What is ELK stack?

What is the ELK stack?

The ELK stack is an acronym used to describe a stack that comprises of three popular open-source projects: Elasticsearch, Logstash, and Kibana. Often referred to as Elasticsearch, the ELK stack gives you the ability to aggregate logs from all your systems and applications, analyze these logs, and create visualizations for application and infrastructure monitoring, faster troubleshooting, security analytics, and more.

ELK stands for Elasticsearch, Logstash and Kibana.

E = Elasticsearch:

Elasticsearch is an open-source, RESTful, distributed search and analytics engine built on Apache Lucene. Since its release in 2010, Elasticsearch has quickly become the most popular search engine, and is commonly used for log analytics, full-text search, security intelligence, business analytics, and operational intelligence use cases.

How does Elasticsearch work?

You can send data in the form of JSON documents to Elasticsearch using the API or ingestion tools such as Logstash and Amazon Kinesis Firehose. Elasticsearch automatically stores the original document and adds a searchable reference to the document in the cluster's index. You can then search and retrieve the document using the Elasticsearch API. You can also use Kibana, an open-source visualization tool, with Elasticsearch to visualize your data and build interactive dashboards.

Is Elasticsearch free?

Yes, Elasticsearch is a free, open source software. You can run Elasticsearch on-premises, on Amazon EC2, or on Amazon Elasticsearch Service. With on-premises or Amazon EC2 deployments, you are responsible for installing Elasticsearch and other necessary software, provisioning infrastructure, and managing the cluster. Amazon Elasticsearch Service, on the other hand, is a fully managed service, so you don't have to worry about time-consuming cluster management tasks such as hardware provisioning, software patching, failure recovery, backups, and monitoring.

L = Logstash:

Logstash is a light-weight, open-source, server-side data processing pipeline that allows you to collect data from a variety of sources, transform it on the fly, and send it to your desired destination. It is most often used as a data pipeline for Elasticsearch, an open-source analytics and search engine. Because of its tight integration with Elasticsearch, powerful log processing capabilities, and over 200 pre-built open-source plugins that can help you easily index your data, Logstash is a popular choice for loading data into Elasticsearch.

Logstash benefits:

EASILY LOAD UNSTRUCTURED DATA

Logstash allows you to easily ingest unstructured data from a variety of data sources including system logs, website logs, and application server logs.

PRE-BUILT FILTERS:

Logstash offers pre-built filters, so you can readily transform common data types, index them in Elasticsearch, and start querying without having to build custom data transformation pipelines.

FLEXIBLE PLUGIN ARCHITECTURE:

With over 200 plugins already available on Github, it is likely that someone has already built the plugin you need to customize your data pipeline. But if none is available that suits your requirements, you can easily create one yourself.

K = Kibana:

Kibana is an open-source data visualization and exploration tool used for log and time-series analytics, application monitoring, and operational intelligence use cases. It offers powerful and easy-to-use features such as histograms, line graphs, pie charts, heat maps, and built-in geospatial support. Also, it provides tight integration with Elasticsearch, a popular analytics and search engine, which makes Kibana the default choice for visualizing data stored in Elasticsearch.

Is Kibana free to use?

Yes, Kibana is a free, open-source visualization tool. You can run Kibana on-premises, on Amazon EC2, or on Amazon Elasticsearch Service. With on-premises or Amazon EC2 deployments, you are responsible for provisioning the infrastructure, installing Kibana software, and managing the cluster. With Amazon Elasticsearch Service, Kibana is deployed automatically with your domain as a fully managed service, automatically taking care of all the heavy-lifting to manage the cluster.

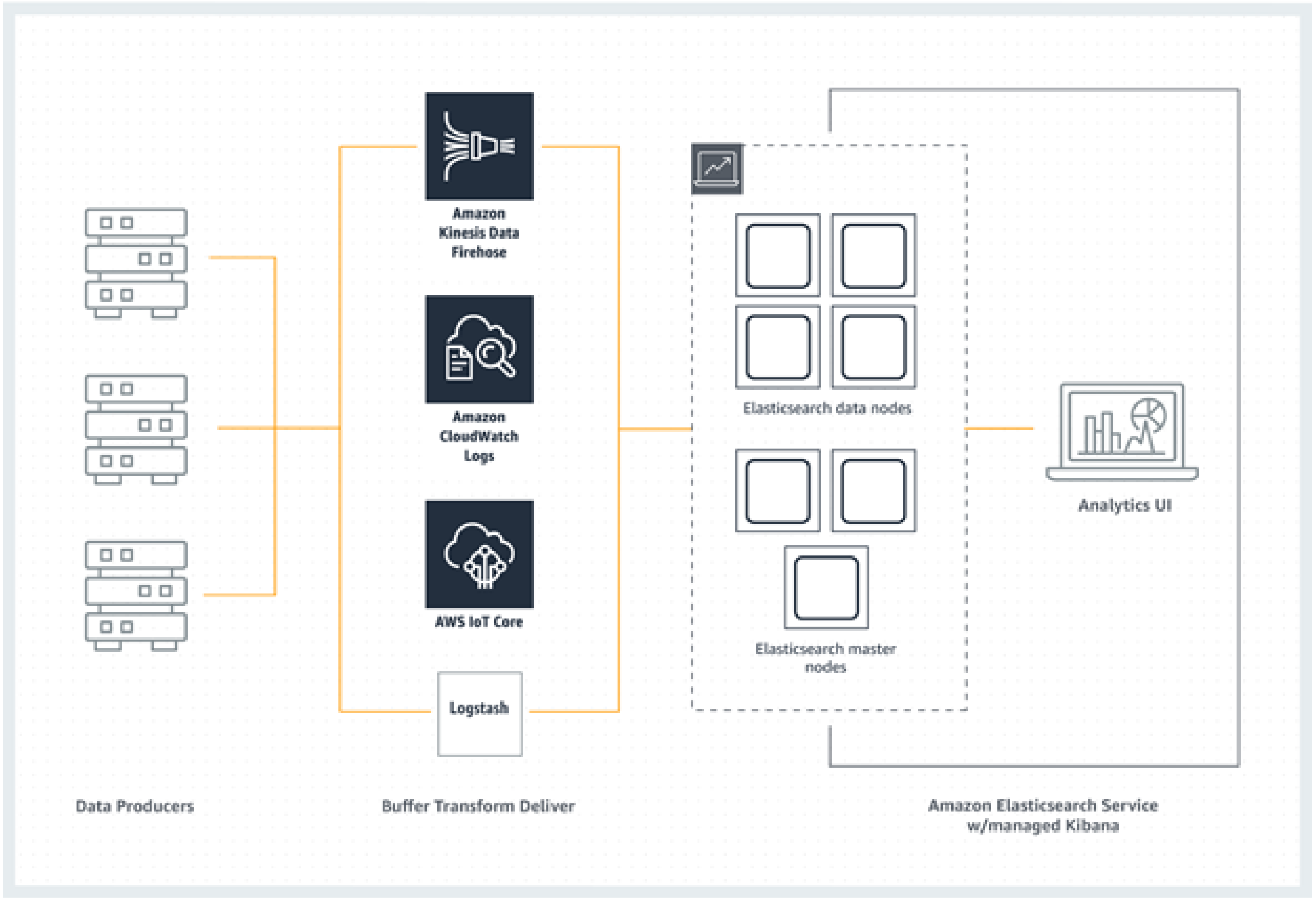
Why is the ELK stack so popular?

The ELK Stack is popular because it fulfills a need in the log analytics space. As more and more of your IT infrastructure move to public clouds, you need a log management and analytics solution to monitor this infrastructure as well as process any server logs, application logs, and clickstreams. The ELK stack provides a simple yet robust log analysis solution for your developers and DevOps engineers to gain valuable insights on failure diagnosis, application performance, and infrastructure monitoring – at a fraction of the price.

Introducing Amazon Elasticsearch Service

Amazon Elasticsearch Service is a fully managed service that makes it easy for you to deploy, secure, and operate Elasticsearch at scale. The service offers support for Elasticsearch APIs, built-in Kibana, and integration with Logstash, so you can continue to use your existing tools and code – without worrying about operational overhead.

Amazon Elasticsearch Service also integrates with other AWS services such as Amazon Kinesis Data Firehose, Amazon CloudWatch Logs, and AWS IoT giving you the flexibility to select the data ingestion tool that meets your use case requirements.



Benefits

EASY TO DEPLOY AND MANAGE:

With Amazon Elasticsearch Service you can deploy a production-ready Elasticsearch cluster in minutes. Amazon Elasticsearch Service simplifies management tasks such as hardware provisioning, software installing and patching, failure recovery, backups, and monitoring, allowing you to reduce operational overhead and build innovative applications.

INTEGRATED WITH OPEN-SOURCE TOOLS & AWS SERVICES:

Amazon Elasticsearch Service offers acces to open-source Elasticsearch APIs, managed Kibana, and integration with Logstash, so you can continue to use your existing code and data ingestion and visualization tools. The service also offers built-in integrations with other AWS services such as Amazon Kinesis Data Firehose, AWS IoT, and Amazon CloudWatch Logs for data ingestion; AWS CloudTrail for auditing; Amazon VPC, AWS KMS, Amazon Cognito, and AWS IAM for security.

EASILY SCALABLE:

Amazon Elasticsearch Service lets you scale easily and rapidly as your business requirement changes. You can scale your cluster up or down via a single API call or a few clicks. You can also configure your cluster to meet your performance requirements by selecting from a range of instance types and storage options including SSD-powered EBS volumes.

SECURE AND COMPLIANT:

Using Amazon Elasticsearch Service, you can achieve network isolation with Amazon VPC, encrypt data at-rest and in-transit using keys you create and control through AWS KMS, and manage authentication and access control with Amazon Cognito and AWS IAM policies. Amazon Elasticsearch Service is also HIPAA eligible, and compliant with PCI DSS and ISO standards to help you meet industry-specific or regulatory requirements.

HIGHLY AVAILABLE:

Amazon Elasticsearch Service is designed to be highly available using multi-AZ deployments, which replicates data between multiple Availability Zones in the same region. The service also monitors the health of your clusters and automatically replaces failed instances.

COST-EFFECTIVE:

With Amazon Elasticsearch Service, you pay only for what you use. There is no upfront fee or usage requirement. With built-in encryption and VPC support, 24x7 monitoring, and AWS support, you don't need a team of Elasticsearch experts to scale, secure, and monitor your infrastructure, resulting in lower total cost of operations.

Use cases:

LOG ANALYTICS:

Analyze unstructured and semi-structured logs generated by websites, mobile devices, servers, and sensors etc., for operational intelligence, application monitoring, root cause analysis and more. Capture, pre-process, and load log data into Amazon Elasticsearch Service using Amazon Kinesis Firehose, Logstash, or Amazon CloudWatch Logs, and subsequently, search, explore, and visualize the data using Kibana and the Elasticsearch query DSL to gain valuable insights about your users and applications.

Adobe uses Amazon Elasticsearch Service to cost-effectively analyze and visualize large amount of log data for its Developer Platform, which at peak receives over 200K API calls per second. With Amazon Elasticsearch Service, Adobe is able to easily see traffic patterns and error rates and quickly identify and troubleshoot any potential issues - all with reduced operational overhead.

REAL-TIME APPLICATION MONITORING:

Capture activity logs across your customer-facing applications and websites for real-time application monitoring and issue resolution. Push these logs to your Amazon Elasticsearch Service domain using Logstash. Elasticsearch indexes the data, makes it available for analysis in real-time, and allows you to visualize the data using the built-in Kibana plugin.

Expedia uses Amazon Elasticsearch Service for application monitoring and root-cause analysis and price optimization. Amazon Elasticsearch enables Expedia to monitor huge volumes of Docker logs cost-effectively, identify and troubleshoot issues in real-time, scale easily to accommodate additional log sources, and offload the operational overhead.

SECURITY ANALYTICS:

Enables security practitioners to centralize and analyze events from across the entire organization to enhance incident response and monitor threats across all their applications and systems in real time. Amazon Elasticsearch Service allows you to index the data as soon as it is ingested allowing you to analyze and visualize data from multiple sources instantly and find and prevent threats faster.

FULL TEXT SEARCH:

Provide a low-latency, high-throughput, personalized search experience for your users across e-commerce applications, website, data lake catalogs, and other curated application data. Amazon Elasticsearch Service provides direct access to all of Elasticsearch's rich search APIs, supporting natural language search across free text, Boolean combinations of text and metadata search, auto-completion, faceted search, location-aware search, and much more.

Mirrorweb uses Amazon Elasticsearch Service to make the UK Government and UK Parliament's web archives searchable. With Amazon Elasticsearch Service, Mirrorweb indexed 1.4 billion documents for just $337 and indexed 146 MM docs per hour – 14x faster than the previously used technology.

