Elasticsearch Workshop

This documentation is Microsoft Windows specific (used on Windows 2012 R2).

Getting Started

https://github.com/elastic/elasticsearch/edit/5.0/docs/reference/getting-started.asciidoc

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

Here are a few sample use-cases that Elasticsearch could be used for:

- You run an online web store where you allow your customers to search for products that you sell. In this case, you can use Elasticsearch to store your entire product catalog and inventory and provide search and autocomplete suggestions for them.
- You want to collect log or transaction data and you want to analyze and mine this data to look for trends, statistics, summarizations, or anomalies. In this case, you can use Logstash (part of the Elasticsearch/Logstash/Kibana stack) to collect, aggregate, and parse your data, and then have Logstash feed this data into Elasticsearch. Once the data is in Elasticsearch, you can run searches and aggregations to mine any information that is of interest to you.
- You run a price alerting platform which allows price-savvy customers to specify a rule like "I am interested in buying a specific electronic gadget and I want to be notified if the price of gadget falls below $X from any vendor within the next month". In this case you can scrape vendor prices, push them into Elasticsearch and use its reverse-search (Percolator) capability to match price movements against customer queries and eventually push the alerts out to the customer once matches are found.
- You have analytics/business-intelligence needs and want to quickly investigate, analyze, visualize, and ask ad-hoc questions on a lot of data (think millions or billions of records). In this case, you can use Elasticsearch to store your data and then use Kibana (part of the Elasticsearch/Logstash/Kibana stack) to build custom dashboards that can visualize aspects of your data that are important to you. Additionally, you can use the Elasticsearch aggregations functionality to perform complex business intelligence queries against your data.

For the rest of this tutorial, I will guide you through the process of getting Elasticsearch up and running, taking a peek inside it, and performing basic operations like indexing, searching, and modifying your data. At the end of this tutorial, you should have a good idea of what Elasticsearch is, how it works, and hopefully be inspired to see how you can use it to either build sophisticated search applications or to mine intelligence from your data.

Basic Concepts:

https://www.elastic.co/guide/en/elasticsearch/reference/current/_basic_concepts.html

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.
Make sure that you don't reuse the same cluster names in different environments, otherwise you might end up with nodes joining the wrong cluster. For instance you could use `logging-dev`,`logging-stage`, and `logging-prod` for the development, staging, and production clusters.

Note that it is valid and perfectly fine to have a cluster with only a single node in it. Furthermore, you may also have multiple independent clusters each with its own unique cluster 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. For example, let's assume you run a blogging platform and store all your data in a single index. In this index, you may define a type for user data, another type for blog data, and yet another type for comments data.

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 (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 and 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.

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 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`, the limit is `2,147,483,519` (= Integer.MAX_VALUE - 128) documents. You can monitor shard sizes using the`_cat/shards` api.

Installation:

https://www.elastic.co/guide/en/elasticsearch/reference/current/_installation.html

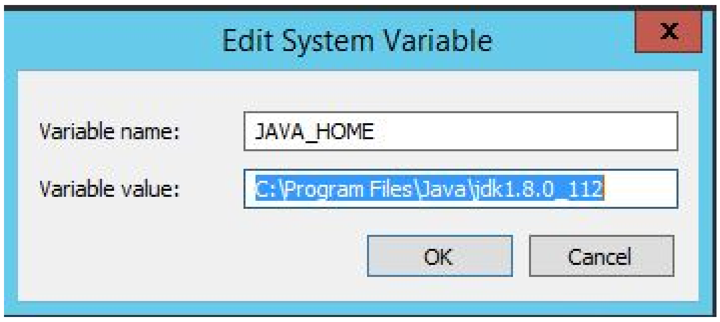
1. Download the latest Java JDK Windows from the following link:

   http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

   For the purpose of this document, we used **Java 8u112.**

2. Once installed, navigate to **System** in Control Panel. Click on **Advanced System Settings**, and then click the **Environment Variables** button.
3. Create a System variable as follows:

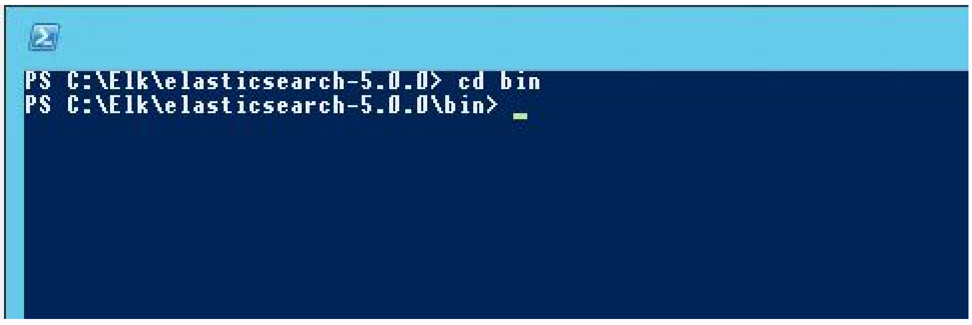
   Variable Name: JAVA_HOME
   Variable Value: <path to your Java JDK folder>



Variable name: JAVA_HOME
Variable value: C:\Program Files\Java\jdk1.8.0_112

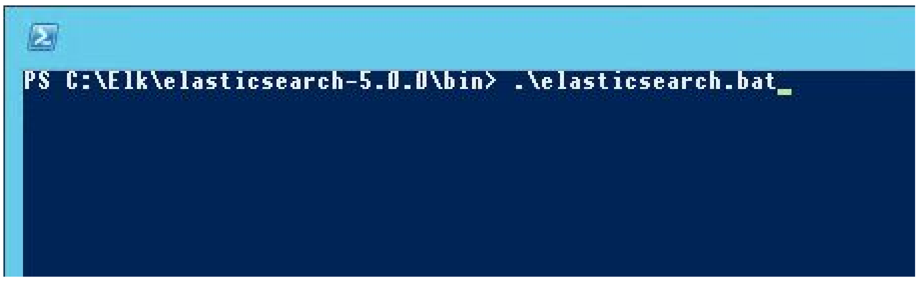
   Click OK and close all windows.

4. Browse to https://www.elastic.co/downloads/elasticsearch and download the ElasticSearch version 5.0.0 zip file.
5. Extract the ZIP file to your desired location.
6. Open a Command or Powershell window, and navigate to the extracted folder, then the \bin folder:



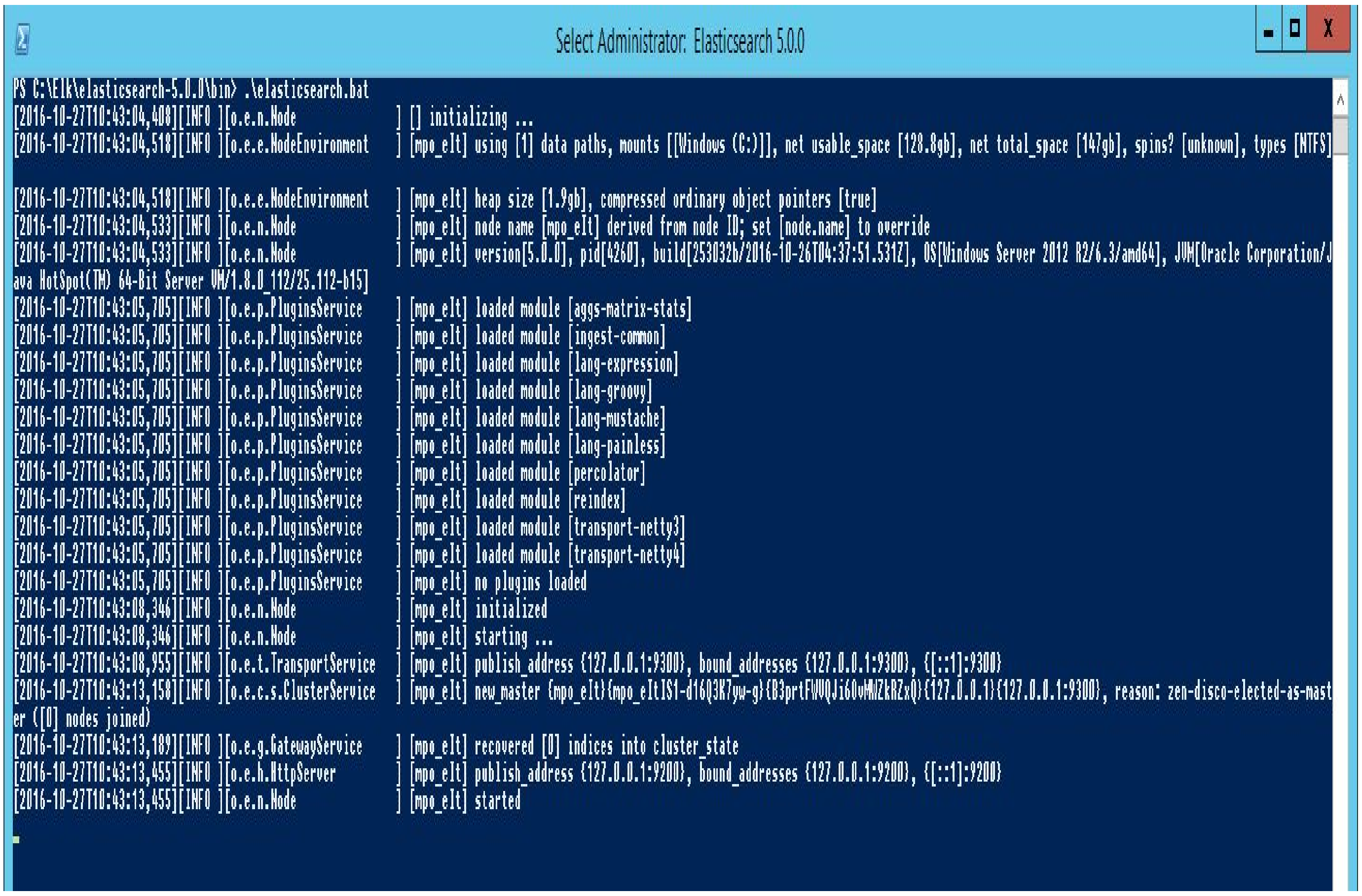
```
PS C:\Elk\elasticsearch-5.0.0> cd bin
PS C:\Elk\elasticsearch-5.0.0\bin> _
```

