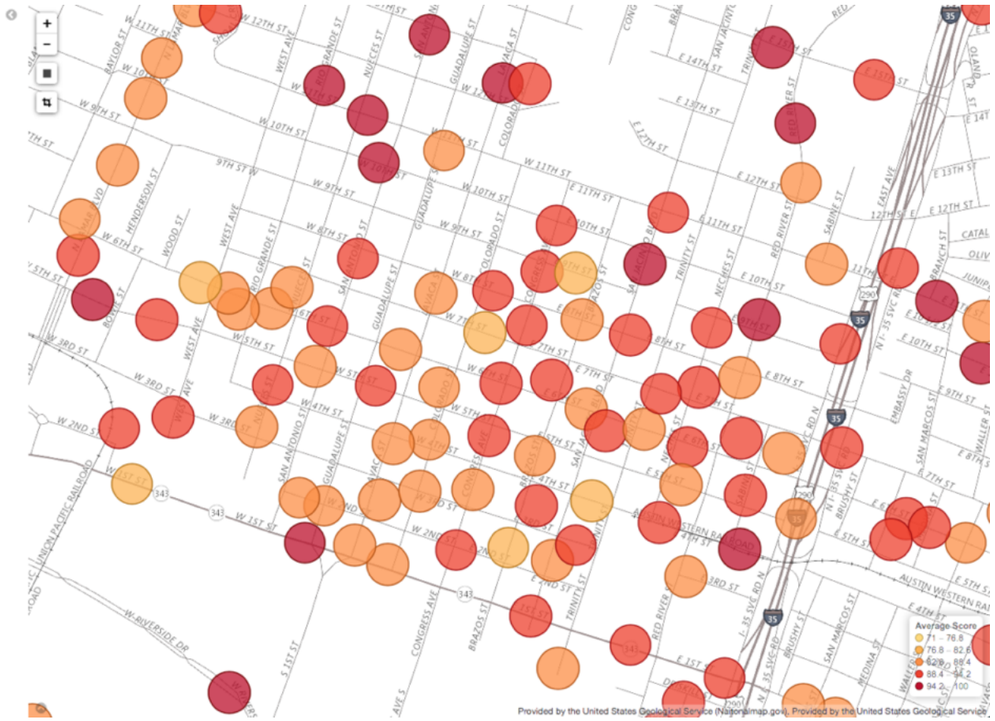
This is a bit more complicated than the tile service which just takes a few coordinates, but luckily the map service we'll use has some tools built in. First of all, if you go to the Nationalmap page, it will give you a nice description page of the map and its features. This page describes the map and all of its layers. If you want more details about the maps capabilities, click on the little WMS link at the top, and you'll see some detailed XML of the server's capabilities, like version, image formats supported and more.

3 Based on this information and the WMS link, we'll use the following settings for this map:

- **WMS url**: https://services.nationalmap.gov/arcgis/services/transportation/MapServer/WMSServer
- **WMS layers**: 0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26,27,28,29,30,31,32,33,34,35,36
- **WMS version**: 1.3.0
- **WMS format**: image/png
- **WMS attribution**: Provided by the United States Geological Service (Nationalmap.gov)
- **Styles**: blank

One note about the layers. It looks like the map we're using has a hierarchy of layers, but there's no shorthand I can find for calling all of them out so just list out the individual layers.

If you have that all set up, you should now be able to see a view like this:



The WMS settings can also be set as a default for all visualizations in the Kibana Advanced Settings screen.

Setting Up Your Own Map or Tile Server

On a final note, once you want to take this into production, running your own tile/map server may be the way to go. You can do fully custom maps, like an NHL rink, or create street maps or topographical maps from something like OpenStreetMap and apply your own layers. There are great open source tools like geoserver which is mentioned in the Elastic blog, or TileCache that all provide tutorials and detailed docs on how to get started. Also, OpenStreepMap provides tons of information in their wiki to help you get started along with the site SWITCH2OSM that provides resources to get started on serving OpenStreetMap data.

Interested in what Elasticsearch can do for your app?
We offer fully managed and hosted Elasticsearch instances complete with DBA experts that can help you get the most from Elasticsearch and Kibana without tying up your development resources. Contact us for a consultation.

Why choose ObjectRocket?

So, you want to get your payload of data into orbit without crashing your ROI?
Here are some of the top reasons why we think you should meet us on the ObjectRocket launchpad.

Open Source Innovators

We're a leader in open source database management and are well known for our deep knowledge of NoSQL databases (especially MongoDB, Elasticsearch, and Redis).

Polyglot Persistence

Using aDBaaS that offers multiple types of databases is critical.
That way, youcan use the right database for your use case, saving time and money.ObjectRocket manages several types of open source databases so thatyou can use one vendor for all your needs.

Fast and Secure

We know how to get the most out of host machines to power demanding workloads. Our platform provides high security while performing millions of operations per second.

We Grow With You

ObjectRocket was built on the core premise of enabling simple and reliable scalability for all of our databases. RocketScale™ is an ObjectRocket technology that automatically adds data nodes as you need them.

Cost Conscious

In June 2017, Crimson Consulting Group released an analysis showing that contracting with a fully managed DBaaS is a much better value than managing databases in-house. Crimson noted that ObjectRocket lowers data management costs by "orders of magnitude." That's the kind of language scientists use to describe the distance between stars.

Contact us today so we can help you travel light years ahead of your competition.

SCHEDULE A CONSULTATION

About ObjectRocket

ObjectRocket's technology and expertise helps businesses build better apps, faster so developers can concentrate on creating applications and features without having to worry about managing databases. We'll migrate your data at no cost and with little-to-no downtime. Our DBAs do all the heavy lifting for you so you can focus on your builds. We provide 24x7x365 expert support and architecture services for MongoDB, Elasticsearch, Redis, and Hadoop instances in data centers across the globe.