CLICK-STREAM ANALYTICS:

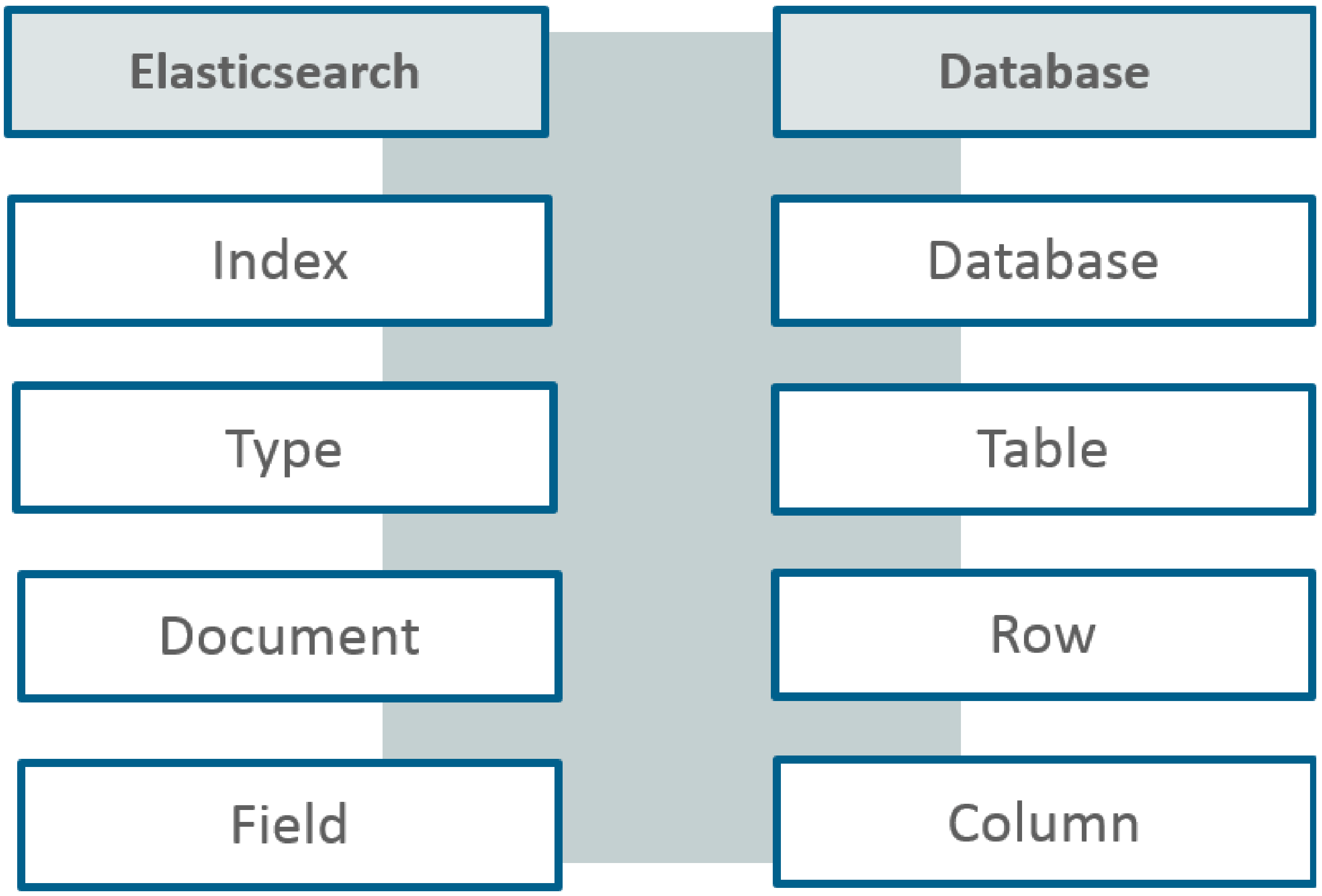
Deliver real-time metrics on digital content to enable authors and marketers to connect with their customers effectively. Load billions of small messages that are compressed and batched to Amazon Elasticsearch Service using Amazon Kinesis Firehose. With Amazon Elasticsearch Service, you can then aggregate, filter, and process the data, and refresh content performance dashboards in near real-time.

Hearst Corporation built a clickstream analytics platform using Amazon Elasticsearch Service, Amazon Kinesis Streams, and Amazon Kinesis Firehose to transmit and process 30 terabytes of data a day from 300+ Hearst websites worldwide. With this platform, Hearst is able to make the entire data stream—from website clicks to aggregated data—available to editors in minutes.

Elasticsearch – ELK Stack Tutorial

As mentioned before, Elasticsearch is a highly scalable search engine which runs on top of Java-based Lucene engine. It is basically a NoSQL database; which means it stores data in an unstructured format and SQL queries can't be performed for any kind of transaction. In other words, it stores the data inside the documents instead of tables and schemas. To have a better picture, check the following table which shows what is what, in Elasticsearch when compared to a database.

Let's now get familiar with the basic concepts of Elasticsearch.



When you work with Elasticsearch there are three major steps which you need to follow:

1. Indexing
2. Mapping
3. Searching

Let’s talk about them in detail, one by one.

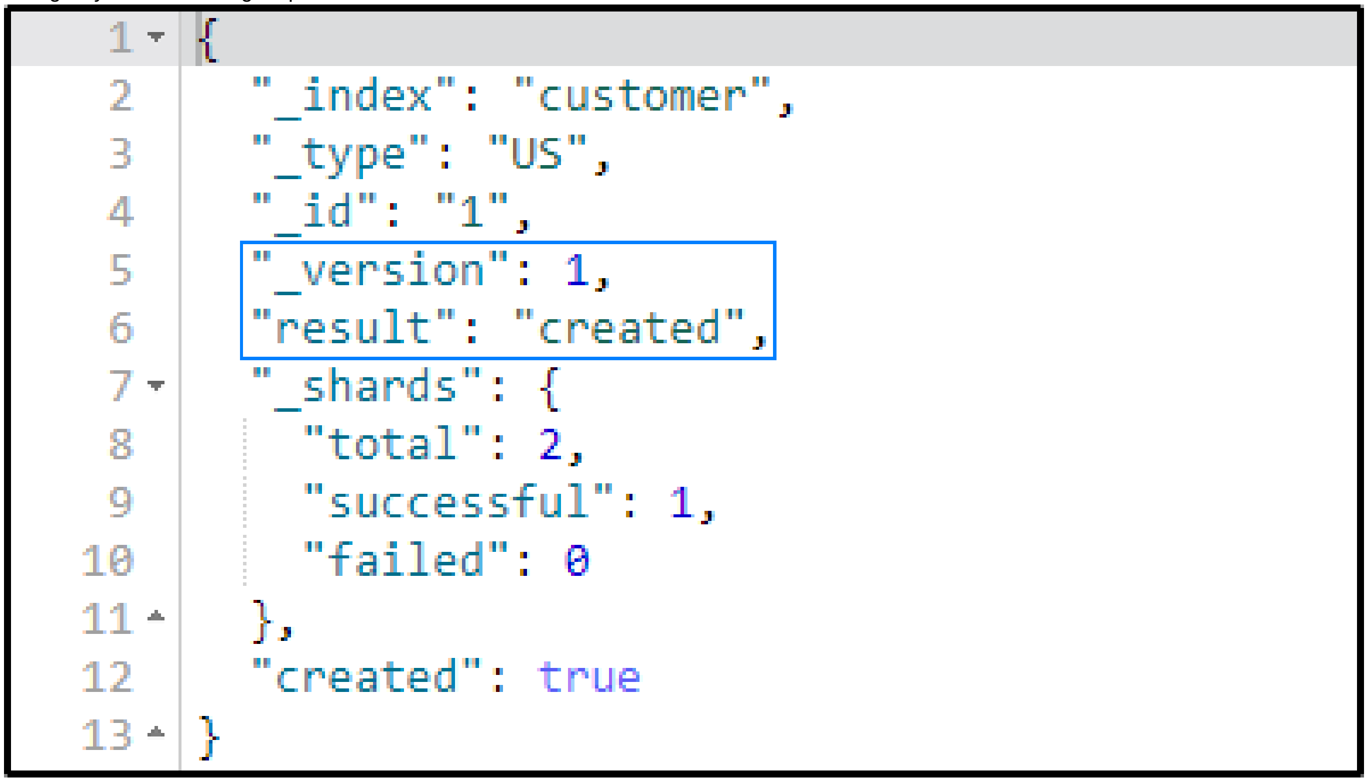
Indexing

Indexing is the process of adding the data Elasticsearch. It is called ‘indexing’ because when the data is entered into Elasticsearch, it gets placed into Apache Lucene indexes. Elasticsearch then uses these Lucene indexes to store and retrieve the data. Indexing is similar to the create and update process of CRUD operations.

An index scheme consists of **name/type/id,** where name and type are mandatory fields. In case you do not provide any ID, Elasticsearch will provide an id on its own. This entire query is then is appended to an HTTP PUT request and the final URL looks like:`PUT name/type/id` Along with the HTTP payload a JSON document, which contains the fields and values, is sent as well.
Following is an example of creating a document of a US-based customer with his details in the fields.

```
PUT /customer/US/1
{
 "ID" : 101,
 "FName" : "James",
 "LName" : "Butt",
 "Email" : "jbutt@gmail.com",
 "City" : "New Orleans",
 "Type" : "VIP"
}
```

It will give you the following output:

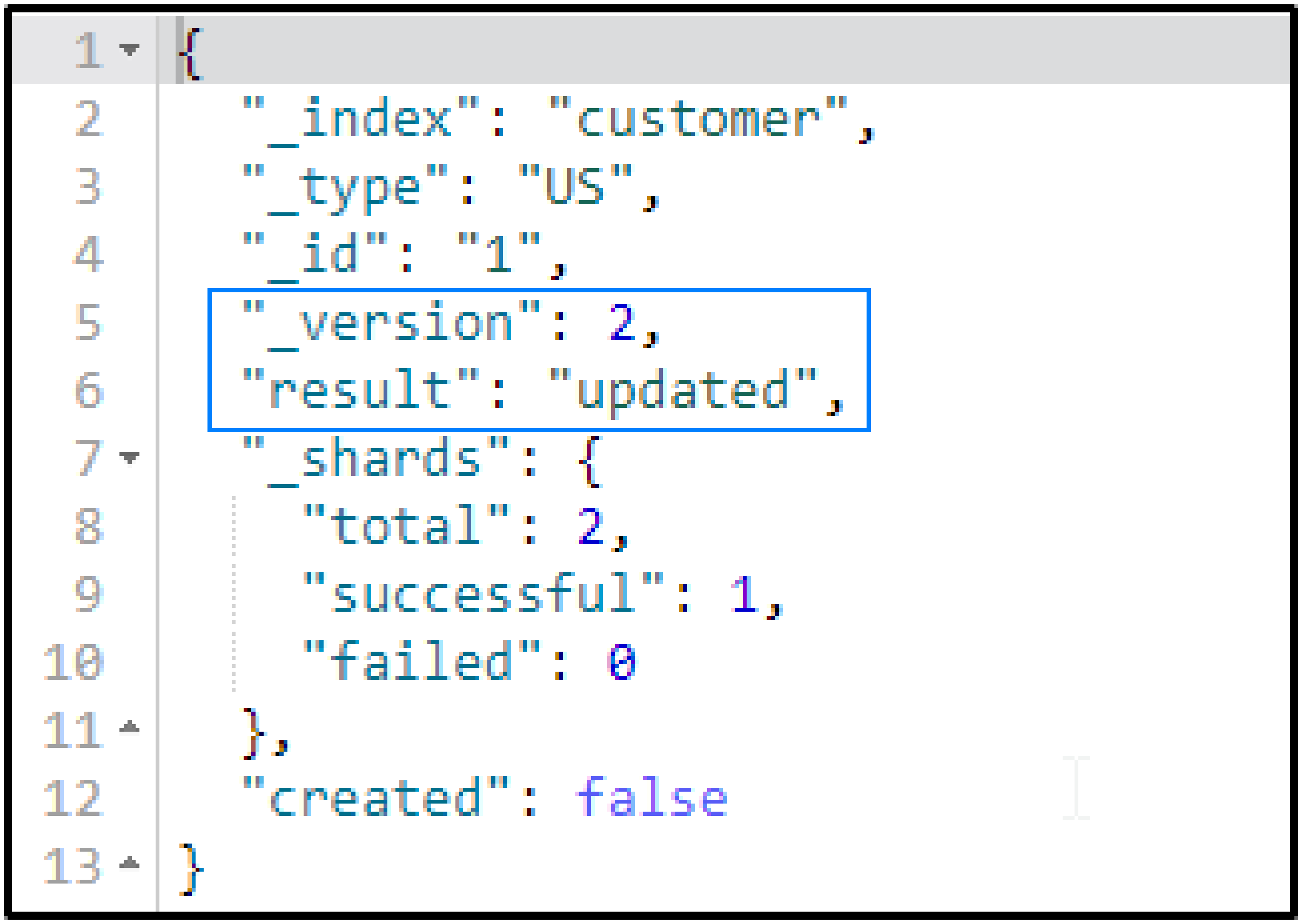


```
{
  "_index": "customer",
  "_type": "US",
  "_id": "1",
  "_version": 1,
  "result": "created",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "created": true
}
```

Here it shows the document has been created and added to the index.

Now if you try to change the field details without changing the id, Elasticsearch will overwrite your existing document with the current details.

```
PUT /customer/US/1
{
 "ID" : 101,
 "FName" : "James",
 "LName" : "Butt",
 "Email" : "jbutt@yahoo.com",
 "City" : "Los Angeles",
 "Type" : "VVIP"
}
```



```
{
  "_index": "customer",
  "_type": "US",
  "_id": "1",
  "_version": 2,
  "result": "updated",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  },
  "created": false
}
```

Here it shows the document has been updated with new details the index.

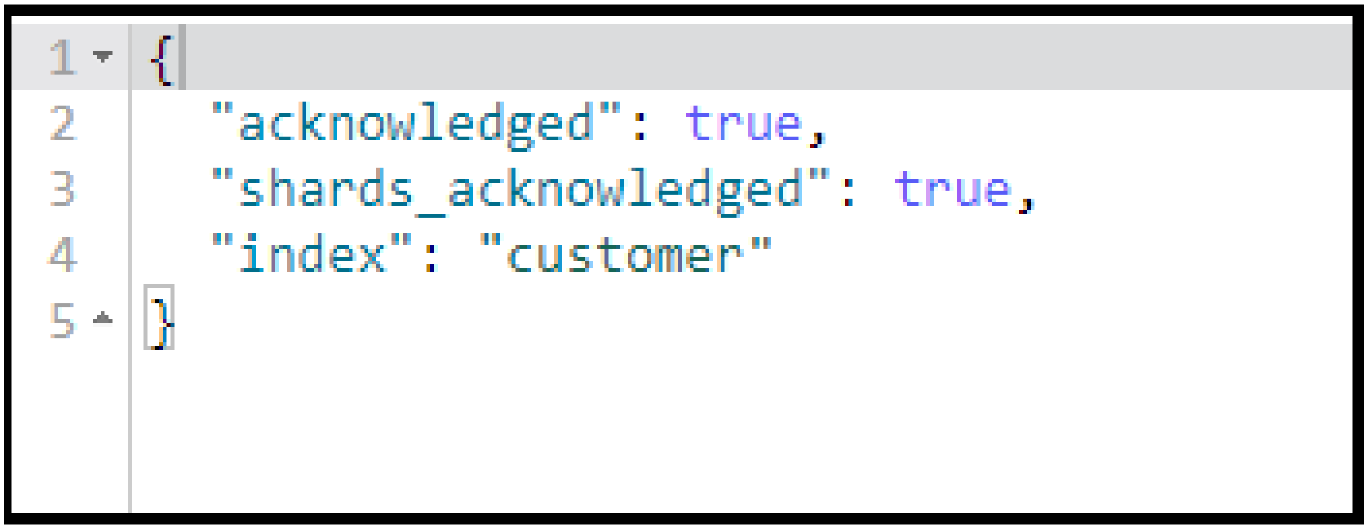
Mapping

Mapping is the process of setting the schema of the index. By mapping, you tell Elasticsearch about the data types of the attributes present in your schema. If the mapping is not done for a specific at the pre-index time, dynamically a generic type will be added to that field by Elasticsearch. But these generic types are very basic and most of the times do not satisfy the query expectations.

Lets now try to map our query.

```
PUT /customer/
{
 "mappings":{
 "US":{
 "properties":{
 "ID":{
 "type": "long"
 },
 "FName" : {
 "type" : "text"
 },
 "LName" : {
 "type" : "text"
 },
 "Email" : {
 "type" : "text"
 },
 "City" : {
 "type" : "text"
 },
 "Type" : {
 "type" : "text"
 }
 }

 }
 }
}
```



```
{
  "acknowledged": true,
  "shards_acknowledged": true,
  "index": "customer"
}
```

When you execute your query you will get this type of output.