7. Execute the elasticsearch.bat file:



```
PS C:\Elk\elasticsearch-5.0.0\bin> .\elasticsearch.bat_
```

8. Output should look similar to the following:



```
PS C:\Elk\elasticsearch-5.0.0\bin> .\elasticsearch.bat
[2016-10-27T10:43:04,408][INFO ][o.e.n.Node               ] [] initializing ...
[2016-10-27T10:43:04,518][INFO ][o.e.e.NodeEnvironment    ] [mpo_elt] using [1] data paths, mounts [[Windows (C:)]], net usable_space [128.8gb], net total_space [147gb], spins? [unknown], types [NTFS]

[2016-10-27T10:43:04,518][INFO ][o.e.e.NodeEnvironment    ] [mpo_elt] heap size [1.9gb], compressed ordinary object pointers [true]
[2016-10-27T10:43:04,533][INFO ][o.e.n.Node               ] [mpo_elt] node name [mpo_elt] derived from node ID; set [node.name] to override
[2016-10-27T10:43:04,533][INFO ][o.e.n.Node               ] [mpo_elt] version[5.0.0], pid[4260], build[253032b/2016-10-26T04:37:51.531Z], OS[Windows Server 2012 R2/6.3/amd64], JVM[Oracle Corporation/J
ava HotSpot(TM) 64-Bit Server VM/1.8.0_112/25.112-b15]
[2016-10-27T10:43:05,705][INFO ][o.e.p.PluginsService     ] [mpo_elt] loaded module [aggs-matrix-stats]
[2016-10-27T10:43:05,705][INFO ][o.e.p.PluginsService     ] [mpo_elt] loaded module [ingest-common]
[2016-10-27T10:43:05,705][INFO ][o.e.p.PluginsService     ] [mpo_elt] loaded module [lang-expression]
[2016-10-27T10:43:05,705][INFO ][o.e.p.PluginsService     ] [mpo_elt] loaded module [lang-groovy]
[2016-10-27T10:43:05,705][INFO ][o.e.p.PluginsService     ] [mpo_elt] loaded module [lang-mustache]
[2016-10-27T10:43:05,705][INFO ][o.e.p.PluginsService     ] [mpo_elt] loaded module [lang-painless]
[2016-10-27T10:43:05,705][INFO ][o.e.p.PluginsService     ] [mpo_elt] loaded module [percolator]
[2016-10-27T10:43:05,705][INFO ][o.e.p.PluginsService     ] [mpo_elt] loaded module [reindex]
[2016-10-27T10:43:05,705][INFO ][o.e.p.PluginsService     ] [mpo_elt] loaded module [transport-netty3]
[2016-10-27T10:43:05,705][INFO ][o.e.p.PluginsService     ] [mpo_elt] loaded module [transport-netty4]
[2016-10-27T10:43:05,705][INFO ][o.e.p.PluginsService     ] [mpo_elt] no plugins loaded
[2016-10-27T10:43:08,346][INFO ][o.e.n.Node               ] [mpo_elt] initialized
[2016-10-27T10:43:08,346][INFO ][o.e.n.Node               ] [mpo_elt] starting ...
[2016-10-27T10:43:08,955][INFO ][o.e.t.TransportService   ] [mpo_elt] publish_address {127.0.0.1:9300}, bound_addresses {127.0.0.1:9300}, {[::1]:9300}
[2016-10-27T10:43:13,158][INFO ][o.e.c.s.ClusterService   ] [mpo_elt] new_master {mpo_elt}{mpo_eltIS1-d16Q3K7yw-g}{B3prtFWVQJi60vMWZkRZxQ}{127.0.0.1}{127.0.0.1:9300}, reason: zen-disco-elected-as-mast
er ([0] nodes joined)
[2016-10-27T10:43:13,189][INFO ][o.e.g.GatewayService     ] [mpo_elt] recovered [0] indices into cluster_state
[2016-10-27T10:43:13,455][INFO ][o.e.h.HttpServer         ] [mpo_elt] publish_address {127.0.0.1:9200}, bound_addresses {127.0.0.1:9200}, {[::1]:9200}
[2016-10-27T10:43:13,455][INFO ][o.e.n.Node               ] [mpo_elt] started
```

Exploring Your Cluster:

https://www.elastic.co/guide/en/elasticsearch/reference/current/_exploring_your_cluster.html

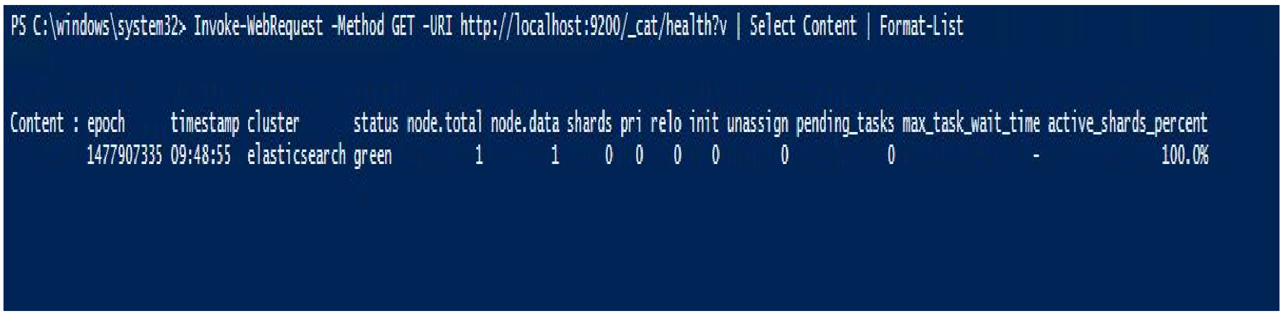
Cluster Health:

https://www.elastic.co/guide/en/elasticsearch/reference/current/_cluster_health.html
For the following section, we will be using the Powershell (version 3 or higher) commandlets. Issue the following command to determine your Powershell version: $psversiontable

1. Check the cluster health, by using the _cat API. Remember that the node HTTP endpoint is at port 9200:

   Invoke-WebRequest -Method GET -URI http://localhost:9200/_cat/health?v | Select Content | Format-List