Searching

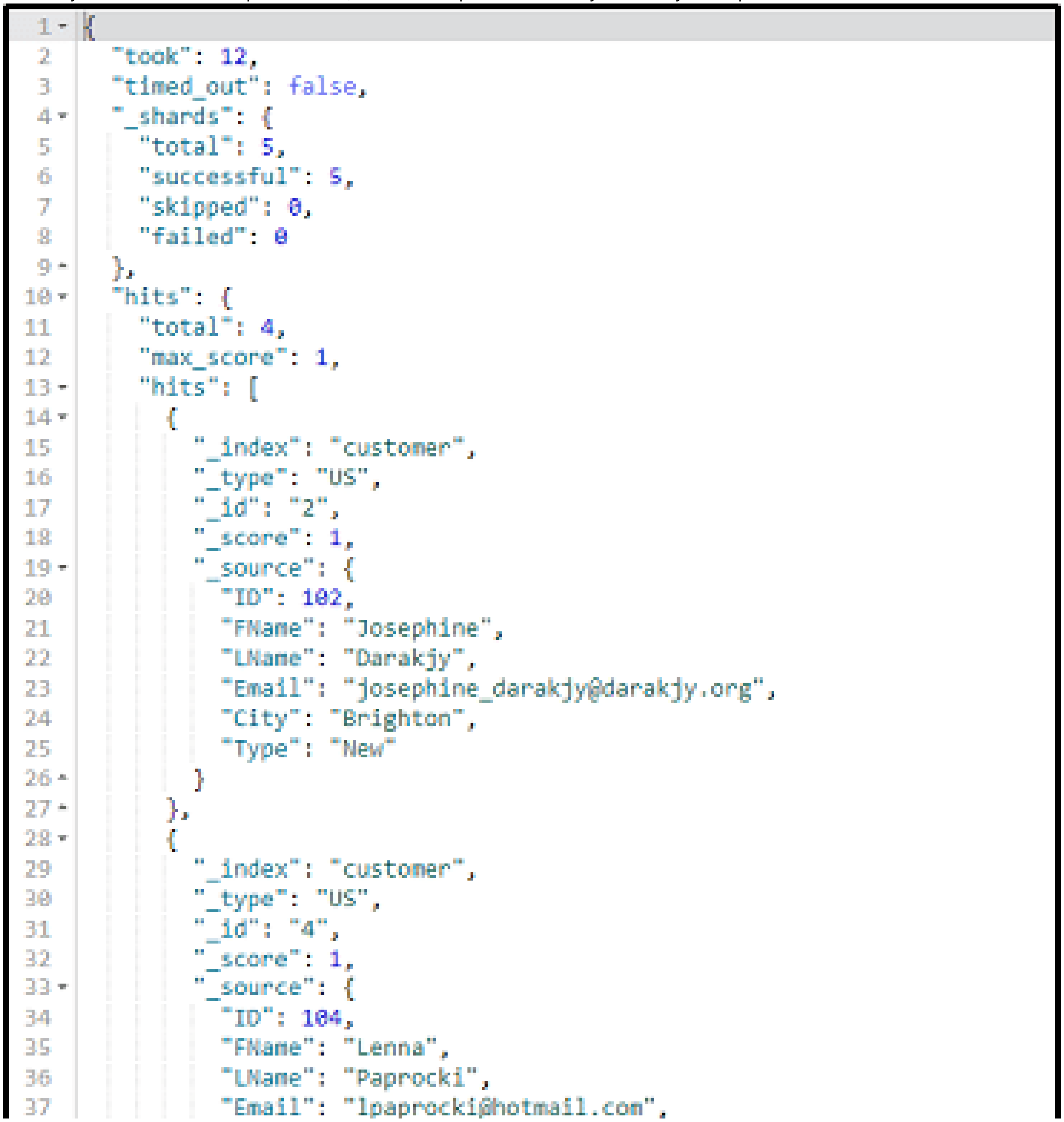
A general search query with specific index and type will look like: `POST index/type/_search`

Lets now try to search for the details of all the customers present in our ‘customer’ index.

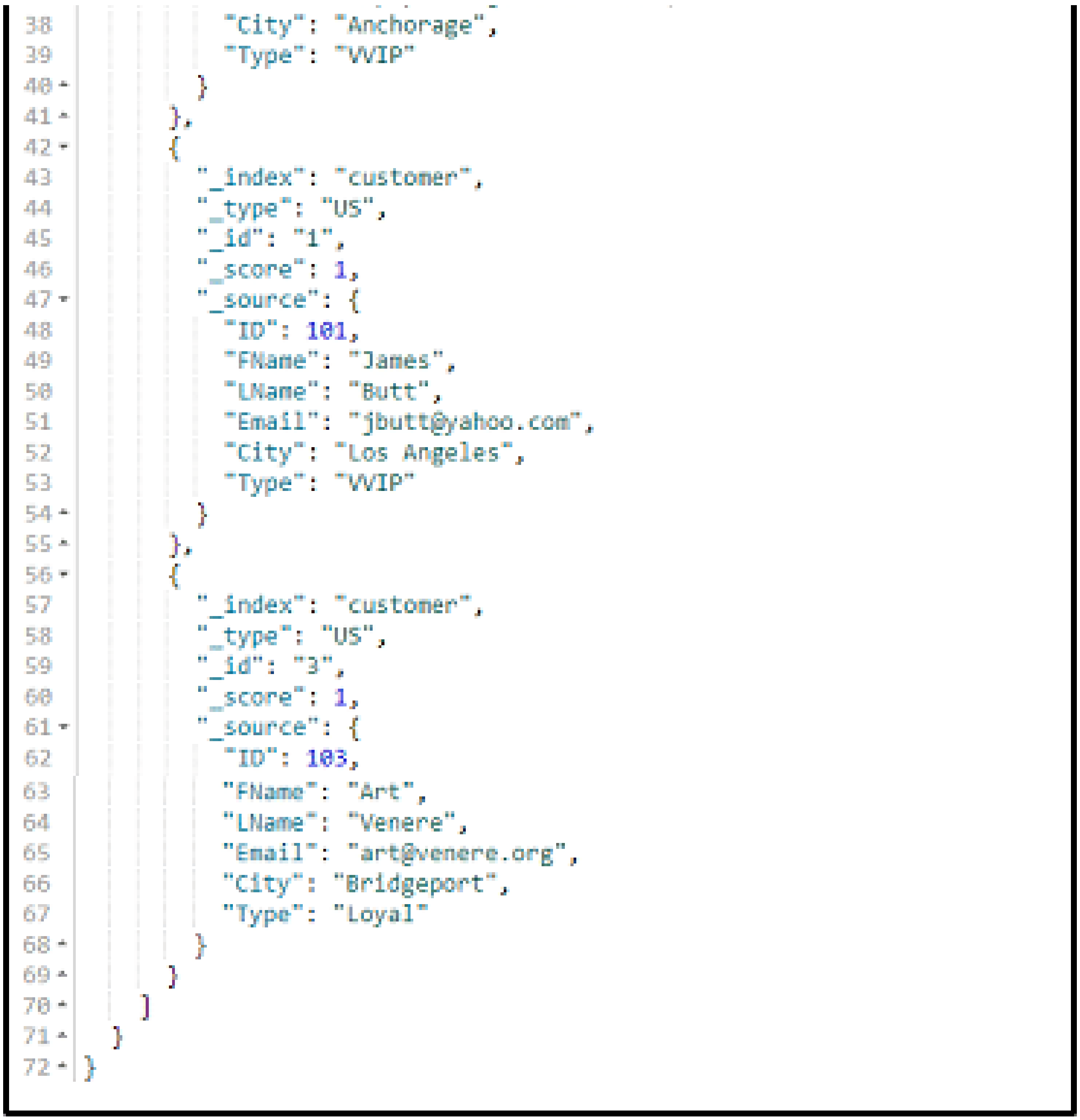
```
POST /customer/US/_search
```

When you execute this query, following result will be generated:

But when you want to search for specific results, Elasticsearch provides three ways in which you can perform it:



```
{
  "took": 12,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 4,
    "max_score": 1,
    "hits": [
      {
        "_index": "customer",
        "_type": "US",
        "_id": "2",
        "_score": 1,
        "_source": {
          "ID": 102,
          "FName": "Josephine",
          "LName": "Darakjy",
          "Email": "josephine_darakjy@darakjy.org",
          "City": "Brighton",
          "Type": "New"
        }
      },
      {
        "_index": "customer",
        "_type": "US",
        "_id": "4",
        "_score": 1,
        "_source": {
          "ID": 104,
          "FName": "Lenna",
          "LName": "Paprocki",
          "Email": "lpaprocki@hotmail.com",
```

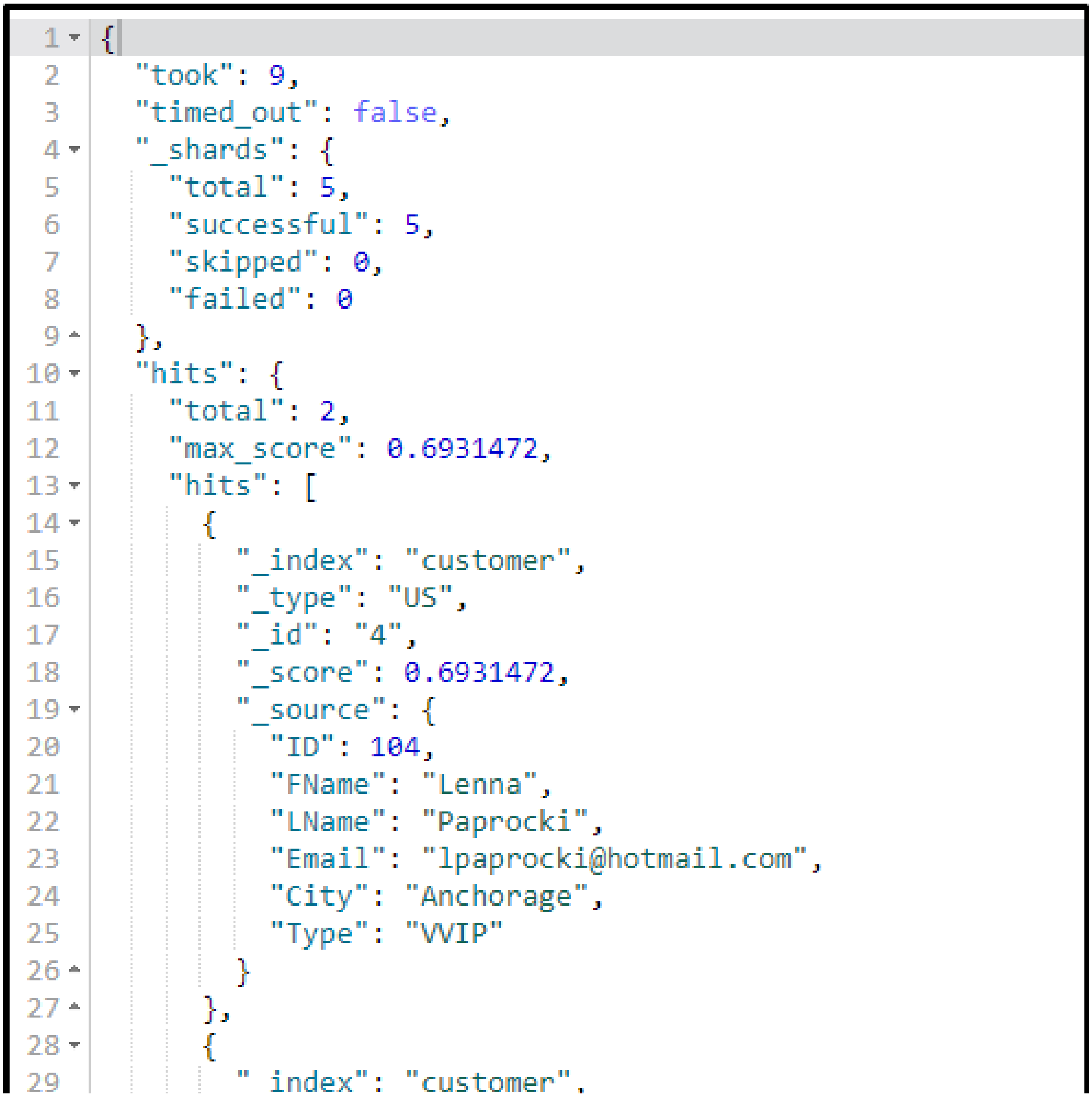


```
          "City": "Anchorage",
          "Type": "VVIP"
        }
      },
      {
        "_index": "customer",
        "_type": "US",
        "_id": "1",
        "_score": 1,
        "_source": {
          "ID": 101,
          "FName": "James",
          "LName": "Butt",
          "Email": "jbutt@yahoo.com",
          "City": "Los Angeles",
          "Type": "VVIP"
        }
      },
      {
        "_index": "customer",
        "_type": "US",
        "_id": "3",
        "_score": 1,
        "_source": {
          "ID": 103,
          "FName": "Art",
          "LName": "Venere",
          "Email": "art@venere.org",
          "City": "Bridgeport",
          "Type": "Loyal"
        }
      }
    ]
  }
}
```

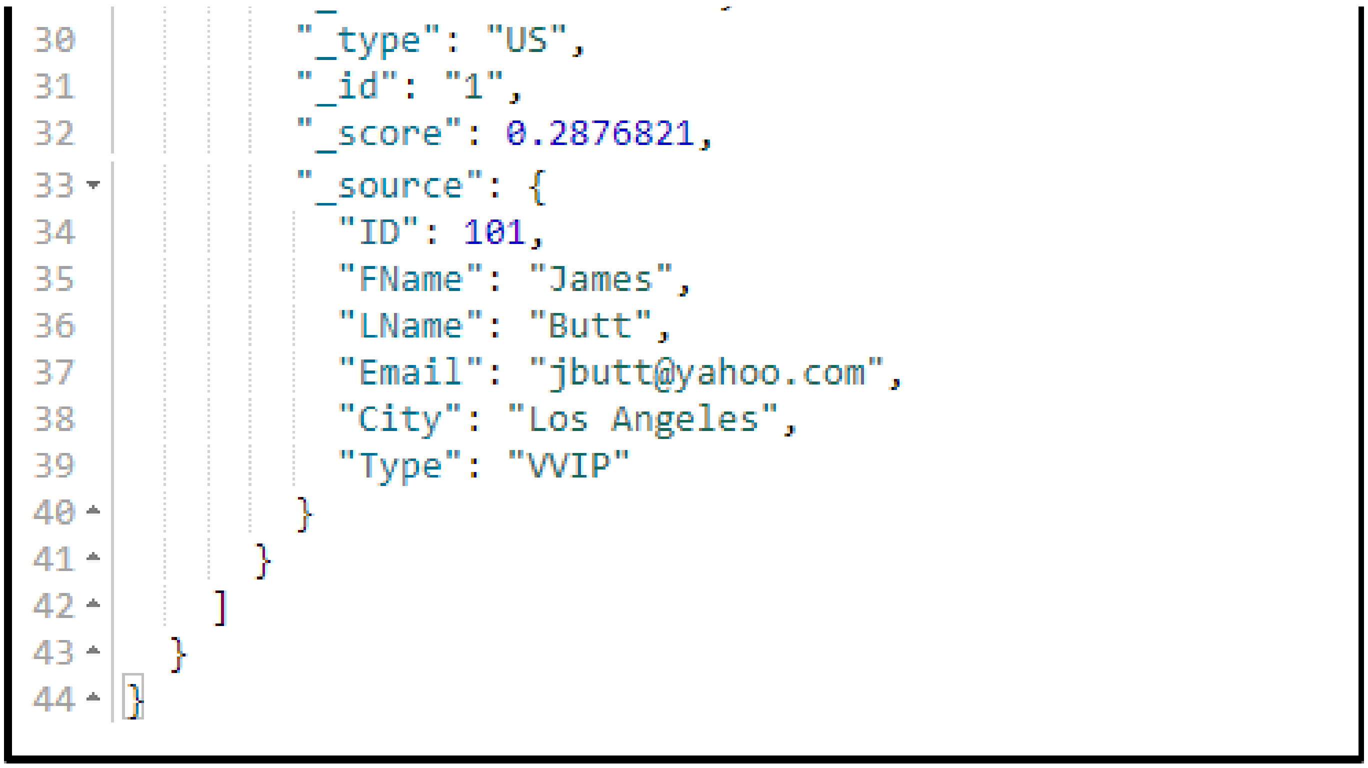
- **Using queries**

  Using queries you can search for some specific document or entries. For example, let us perform a search query for the customers who fall under ‘VVIP’ category.

```
POST /customer/US/_search
{
 "query": {
 "match": {
 "Type": "VVIP"
 }
 }
}
```



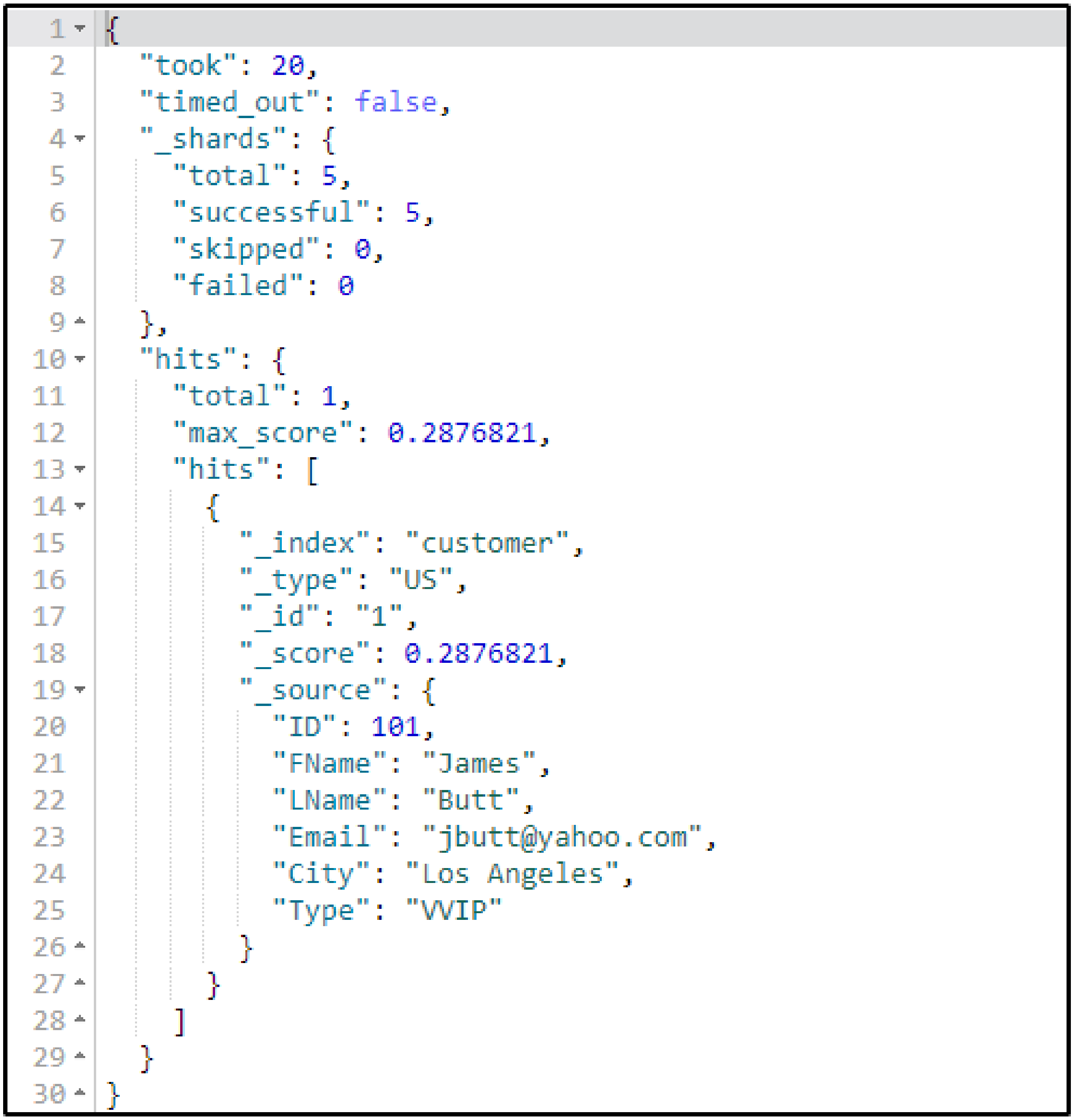
```
{
  "took": 9,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 2,
    "max_score": 0.6931472,
    "hits": [
      {
        "_index": "customer",
        "_type": "US",
        "_id": "4",
        "_score": 0.6931472,
        "_source": {
          "ID": 104,
          "FName": "Lenna",
          "LName": "Paprocki",
          "Email": "lpaprocki@hotmail.com",
          "City": "Anchorage",
          "Type": "VVIP"
        }
      },
      {
        "_index": "customer",
```


```
            "_type": "US",
            "_id": "1",
            "_score": 0.2876821,
            "_source": {
              "ID": 101,
              "FName": "James",
              "LName": "Butt",
              "Email": "jbutt@yahoo.com",
              "City": "Los Angeles",
              "Type": "VVIP"
            }
          }
        ]
      }
    }
```

- **Using filters**

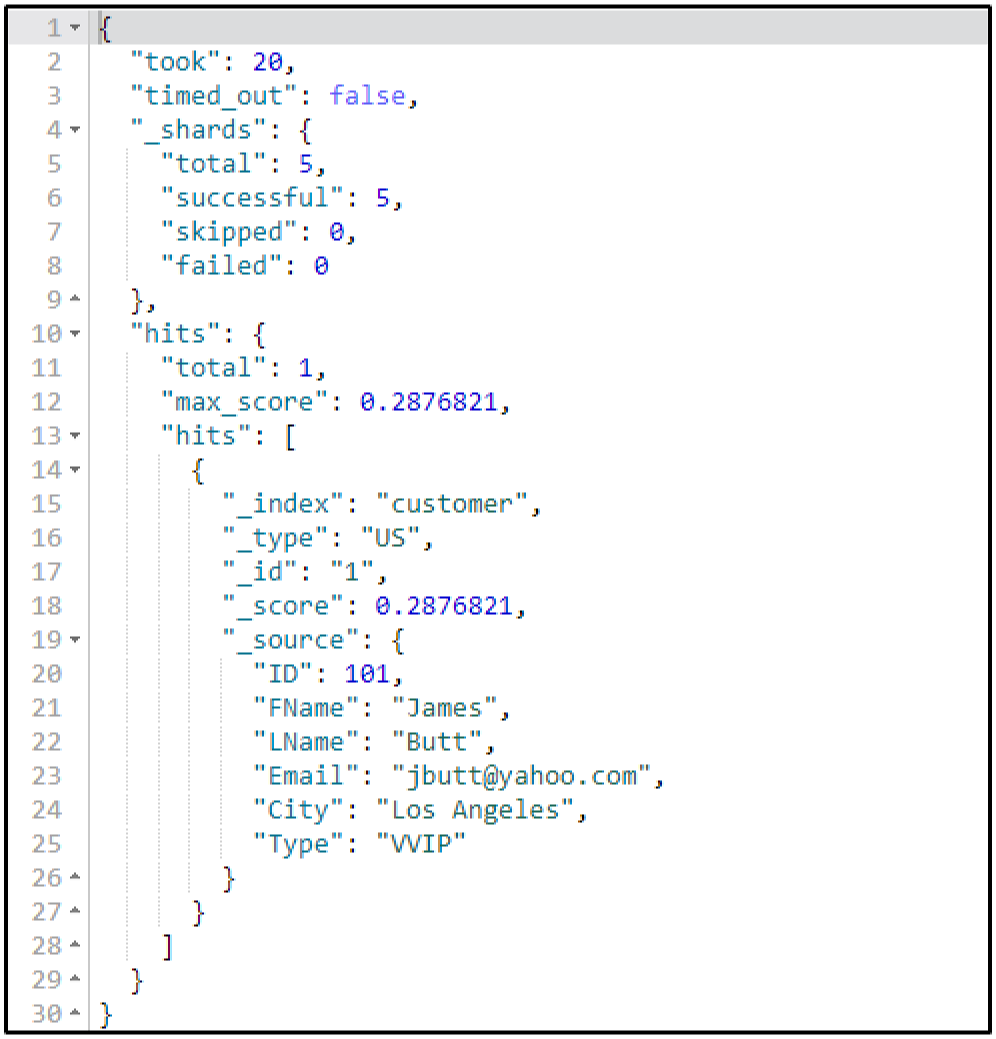
  Using filters, you can further narrow down your searches. Following is an example which searches for a VVIP customer with ID as ‘101’:

```
POST /customer/_search
{
 "query": {
 "match": {
 "Type": "VVIP"
 }
 },
 "post_filter": {
 "match" : {
 "ID" : 101
 }
 }
}
```



```
{
  "took": 20,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 1,
    "max_score": 0.2876821,
    "hits": [
      {
        "_index": "customer",
        "_type": "US",
        "_id": "1",
        "_score": 0.2876821,
        "_source": {
          "ID": 101,
          "FName": "James",
          "LName": "Butt",
          "Email": "jbutt@yahoo.com",
          "City": "Los Angeles",
          "Type": "VVIP"
        }
      }
    ]
  }
}
```

If you execute this query you'll get the following kind of result:

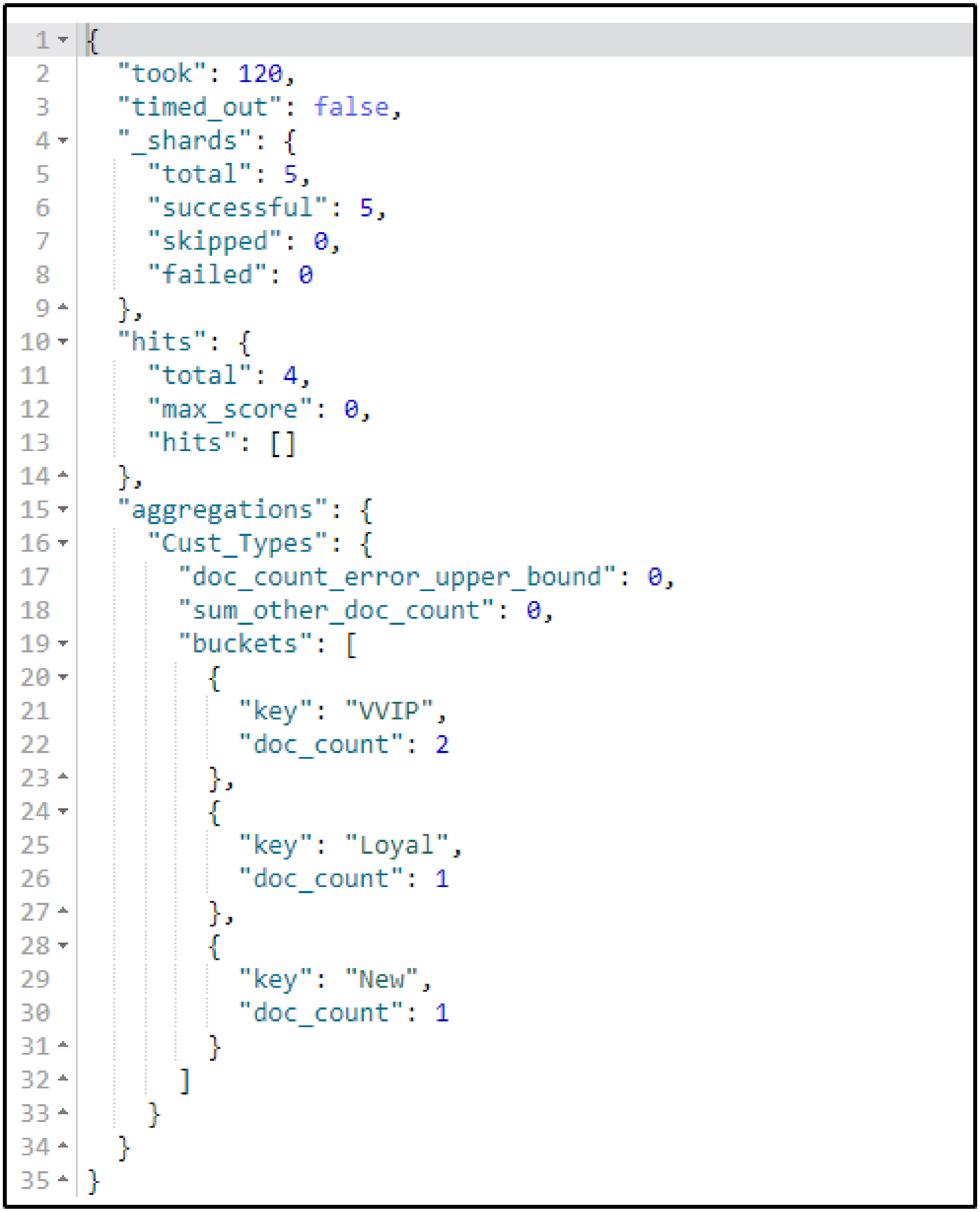


```
{
  "took": 20,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 1,
    "max_score": 0.2876821,
    "hits": [
      {
        "_index": "customer",
        "_type": "US",
        "_id": "1",
        "_score": 0.2876821,
        "_source": {
          "ID": 101,
          "FName": "James",
          "LName": "Butt",
          "Email": "jbutt@yahoo.com",
          "City": "Los Angeles",
          "Type": "VVIP"
        }
      }
    ]
  }
}
```

- **Using aggregations**

  Aggregation is a framework which helps in aggregating data through a search query. Small aggregations can be joined together in order to build up complex summaries of the data provided. Let’s perform a simple aggregation to check how many types of customers do we have in our index:

```
POST /customer/_search
{
 "size": 0,
 "aggs" : {
 "Cust_Types" : {
 "terms" : { "field" : "Type.keyword" }
 }
 }
}
```



```
{
  "took": 120,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 4,
    "max_score": 0,
    "hits": []
  },
  "aggregations": {
    "Cust_Types": {
      "doc_count_error_upper_bound": 0,
      "sum_other_doc_count": 0,
      "buckets": [
        {
          "key": "VVIP",
          "doc_count": 2
        },
        {
          "key": "Loyal",
          "doc_count": 1
        },
        {
          "key": "New",
          "doc_count": 1
        }
      ]
    }
  }
}
```

Lets now see how to retrieve a data set from an index.