2. The response would look similar to the following:

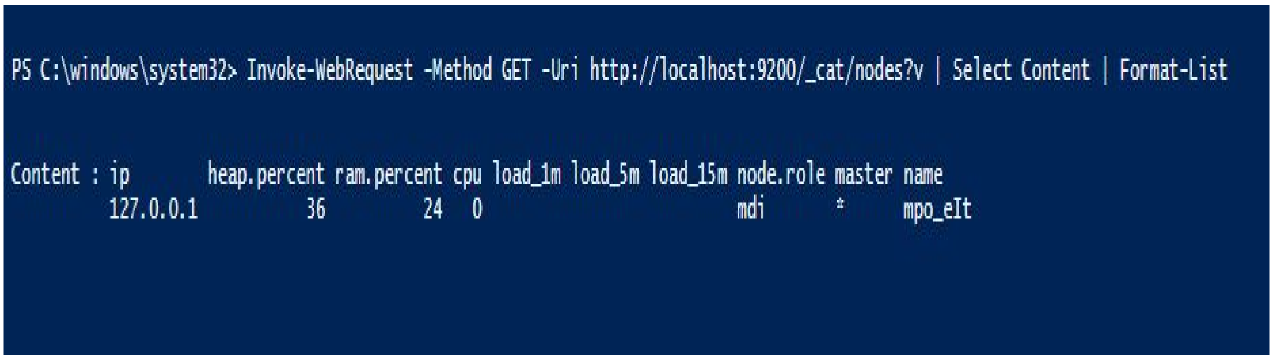


```
PS C:\windows\system32> Invoke-WebRequest -Method GET -URI http://localhost:9200/_cat/health?v | Select Content | Format-List


Content : epoch      timestamp cluster       status node.total node.data shards pri relo init unassign pending_tasks max_task_wait_time active_shards_percent
          1477907335 09:48:55  elasticsearch green           1         1      0   0    0    0        0             0                  -                100.0%
```

3. Get a list of nodes with the following command:

   Invoke-WebRequest -Method GET -Uri http://localhost:9200/_cat/nodes?v | Select Content | Format-List

4. The output should look similar to the following:



```
PS C:\windows\system32> Invoke-WebRequest -Method GET -Uri http://localhost:9200/_cat/nodes?v | Select Content | Format-List


Content : ip        heap.percent ram.percent cpu load_1m load_5m load_15m node.role master name
          127.0.0.1           36          24   0                          mdi       *      mpo_eIt
```

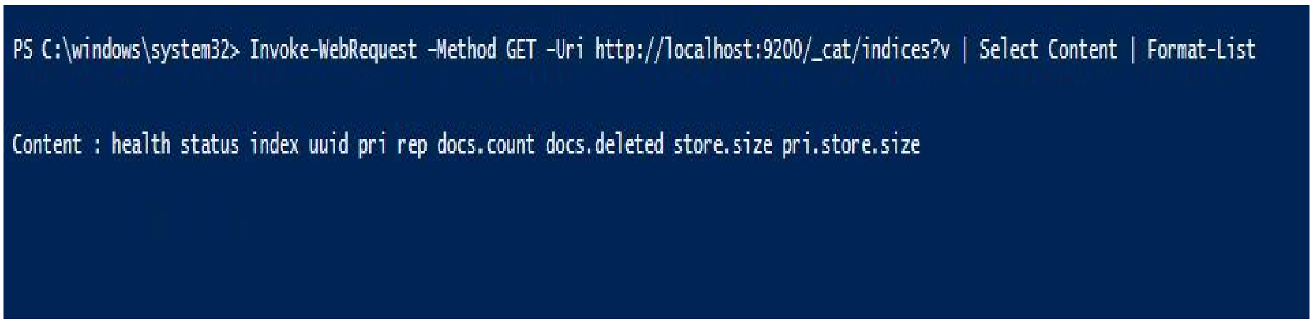
List All Indices:

https://www.elastic.co/guide/en/elasticsearch/reference/current/_list_all_indices.html

1. Take a look at the indices with the following command:

    Invoke-WebRequest –Method GET –Uri http://localhost:9200/_cat/indices?v | Select Content | Format-List

2. The output should look similar to the following:



```
PS C:\windows\system32> Invoke-WebRequest -Method GET -Uri http://localhost:9200/_cat/indices?v | Select Content | Format-List

Content : health status index uuid pri rep docs.count docs.deleted store.size pri.store.size
```

The above indicates that there are no indices yet in the cluster.

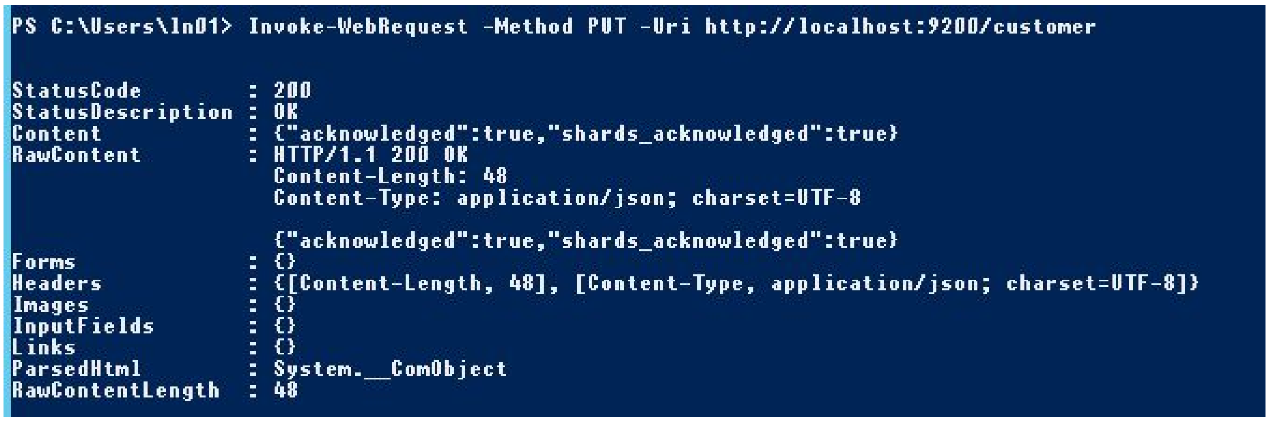
Create an Index:

https://www.elastic.co/guide/en/elasticsearch/reference/current/_create_an_index.html

1. Create an index named “customer” with the following command:

    Invoke-WebRequest -Method PUT -Uri http://localhost:9200/customer

   Output as follows:

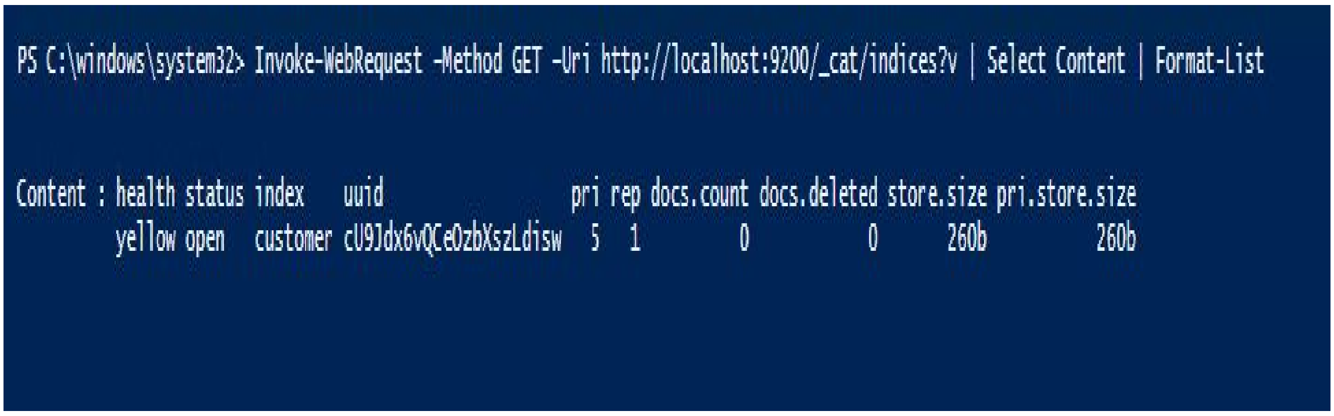


```
PS C:\Users\ln01> Invoke-WebRequest -Method PUT -Uri http://localhost:9200/customer


StatusCode        : 200
StatusDescription : OK
Content           : {"acknowledged":true,"shards_acknowledged":true}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 48
                    Content-Type: application/json; charset=UTF-8

                    {"acknowledged":true,"shards_acknowledged":true}
Forms             : {}
Headers           : {[Content-Length, 48], [Content-Type, application/json; charset=UTF-8]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 48
```

2. List the indexes again with the following command:

    Invoke-WebRequest –Method GET –Uri http://localhost:9200/_cat/indices?v | Select Content | Format-List