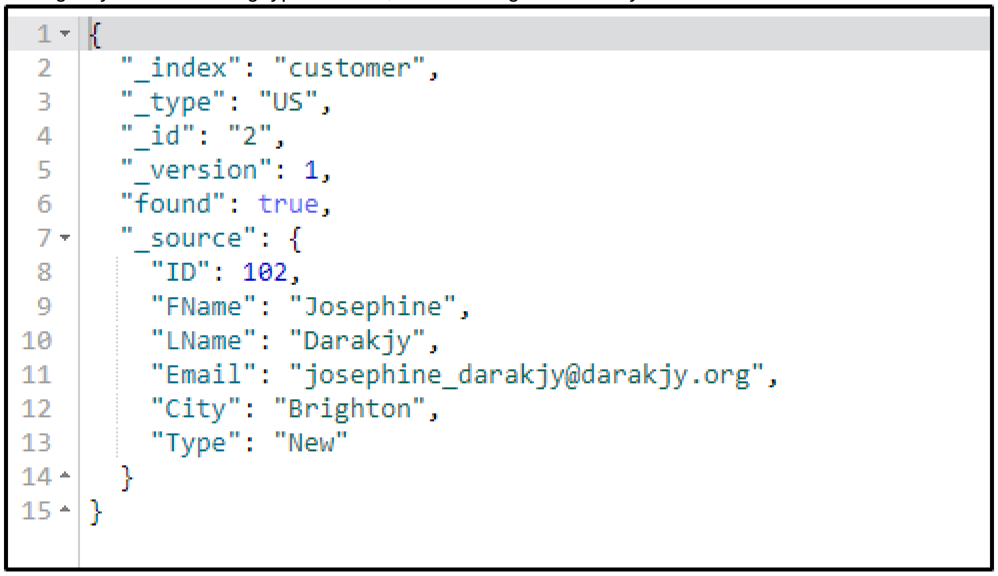
Getting Data

To check the list of documents you have within an index, you just need to send an HTTP GET request in the following format: `GET` `index/type/id`

Let us try to retrieve the details of the customer with ‘id’ equals 2:

```
GET /customer/US/2
```

It will give you the following type of result, on executing successfully.



```
{
  "_index": "customer",
  "_type": "US",
  "_id": "2",
  "_version": 1,
  "found": true,
  "_source": {
    "ID": 102,
    "FName": "Josephine",
    "LName": "Darakjy",
    "Email": "josephine_darakjy@darakjy.org",
    "City": "Brighton",
    "Type": "New"
  }
}
```

With Elasticsearch, you can not only browse through the data, but you can delete or remove the documents as well.

Deleting Data

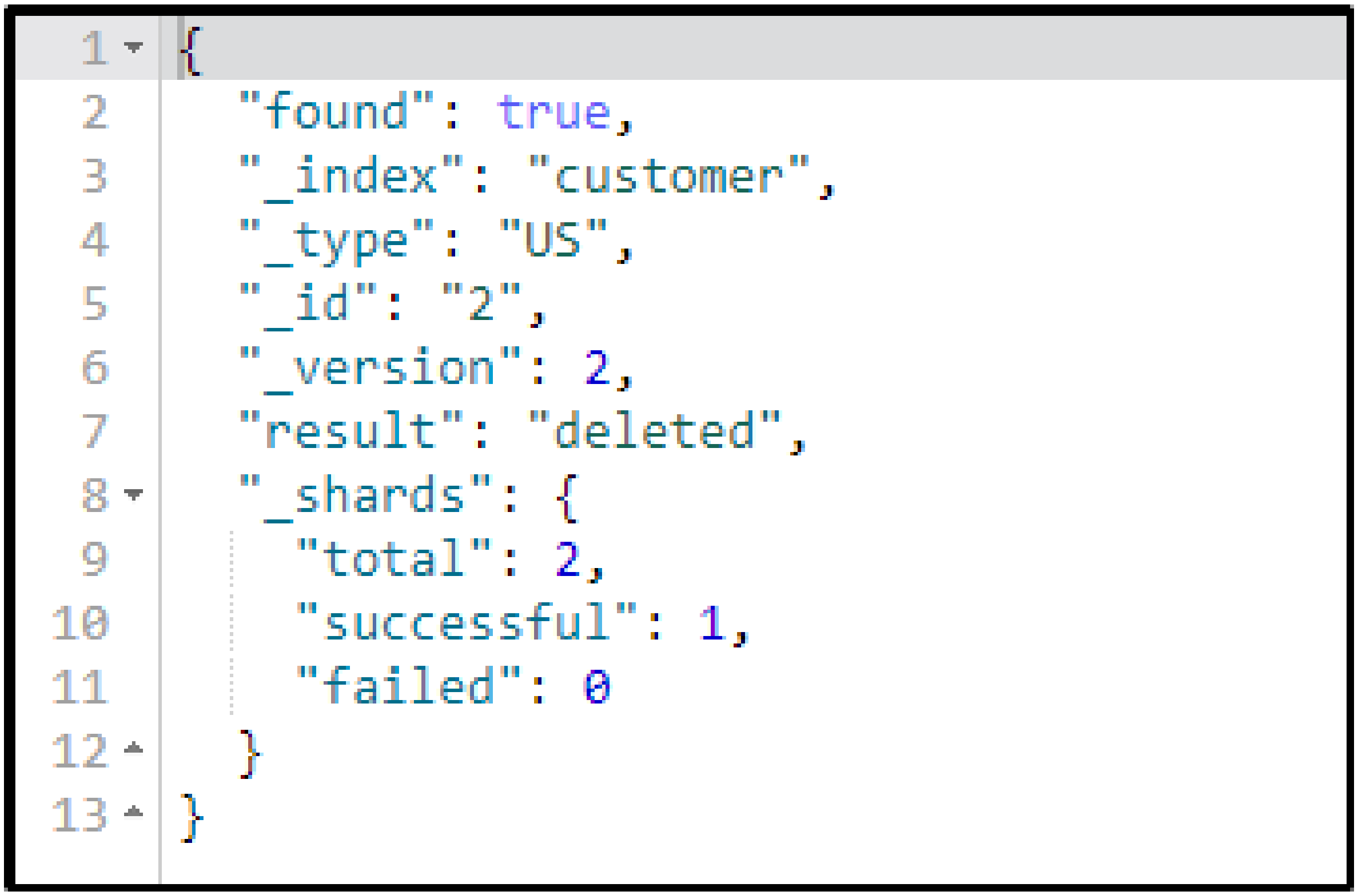
Using the delete convention you can easily remove the unwanted data from your index and free up the memory space. To delete any document you need to send an HTTP DELETE request in the following format: `DELETE index/type/id.`

Lets now try to delete the details of a customer with id 2.

```
DELETE /customer/US/2
```

On executing this query, you will get the following type of result.

So, this concludes the basics of CRUD operations using Elasticsearch. Knowing these basic operations will help you perform a different kind of searches and you are good enough to proceed with the ELK Tutorial. But if you want to learn Elasticsearch in depth, you can refer to my blog on Elasticsearch Tutorial.

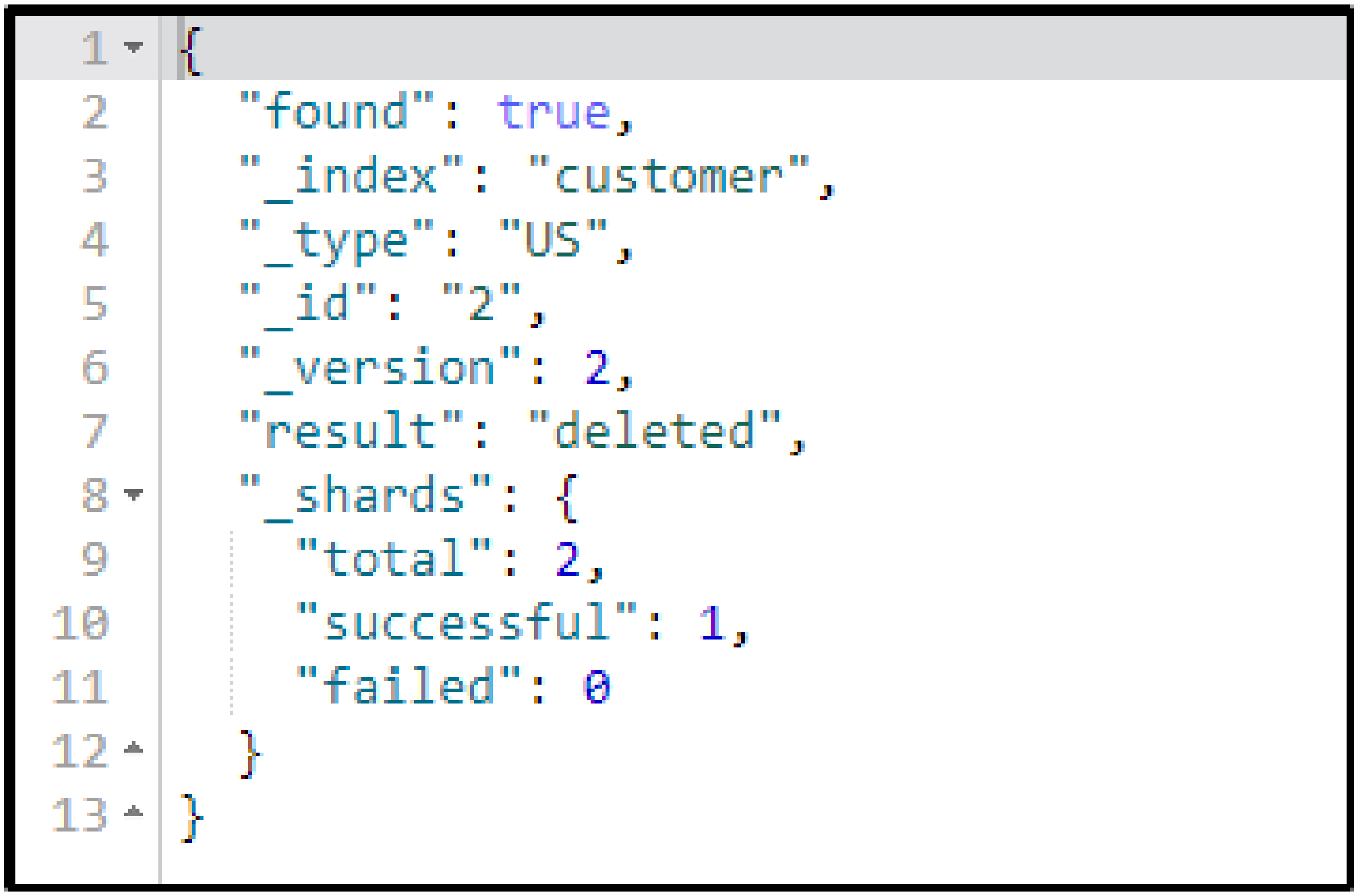


```
{
  "found": true,
  "_index": "customer",
  "_type": "US",
  "_id": "2",
  "_version": 2,
  "result": "deleted",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  }
}
```

Lets now begin with next tool of ELK Stack, which is Logstash.

On executing this query, you will get the following type of result.

So, this concludes the basics of CRUD operations using Elasticsearch. Knowing these basic operations will help you perform a different kind of searches and you are good enough to proceed with the ELK Tutorial. But if you want to learn Elasticsearch in depth, you can refer to my blog on Elasticsearch Tutorial.