   Output as follows:



```
PS C:\windows\system32> Invoke-WebRequest -Method GET -Uri http://localhost:9200/_cat/indices?v | Select Content | Format-List


Content : health status index    uuid                   pri rep docs.count docs.deleted store.size pri.store.size
          yellow open   customer cU9Jdx6vQCeOzbXszLdisw   5   1          0            0       260b           260b
```

Index and Query a Document:

https://www.elastic.co/guide/en/elasticsearch/reference/current/_index_and_query_a_document.html#_index_and_query_a_document

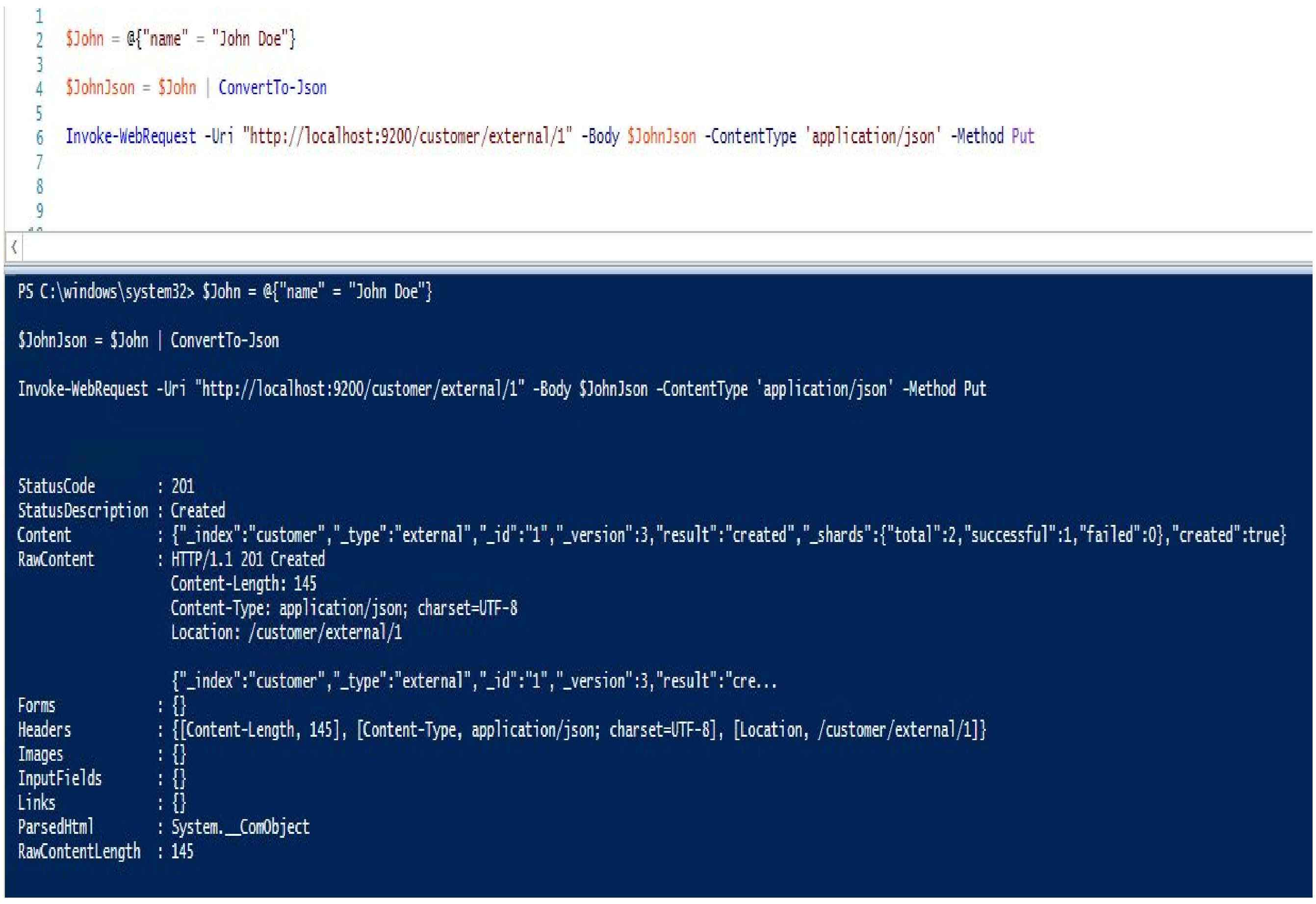
We will now enter something into the customer index. In order to **index** a **document**, we must tell Elasticsearch which **type** in the index it should go to.

As we are getting into more complex commands/scripts in Powershell, it is advised to use the **Powershell ISE** (or any other equivalent editor of your choice) for the following sections.

1. Index a customer document into the customer index, "external" type, with ID of "1" as follows:

```
$name = @{"name" = "John Doe"}
$json = $name | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body
$json -ContentType 'application/json' -Method Put
```

2. The output will look similar to the following:



```
$John = @{"name" = "John Doe"}

$JohnJson = $John | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body $JohnJson -ContentType 'application/json' -Method Put
```

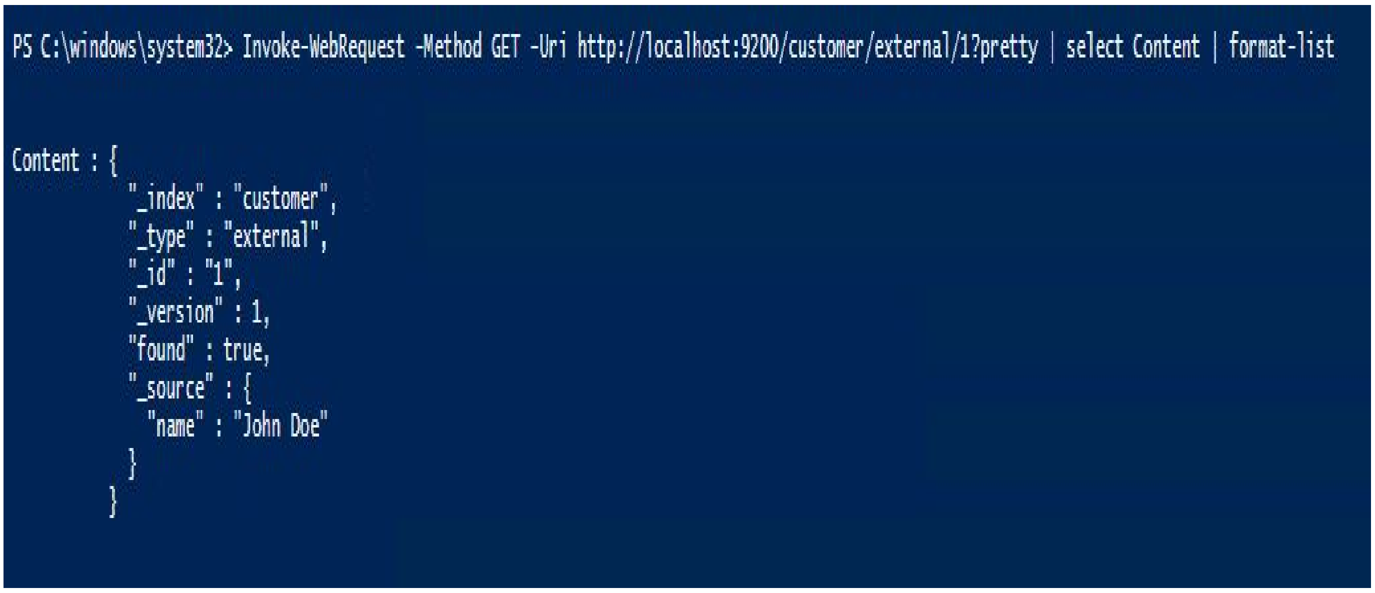
```
PS C:\windows\system32> $John = @{"name" = "John Doe"}

$JohnJson = $John | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body $JohnJson -ContentType 'application/json' -Method Put



StatusCode        : 201
StatusDescription : Created
Content           : {"_index":"customer","_type":"external","_id":"1","_version":3,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"created":true}
RawContent        : HTTP/1.1 201 Created
                    Content-Length: 145
                    Content-Type: application/json; charset=UTF-8
                    Location: /customer/external/1

                    {"_index":"customer","_type":"external","_id":"1","_version":3,"result":"cre...
Forms             : {}
Headers           : {[Content-Length, 145], [Content-Type, application/json; charset=UTF-8], [Location, /customer/external/1]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 145
```

The above indicates that a new customer document was successfully created inside the customer index and external type. Note: Elasticsearch does not require you to explicitly create an index first before you can create documents into it. Elasticsearch will automatically create the customer index if it didn't already exist beforehand.