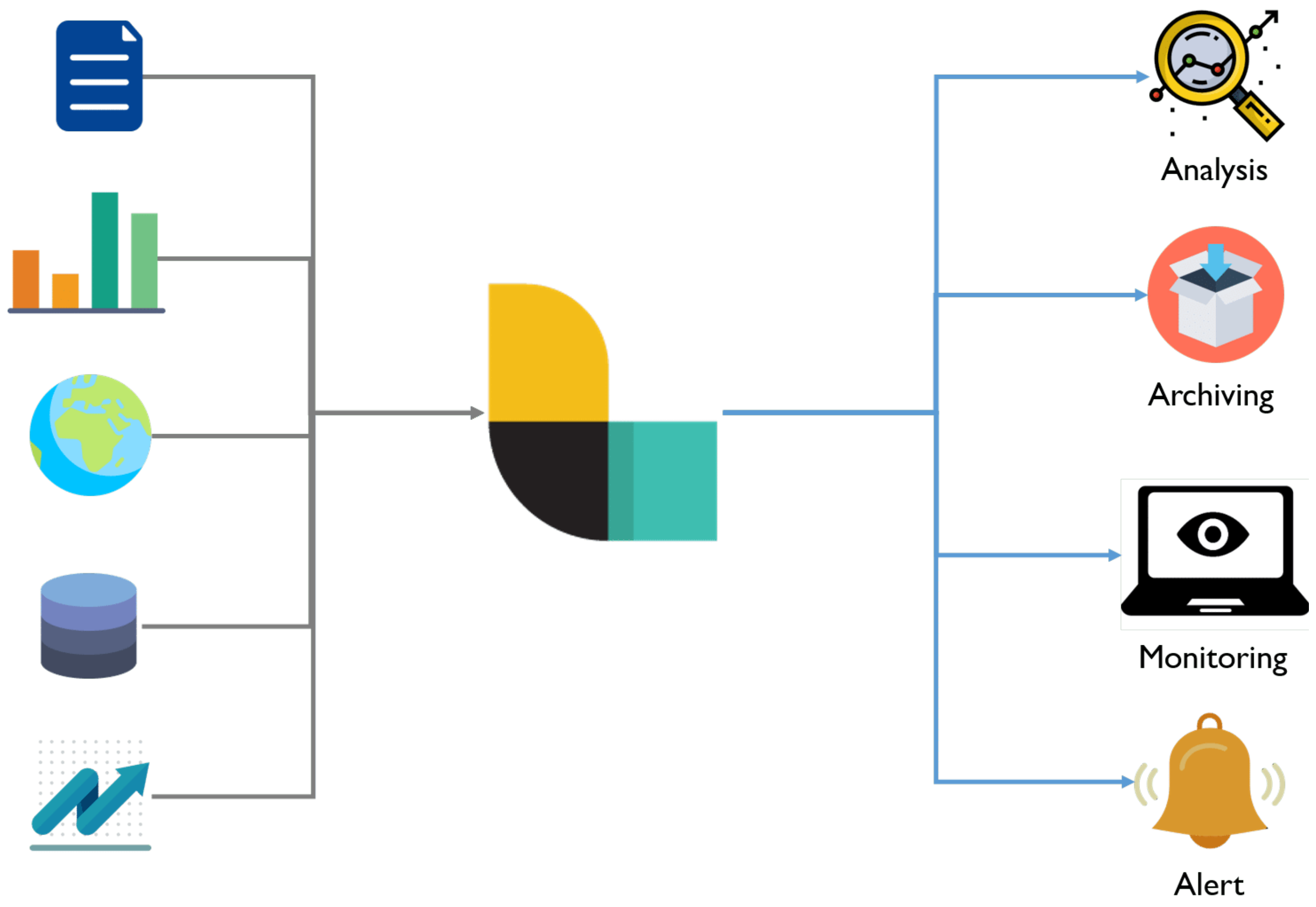


```
{
  "found": true,
  "_index": "customer",
  "_type": "US",
  "_id": "2",
  "_version": 2,
  "result": "deleted",
  "_shards": {
    "total": 2,
    "successful": 1,
    "failed": 0
  }
}
```

Lets now begin with next tool of ELK Stack, which is Logstash.

Logstash – ELK Stack Tutorial

Logstash is a pipeline tool generally used for collecting and forwarding the logs or events. It is an open source data collection engine which can dynamically integrate data from various sources and normalize it into the specified destinations.



Using a number of input, filter, and output plugins, Logstash enables the easy transformation of various events. At the very least, Logstash needs an input and an output plugin specified in its configurational file to perform the transformations. Following is the structure of a Logstash config file:

```
input {
  ...
}

filter {
  ...
}

output {
  ...
}
```

As you can see, the entire configuration file is divided into three sections and each of these sections holds the configuration options for one or more plugins. The three sections are:

1. input
2. filter
3. output

You can apply more than one filter in your config file as well. In such cases, the order of their application will be the same as the order of specification in the config file.

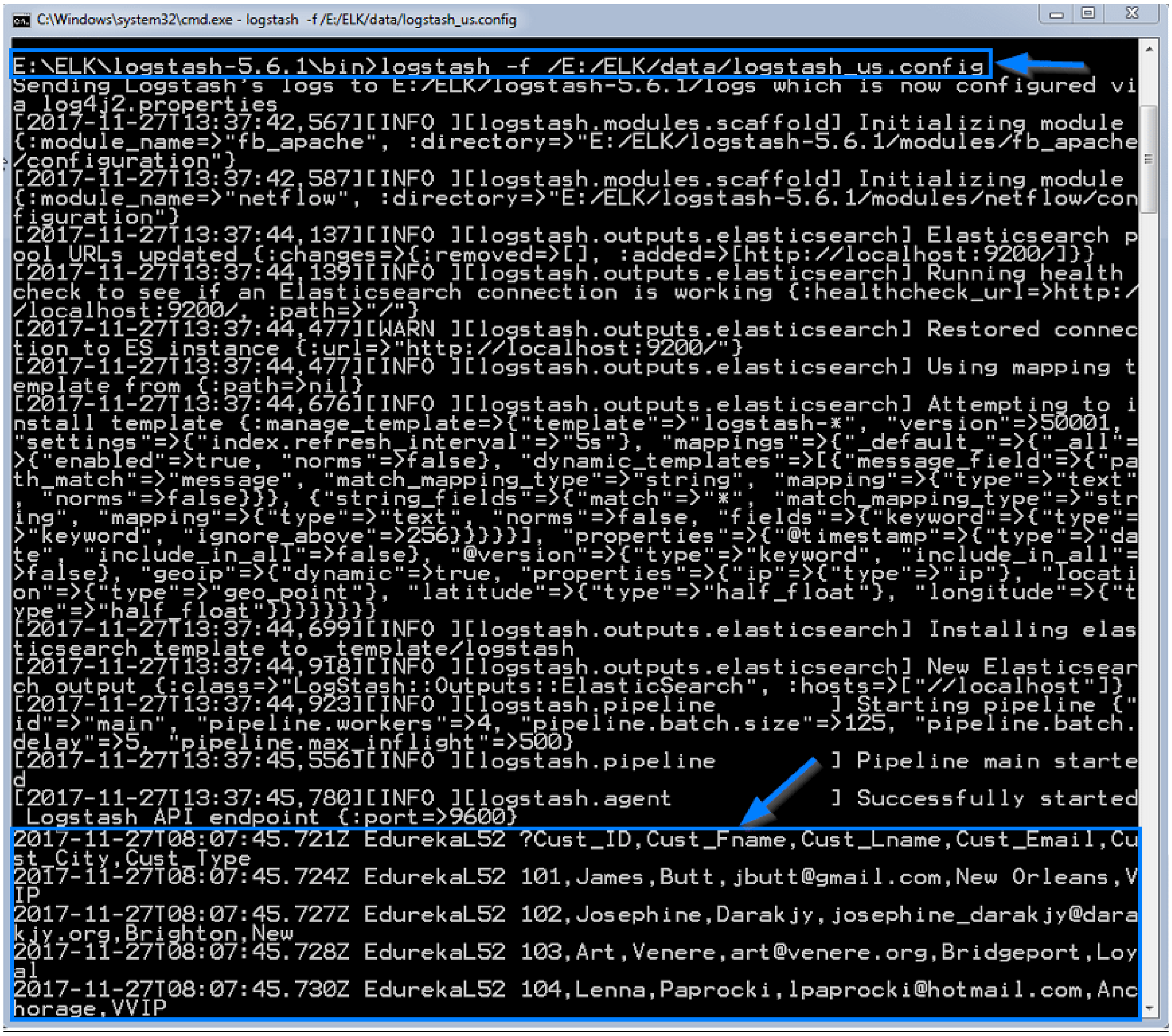
Lets now try to configure our US customer data set file which is in CSV file format.

```
input{
          file{
          path => "E:/ELK/data/US_Customer_List.csv"
          start_position => "beginning"
          sincedb_path => "/dev/null"
        }
}
filter{
     csv{
     separator => ","
     columns
=>["Cust_ID","Cust_Fname","Cust_Lname","Cust_Email","Cust_City","Cust_Type"]
     }
     mutate{convert => ["Cust_ID","integer"]}
}
output{
     elasticsearch{
     hosts => "localhost"
     index => "customers"
     document_type => "US_Based_Cust"
     }
     stdout{}
}
```

To insert this CSV file data into the elasticsearch you have to notify the Logstash server.

For that follow the below steps:

1. Open command prompt
2. Go the bin directory of Logstash
3. Type: **logstash –f X**:/foldername/config_filename.config and hit enter. Once your logstash server is up and running it will start pipelining your data from the file, into the Elasticsearch.


C:\Windows\system32\cmd.exe - logstash -f /E:/ELK/data/logstash_us.config

```
E:\ELK\logstash-5.6.1\bin>logstash -f /E:/ELK/data/logstash_us.config
Sending Logstash's logs to E:/ELK/logstash-5.6.1/logs which is now configured vi
a log4j2.properties
[2017-11-27T13:37:42,567][INFO ][logstash.modules.scaffold] Initializing module
{:module_name=>"fb_apache", :directory=>"E:/ELK/logstash-5.6.1/modules/fb_apache
/configuration"}
[2017-11-27T13:37:42,587][INFO ][logstash.modules.scaffold] Initializing module
{:module_name=>"netflow", :directory=>"E:/ELK/logstash-5.6.1/modules/netflow/con
figuration"}
[2017-11-27T13:37:44,137][INFO ][logstash.outputs.elasticsearch] Elasticsearch p
ool URLs updated {:changes=>{:removed=>[], :added=>[http://localhost:9200/]}}
[2017-11-27T13:37:44,139][INFO ][logstash.outputs.elasticsearch] Running health
check to see if an Elasticsearch connection is working {:healthcheck_url=>http:/
/localhost:9200/, :path=>"/"}
[2017-11-27T13:37:44,477][WARN ][logstash.outputs.elasticsearch] Restored connec
tion to ES instance {:url=>"http://localhost:9200/"}
[2017-11-27T13:37:44,477][INFO ][logstash.outputs.elasticsearch] Using mapping t
emplate from {:path=>nil}
[2017-11-27T13:37:44,676][INFO ][logstash.outputs.elasticsearch] Attempting to i
nstall template {:manage_template=>{"template"=>"logstash-*", "version"=>50001,
"settings"=>{"index.refresh_interval"=>"5s"}, "mappings"=>{"_default_"=>{"_all"=
>{"enabled"=>true, "norms"=>false}, "dynamic_templates"=>[{"message_field"=>{"pa
th_match"=>"message", "match_mapping_type"=>"string", "mapping"=>{"type"=>"text"
, "norms"=>false}}}, {"string_fields"=>{"match"=>"*", "match_mapping_type"=>"str
ing", "mapping"=>{"type"=>"text", "norms"=>false, "fields"=>{"keyword"=>{"type"=
>"keyword", "ignore_above"=>256}}}}}], "properties"=>{"@timestamp"=>{"type"=>"da
te", "include_in_all"=>false}, "@version"=>{"type"=>"keyword", "include_in_all"=
>false}, "geoip"=>{"dynamic"=>true, "properties"=>{"ip"=>{"type"=>"ip"}, "locati
on"=>{"type"=>"geo_point"}, "latitude"=>{"type"=>"half_float"}, "longitude"=>{"t
ype"=>"half_float"}}}}}}}}
[2017-11-27T13:37:44,699][INFO ][logstash.outputs.elasticsearch] Installing elas
ticsearch template to _template/logstash
[2017-11-27T13:37:44,918][INFO ][logstash.outputs.elasticsearch] New Elasticsear
ch output {:class=>"LogStash::Outputs::ElasticSearch", :hosts=>["//localhost"]}
[2017-11-27T13:37:44,923][INFO ][logstash.pipeline        ] Starting pipeline {"
id"=>"main", "pipeline.workers"=>4, "pipeline.batch.size"=>125, "pipeline.batch.
delay"=>5, "pipeline.max_inflight"=>500}
[2017-11-27T13:37:45,556][INFO ][logstash.pipeline        ] Pipeline main starte
d
[2017-11-27T13:37:45,780][INFO ][logstash.agent           ] Successfully started
 Logstash API endpoint {:port=>9600}
2017-11-27T08:07:45.721Z EdurekaL52 ?Cust_ID,Cust_Fname,Cust_Lname,Cust_Email,Cu
st_City,Cust_Type
2017-11-27T08:07:45.724Z EdurekaL52 101,James,Butt,jbutt@gmail.com,New Orleans,V
IP
2017-11-27T08:07:45.727Z EdurekaL52 102,Josephine,Darakjy,josephine_darakjy@dara
kjy.org,Brighton,New
2017-11-27T08:07:45.728Z EdurekaL52 103,Art,Venere,art@venere.org,Bridgeport,Loy
al
2017-11-27T08:07:45.730Z EdurekaL52 104,Lenna,Paprocki,lpaprocki@hotmail.com,Anc
horage,VVIP
```

If you want to check whether your data was inserted successfully or not, go to the sense plugin and type:
`GET /customers/`

It will give you the number of documents that have been created.

Now if you want to visualize this data, you have to make use of the last tool of ELK Stack i.e Kibana. So, in the next section of this ELK Stack Tutorial, I will be discussing Kibana and the ways to use, it to visualize your data.

Kibana – ELK Stack Tutorial

As mentioned earlier, Kibana is an open source visualization and analytics tool. It helps in visualizing the data that is piped down by the Logstash and is stored into the Elasticsearch. You can use Kibana to search, view, and interact with this stored data and then visualize it in various charts, tables, and maps.The browser-based interface of Kibana simplifies the huge volumes of data and reflects the real-time changes in the Elasticsearch queries. Moreover, you can easily create, customize, save and share your dashboards as well.

Once you have learned, how to work with Elasticsearch and Logstash, leaning Kibana becomes no big deal. In this section of the ELK tutorial blog, I will introduce you to the different functions which you need in order to perform the analysis on your data.

- **Management Page**

  This is where you have to perform your runtime configurations of Kibana. In this page, you need to specify few things for your search. See the following example, in which I have configured the entries for my ‘customer’ index.Here as you can see, in the ‘Index pattern’ field you need to specify the index on which you want to use. In the ‘Time Filter field name’ make sure you select it as @timestamp. Then you can go ahead and click on Create in order to create the index. If your index is created successfully you will see the following type of page:Here you can choose for different filters from the drop-down list, according to your requirements. Moreover, to free up your memory you can delete a particular index as well.

- **Discover Page**

  Through the Discover page, you have access to every document present each index which matches the selected index pattern. You can easily interact and explore, every bit of data that is present on your Kibana server. Moreover, you can view the data present in the documents and perform search queries on it.Below you can see, I am performing a search for the 'VIP' customers hailing from 'Los Angeles'.So, as you can see, we have only one VIP customer from Los Angeles.

- **Visualize Page**

  Visualize page enables you to visualize the data present in your Elasticsearch indices, in the form of charts, bars, pies etc. Here you can even build the dashboards which will display the related visualizations based on Elasticsearch queries. Generally, a series of Elasticsearch aggregation queries are used to extract and process the data. When you go to the Visualize page and search for your saved visualizations or you can create a new one.

  You can aggregate your data in any form. For the user's ease, different types of visualization options are provided.Let me show you how can you visualize the US customer data based on the user types. To perform the visualization, follow the below steps:

  1. Select the visualization type. [Here I am using a Pie]
  2. In the aggregation field, select 'term' from the drop-down list.
  3. In the 'field', select the field type based on which you want to perform the search.
  4. You can specify the order and the size of your visualizations as well.
  5. Now click on the execute button to generate the Pie chart.

- **Dashboard Page**

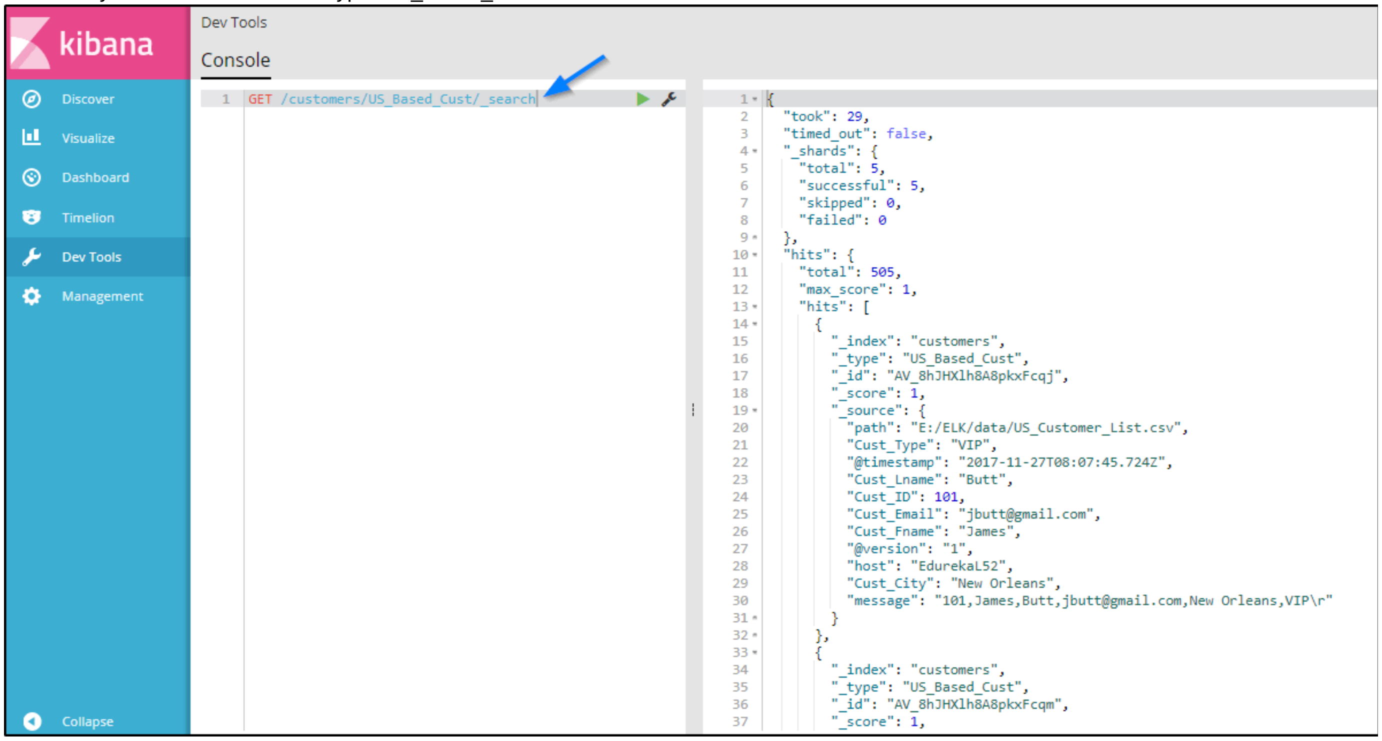
  The Dashboard page displays a collection of saved visualizations. Here you can either add new visualizations or you can use any saved visualization as well.

- **Timelion Page**

  Timelion is a time series data visualizer which brings together totally independent data sources into a single interface. It is driven by a one-line expression language that one uses to retrieve time series data, perform calculations to simplify complex questions and visualize the results.

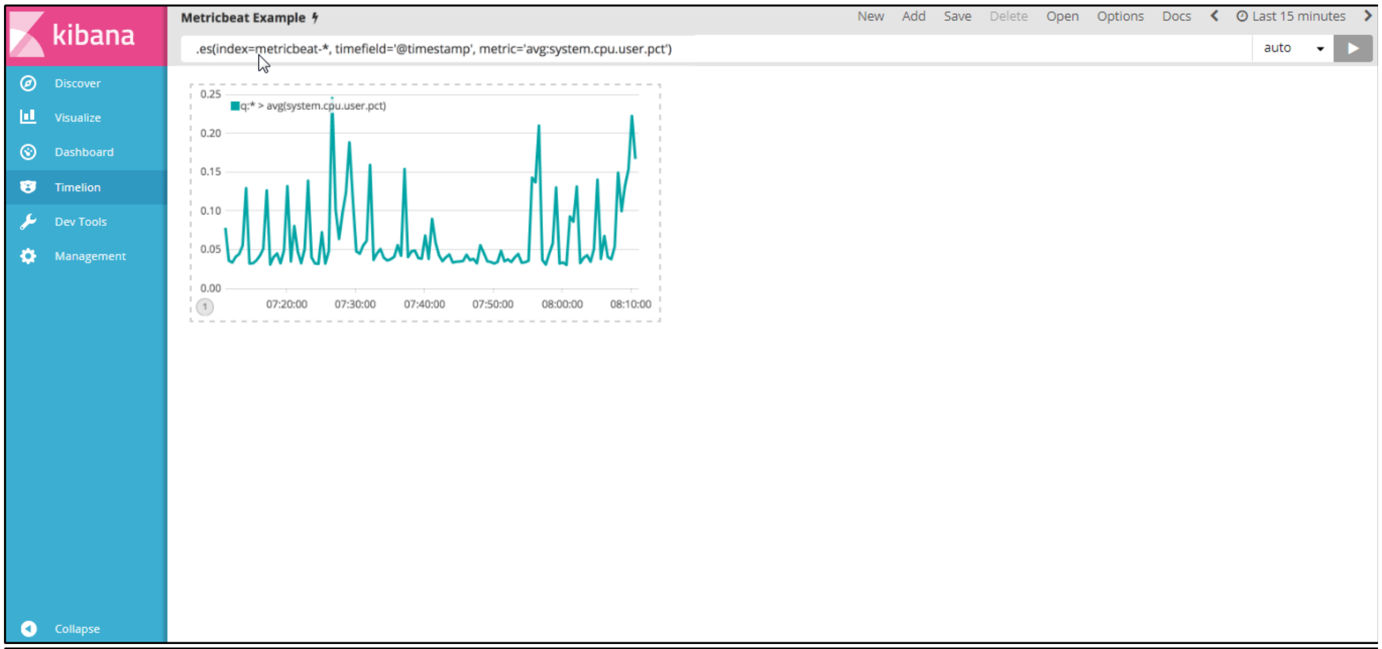
- **Dev Tools Page**

  The Dev Tools page of Kibana contains the development tools like 'Beta Sense' plugin, which is used to interact with the data present in Elasticsearch. It is often referred as Kibana's console. Following is an example in which I have used Kibana's sense plugin to search for my 'customers' index with type 'US_based_cust':

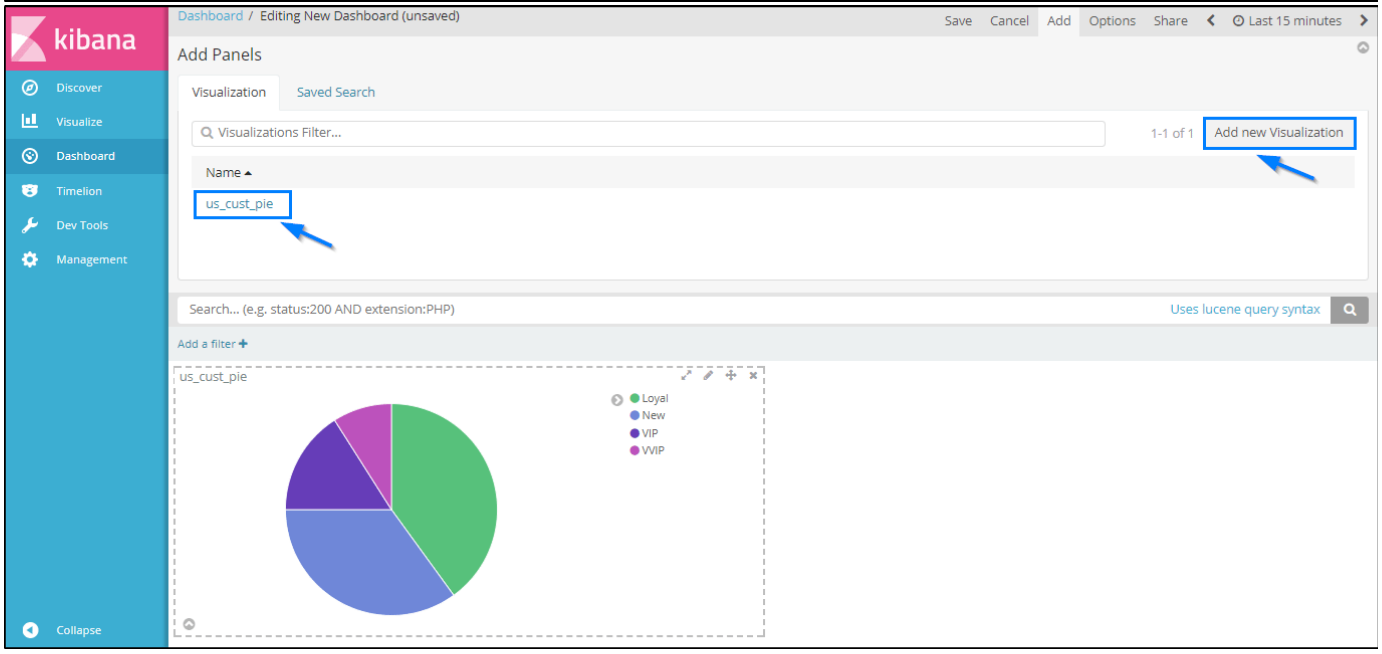


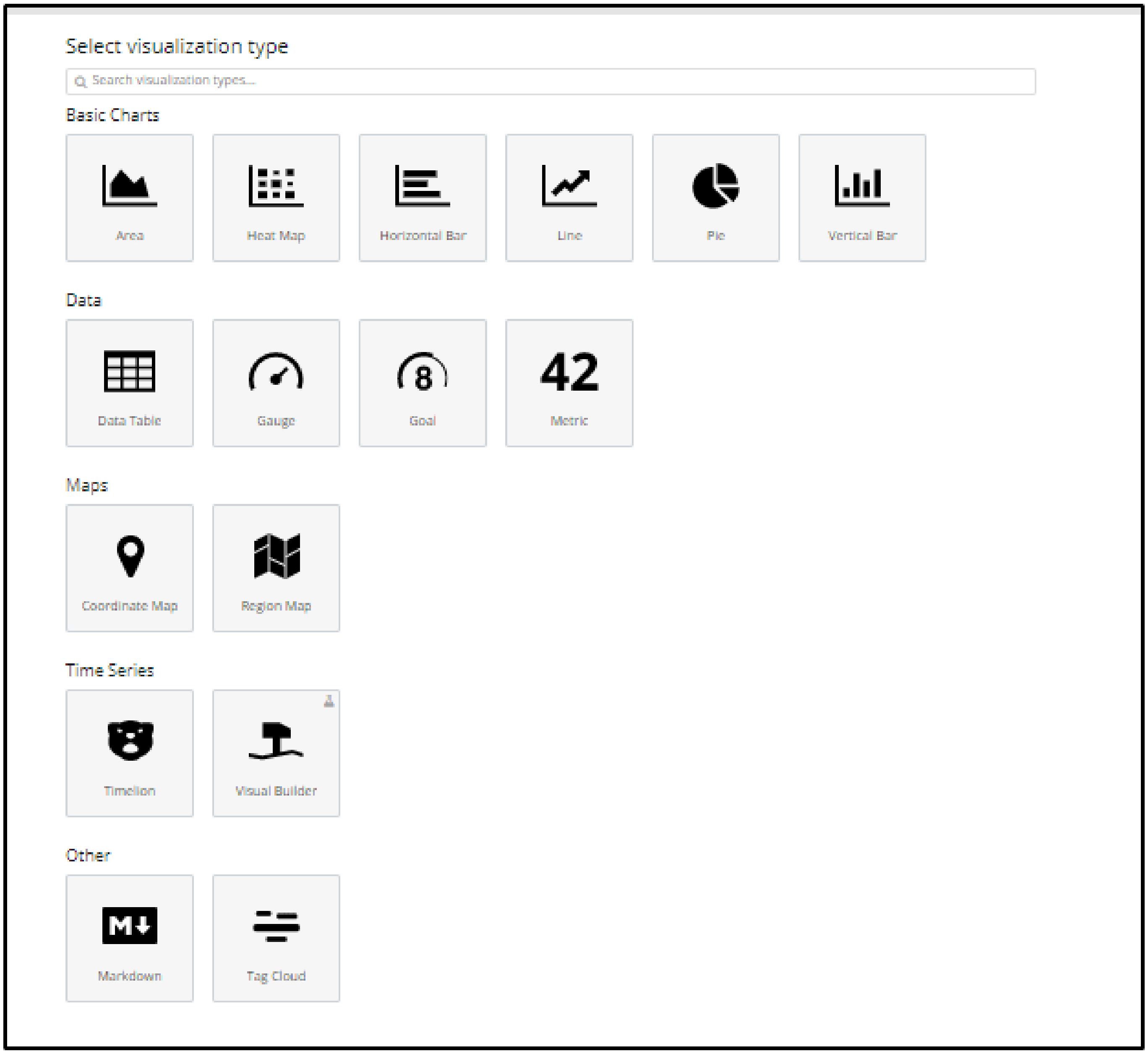
```
GET /customers/US_Based_Cust/_search
```