3. Retrieve the document created in the index with the following command:

   Invoke-WebRequest -Method GET -Uri http://localhost:9200/customer/external/1?pretty | select Content | format-list

4. The output should look similar to the following:



```
PS C:\windows\system32> Invoke-WebRequest -Method GET -Uri http://localhost:9200/customer/external/1?pretty | select Content | format-list


Content : {
            "_index" : "customer",
            "_type" : "external",
            "_id" : "1",
            "_version" : 1,
            "found" : true,
            "_source" : {
              "name" : "John Doe"
            }
          }
```

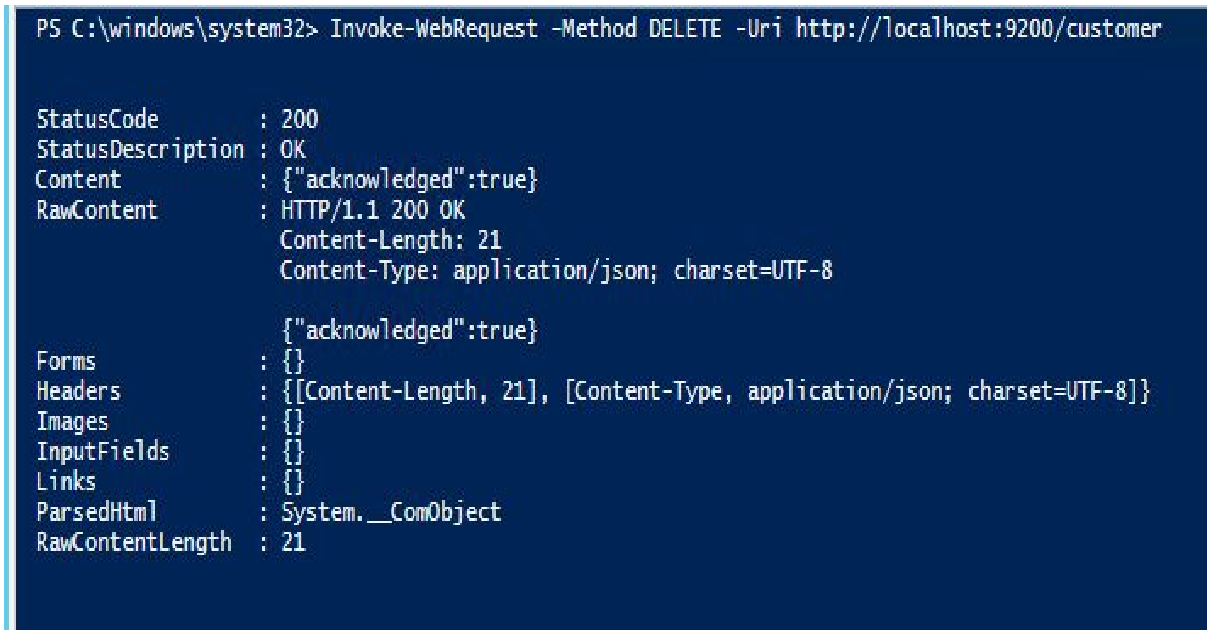
Delete an Index:

https://www.elastic.co/guide/en/elasticsearch/reference/current/_delete_an_index.html

1. Delete the index created, with the following command:

    Invoke-WebRequest -Method DELETE -Uri http://localhost:9200/customer

2. The output should look as follows:

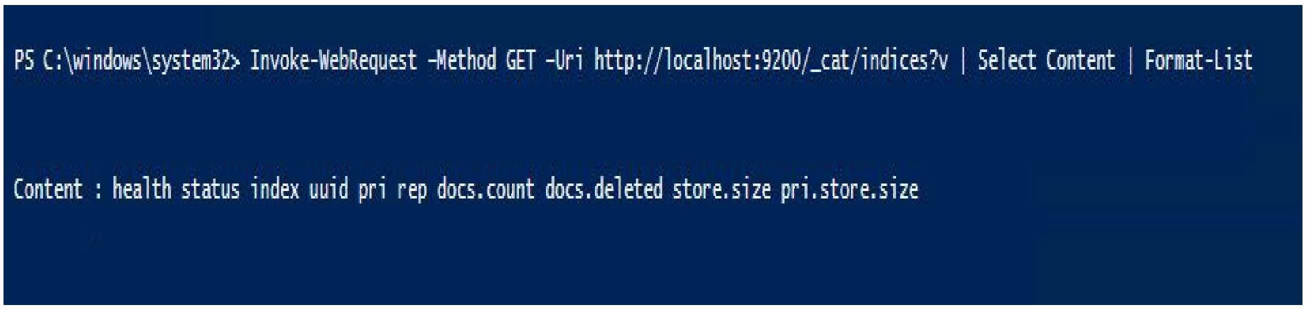


```
PS C:\windows\system32> Invoke-WebRequest -Method DELETE -Uri http://localhost:9200/customer


StatusCode        : 200
StatusDescription : OK
Content           : {"acknowledged":true}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 21
                    Content-Type: application/json; charset=UTF-8

                    {"acknowledged":true}
Forms             : {}
Headers           : {[Content-Length, 21], [Content-Type, application/json; charset=UTF-8]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 21
```

3. List all indexes with the following command:

    Invoke-WebRequest –Method GET –Uri http://localhost:9200/_cat/indices?v | Select Content | Format-List

4. With output as follows:



```
PS C:\windows\system32> Invoke-WebRequest -Method GET -Uri http://localhost:9200/_cat/indices?v | Select Content | Format-List


Content : health status index uuid pri rep docs.count docs.deleted store.size pri.store.size
```

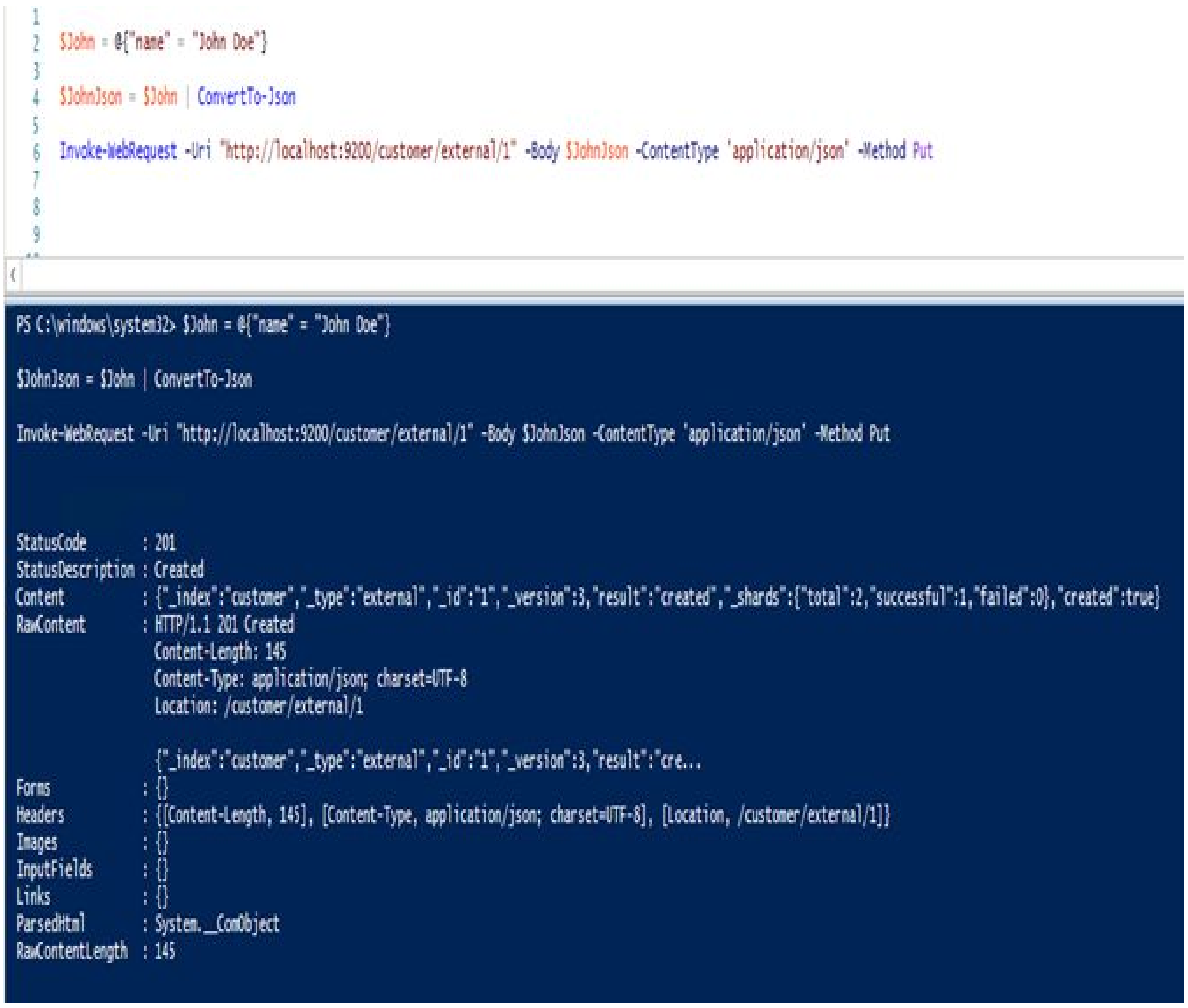
Modifying your Data:

https://www.elastic.co/guide/en/elasticsearch/reference/current/_modifying_your_data.html

Elasticsearch provides data manipulation and search capabilities in near real time. By Default, you can expect a one second delay (refresh interval) from the time you index/update/delete your data until the time that it appears in your search results. This is an important distinction from other platforms like SQL wherein data is immediately available after a transaction is completed.

Indexing/Replacing Documents:

1. Previously we saw how to index a document. Here is the command again:

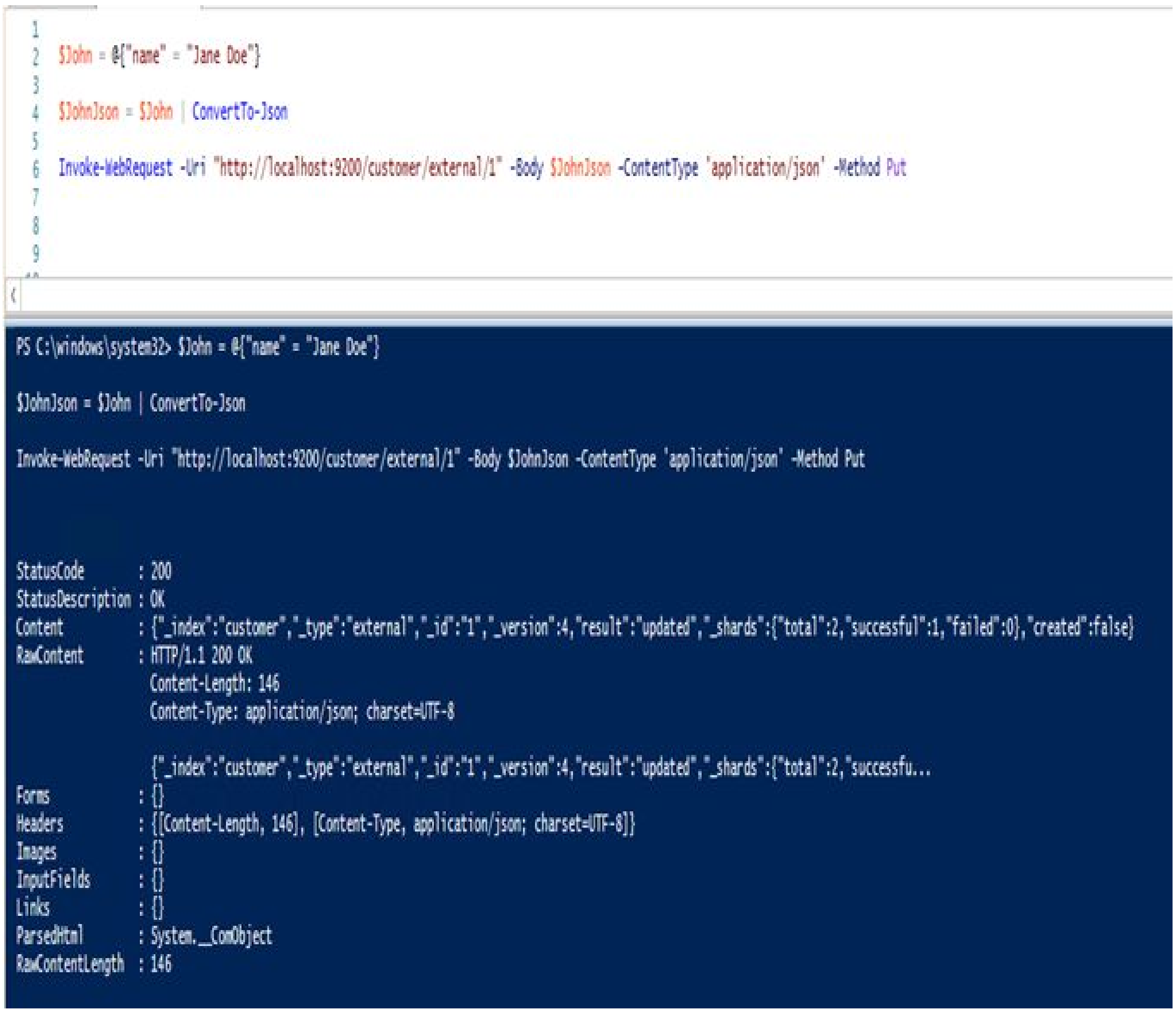


```

$John = @{"name" = "John Doe"}

$JohnJson = $John | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body $JohnJson -ContentType 'application/json' -Method Put



```

```
PS C:\windows\system32> $John = @{"name" = "John Doe"}

$JohnJson = $John | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body $JohnJson -ContentType 'application/json' -Method Put



StatusCode        : 201
StatusDescription : Created
Content           : {"_index":"customer","_type":"external","_id":"1","_version":3,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"created":true}
RawContent        : HTTP/1.1 201 Created
                    Content-Length: 145
                    Content-Type: application/json; charset=UTF-8
                    Location: /customer/external/1

                    {"_index":"customer","_type":"external","_id":"1","_version":3,"result":"cre...
Forms             : {}
Headers           : {[Content-Length, 145], [Content-Type, application/json; charset=UTF-8], [Location, /customer/external/1]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 145
```

The above will index the specified document into a customer index, external type and with ID of 1.