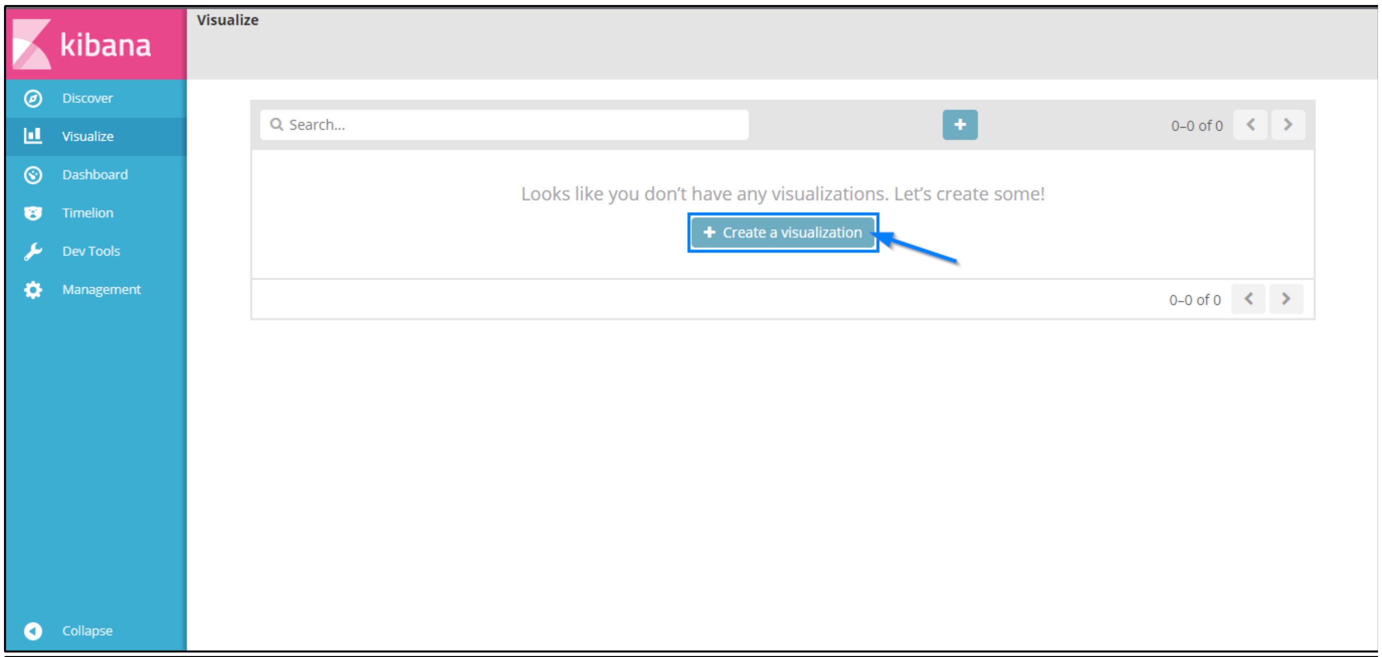
```
{
  "took": 29,
  "timed_out": false,
  "_shards": {
    "total": 5,
    "successful": 5,
    "skipped": 0,
    "failed": 0
  },
  "hits": {
    "total": 505,
    "max_score": 1,
    "hits": [
      {
        "_index": "customers",
        "_type": "US_Based_Cust",
        "_id": "AV_8hJHXlh8A8pkxFcqj",
        "_score": 1,
        "_source": {
          "path": "E:/ELK/data/US_Customer_List.csv",
          "Cust_Type": "VIP",
          "@timestamp": "2017-11-27T08:07:45.724Z",
          "Cust_Lname": "Butt",
          "Cust_ID": 101,
          "Cust_Email": "jbutt@gmail.com",
          "Cust_Fname": "James",
          "@version": "1",
          "host": "EdurekaL52",
          "Cust_City": "New Orleans",
          "message": "101,James,Butt,jbutt@gmail.com,New Orleans,VIP\r"
        }
      },
      {
        "_index": "customers",
        "_type": "US_Based_Cust",
        "_id": "AV_8hJHXlh8A8pkxFcqm",
        "_score": 1,
```



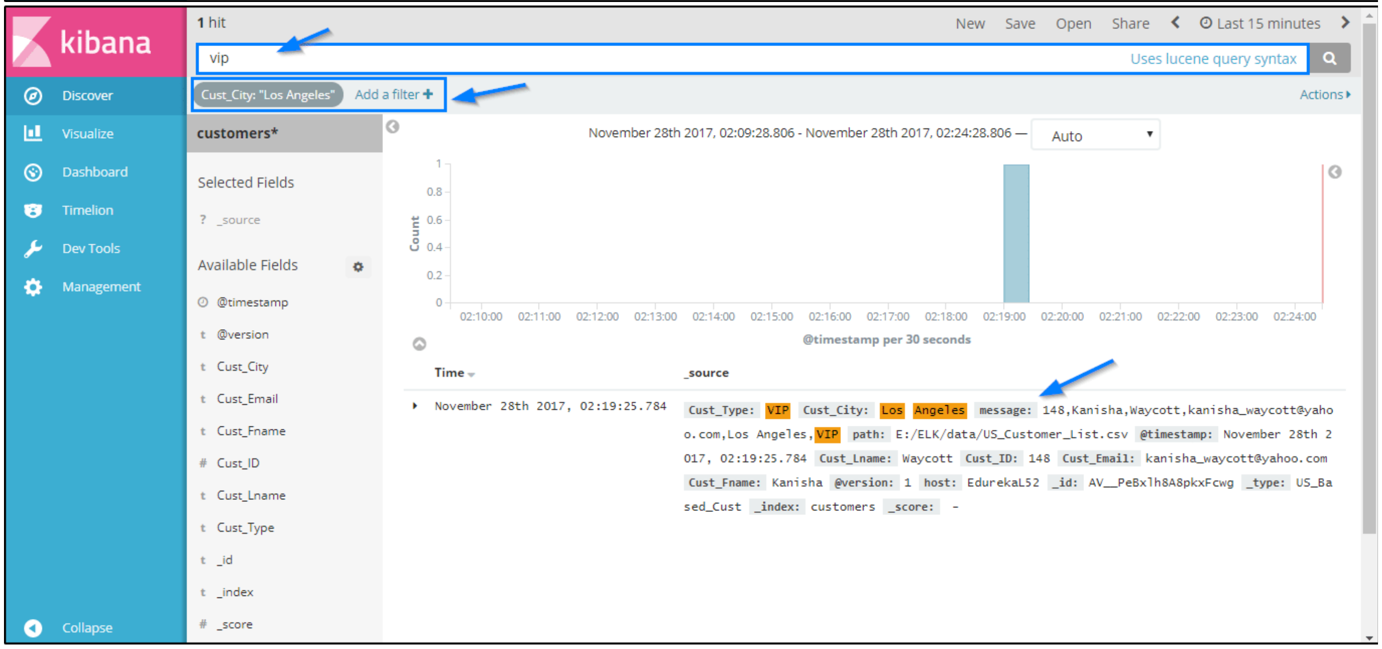
```
.es(index=metricbeat-*, timefield='@timestamp', metric='avg:system.cpu.user.pct')
```




Select visualization type
Search visualization types...
Basic Charts
Area
Heat Map
Horizontal Bar
Line
Pie
Vertical Bar
Data
Data Table
Gauge
Goal
42
Metric
Maps
Coordinate Map
Region Map
Time Series
Timelion
Visual Builder
Other
Markdown
Tag Cloud



Looks like you don't have any visualizations. Let's create some!

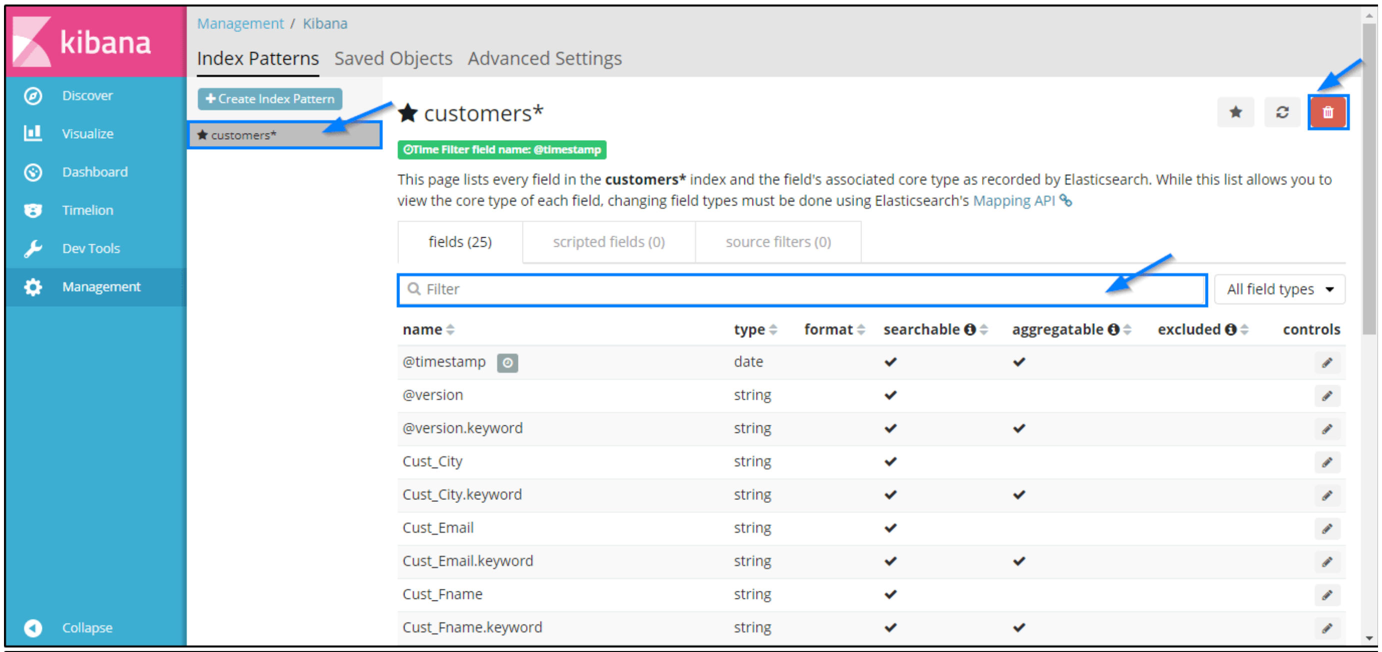


vip

Cust_City: "Los Angeles"

November 28th 2017, 02:09:28.806 - November 28th 2017, 02:24:28.806 —

```
November 28th 2017, 02:19:25.784   Cust_Type: VIP Cust_City: Los Angeles message: 148,Kanisha,Waycott,kanisha_waycott@yaho
                                   o.com,Los Angeles,VIP path: E:/ELK/data/US_Customer_List.csv @timestamp: November 28th 2
                                   017, 02:19:25.784 Cust_Lname: Waycott Cust_ID: 148 Cust_Email: kanisha_waycott@yahoo.com
                                   Cust_Fname: Kanisha @version: 1 host: EdurekaL52 _id: AV__PeBxlh8A8pkxFcwg _type: US_Ba
                                   sed_Cust _index: customers _score: -
```



★ customers*

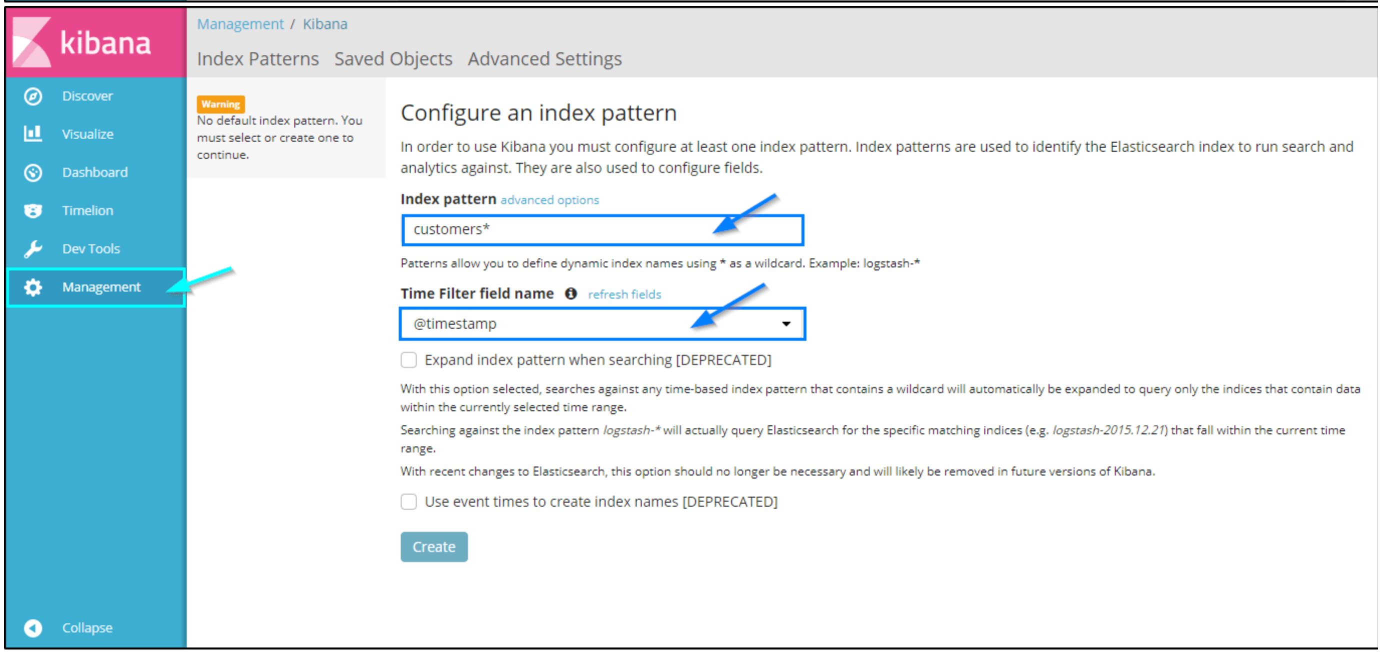
Time Filter field name: @timestamp

This page lists every field in the **customers*** index and the field's associated core type as recorded by Elasticsearch. While this list allows you to view the core type of each field, changing field types must be done using Elasticsearch's Mapping API

fields (25) | scripted fields (0) | source filters (0)

Filter | All field types

name	type	format	searchable	aggregatable	excluded	controls
@timestamp	date		✓	✓		
@version	string		✓			
@version.keyword	string		✓	✓		
Cust_City	string		✓			
Cust_City.keyword	string		✓	✓		
Cust_Email	string		✓			
Cust_Email.keyword	string		✓	✓		
Cust_Fname	string		✓			
Cust_Fname.keyword	string		✓	✓		



Warning
No default index pattern. You must select or create one to continue.

Configure an index pattern

In order to use Kibana you must configure at least one index pattern. Index patterns are used to identify the Elasticsearch index to run search and analytics against. They are also used to configure fields.

Index pattern advanced options

customers*

Patterns allow you to define dynamic index names using * as a wildcard. Example: logstash-*

Time Filter field name refresh fields

@timestamp

☐ Expand index pattern when searching [DEPRECATED]

With this option selected, searches against any time-based index pattern that contains a wildcard will automatically be expanded to query only the indices that contain data within the currently selected time range.

Searching against the index pattern *logstash-** will actually query Elasticsearch for the specific matching indices (e.g. *logstash-2015.12.21*) that fall within the current time range.

With recent changes to Elasticsearch, this option should no longer be necessary and will likely be removed in future versions of Kibana.

☐ Use event times to create index names [DEPRECATED]

Create

This concludes this blog on ELK Stack Tutorial. Now you are ready to perform various search and analysis on any data you want, using Logstash, Elasticsearch, and Kibana.