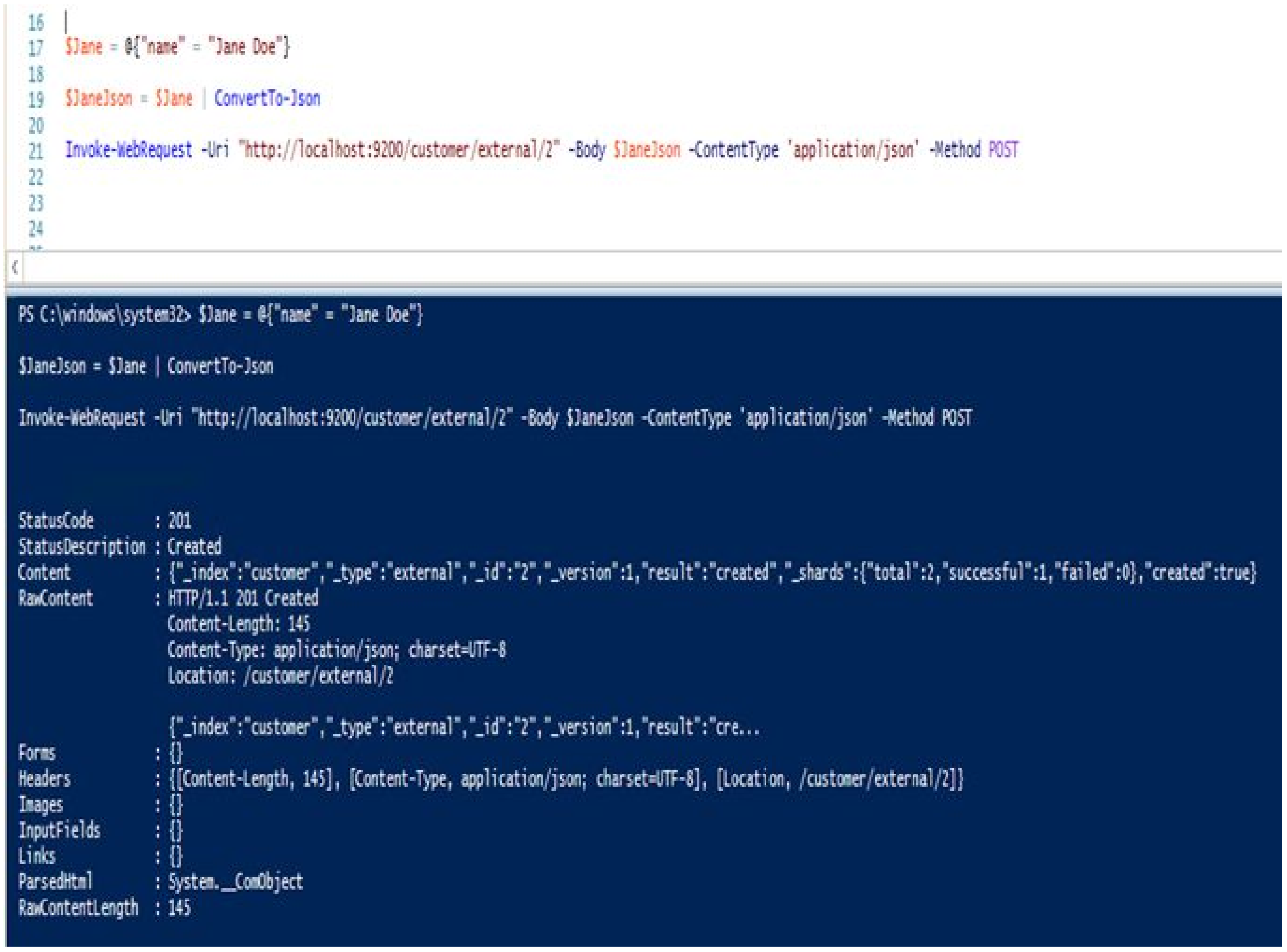
2. If the above command is executed again with a different (or same) document, Elasticsearch will replace (ie. reindex) a new document on top of the existing one with the ID of 1:



```
$John = @{"name" = "Jane Doe"}

$JohnJson = $John | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body $JohnJson -ContentType 'application/json' -Method Put
```

```
PS C:\windows\system32> $John = @{"name" = "Jane Doe"}

$JohnJson = $John | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body $JohnJson -ContentType 'application/json' -Method Put



StatusCode        : 200
StatusDescription : OK
Content           : {"_index":"customer","_type":"external","_id":"1","_version":4,"result":"updated","_shards":{"total":2,"successful":1,"failed":0},"created":false}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 146
                    Content-Type: application/json; charset=UTF-8

                    {"_index":"customer","_type":"external","_id":"1","_version":4,"result":"updated","_shards":{"total":2,"successfu...
Forms             : {}
Headers           : {[Content-Length, 146], [Content-Type, application/json; charset=UTF-8]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 146
```

3. The above changes the name of the document with the ID of 1 from “John Doe” to “Jane Doe”. If we used a different ID, a new document will be indexed and the existing document(s) already in the index remains untouched.



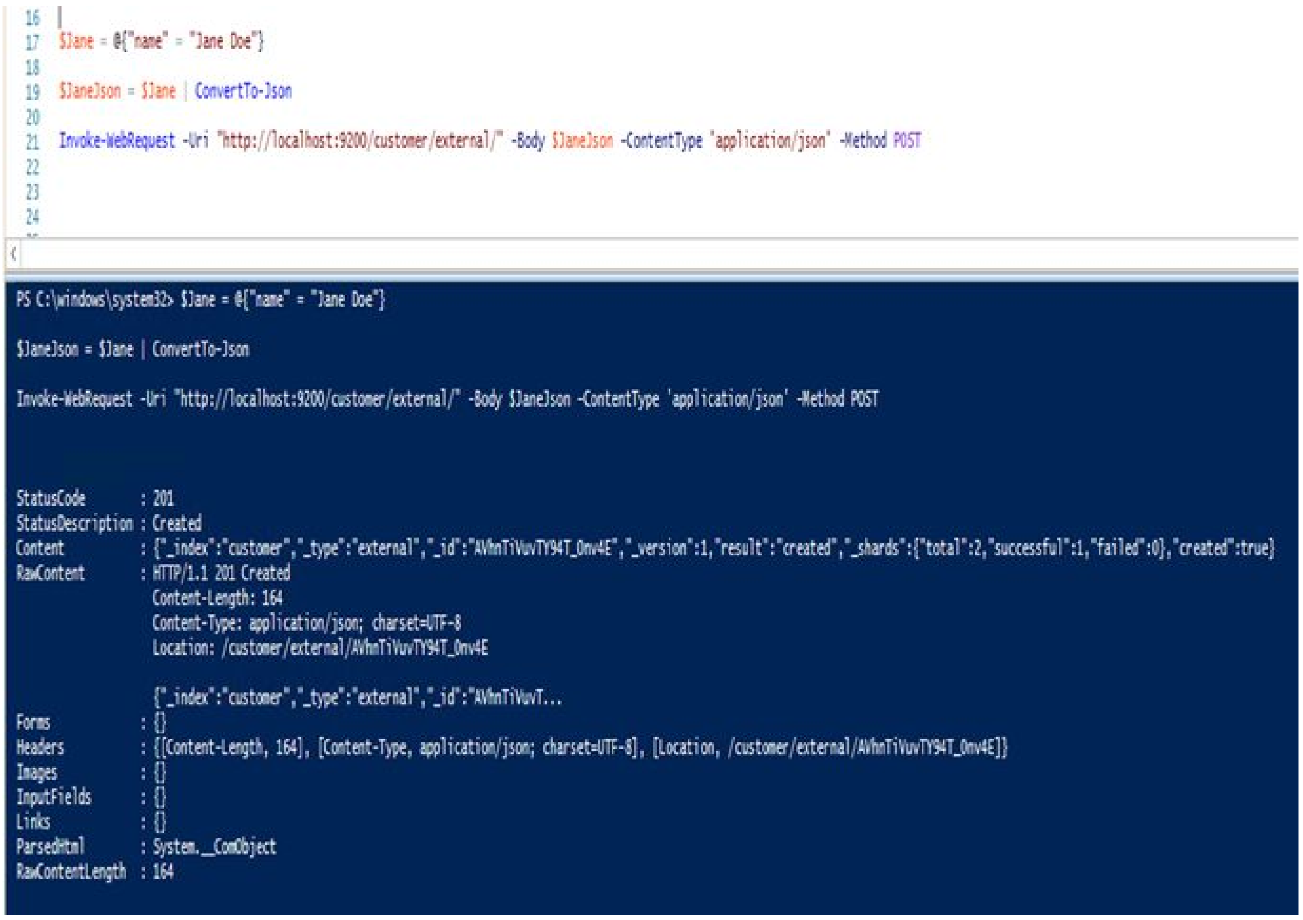
```
$Jane = @{"name" = "Jane Doe"}

$JaneJson = $Jane | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/2" -Body $JaneJson -ContentType 'application/json' -Method POST
```

```
PS C:\windows\system32> $Jane = @{"name" = "Jane Doe"}

$JaneJson = $Jane | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/2" -Body $JaneJson -ContentType 'application/json' -Method POST



StatusCode        : 201
StatusDescription : Created
Content           : {"_index":"customer","_type":"external","_id":"2","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"created":true}
RawContent        : HTTP/1.1 201 Created
                    Content-Length: 145
                    Content-Type: application/json; charset=UTF-8
                    Location: /customer/external/2

                    {"_index":"customer","_type":"external","_id":"2","_version":1,"result":"cre...
Forms             : {}
Headers           : {[Content-Length, 145], [Content-Type, application/json; charset=UTF-8], [Location, /customer/external/2]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 145
```

The above indexes a new document with an ID of 2.

4. When indexing, the ID part is optional. If not specified, Elasticsearch will generate a random ID and then use it to index the document. The actual ID Elasticsearch generates (or whatever we specified explicitly) is returned as part of the index API call.

5. This example shows how to index a document without an explicit ID:



```
$Jane = @{"name" = "Jane Doe"}

$JaneJson = $Jane | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/" -Body $JaneJson -ContentType 'application/json' -Method POST
```

```
PS C:\windows\system32> $Jane = @{"name" = "Jane Doe"}

$JaneJson = $Jane | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/" -Body $JaneJson -ContentType 'application/json' -Method POST



StatusCode        : 201
StatusDescription : Created
Content           : {"_index":"customer","_type":"external","_id":"AVhnTiVuvTY94T_Onv4E","_version":1,"result":"created","_shards":{"total":2,"successful":1,"failed":0},"created":true}
RawContent        : HTTP/1.1 201 Created
                    Content-Length: 164
                    Content-Type: application/json; charset=UTF-8
                    Location: /customer/external/AVhnTiVuvTY94T_Onv4E

                    {"_index":"customer","_type":"external","_id":"AVhnTiVuvT...
Forms             : {}
Headers           : {[Content-Length, 164], [Content-Type, application/json; charset=UTF-8], [Location, /customer/external/AVhnTiVuvTY94T_Onv4E]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 164
```

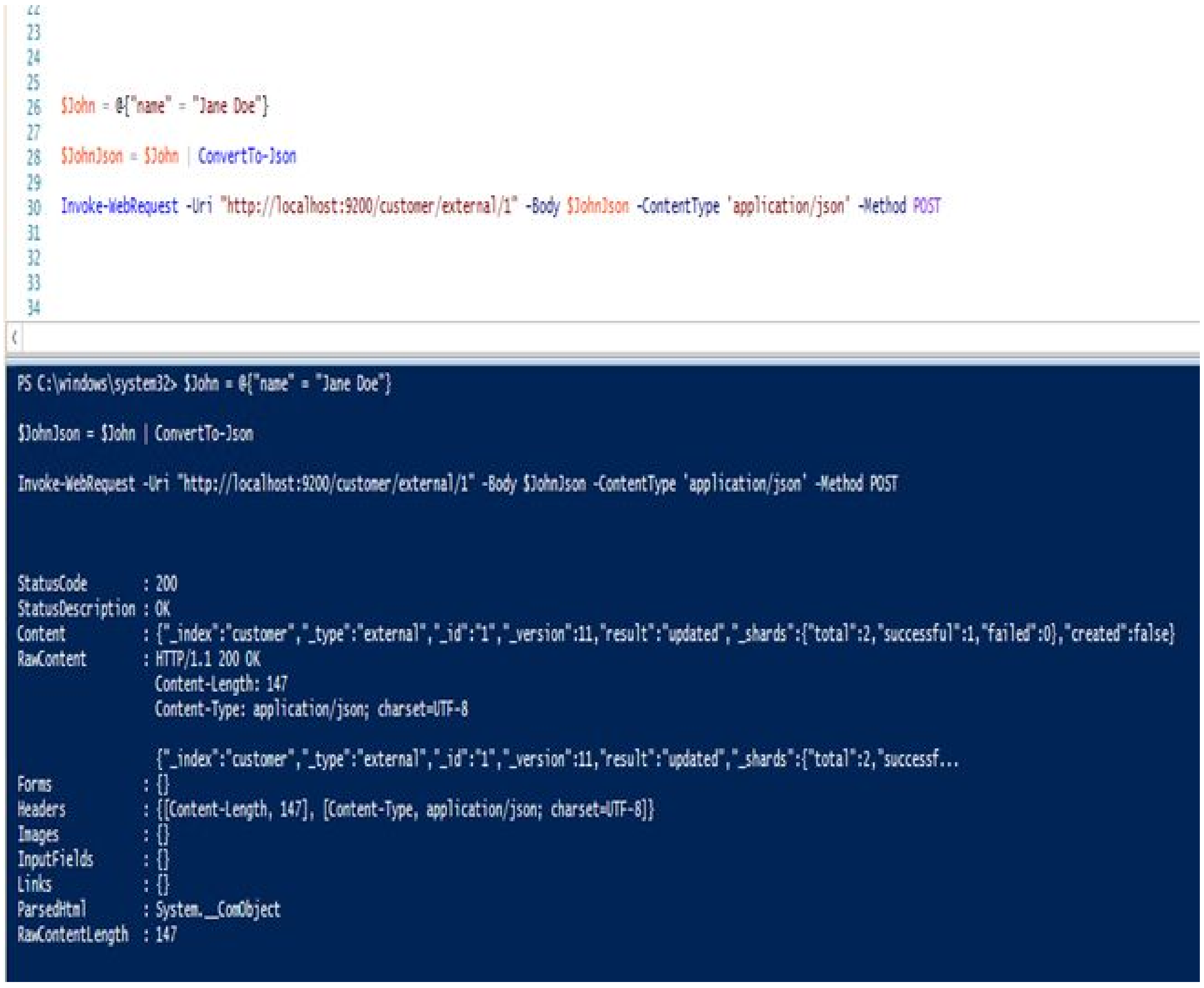
Note that in the above case, we are using the POST verb instead of PUT since we didn't specify an ID.

Updating Documents:

https://www.elastic.co/guide/en/elasticsearch/reference/current/_updating_documents.html

In addition to being able to index and replace documents, we can also update them. Note though that Elasticsearch does not actually do in-place updates under the hood. Whenever you do an update, Elasticsearch deletes the old document and then indexes a new document with the update applied to it in one shot.

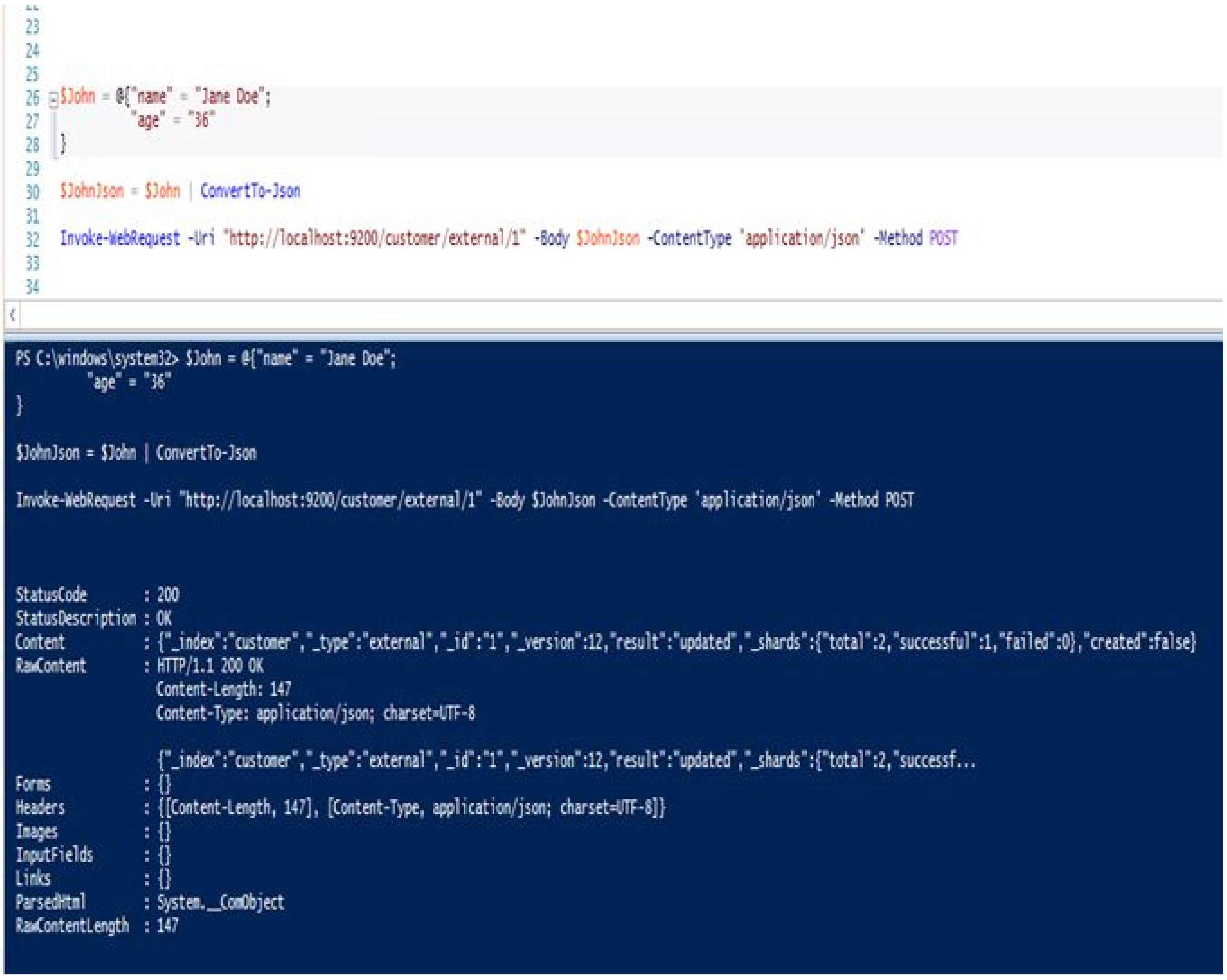
1. Update the previous document (ID of 1) by changing the name field to "Jane Doe":



```
$John = @{"name" = "Jane Doe"}

$JohnJson = $John | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body $JohnJson -ContentType 'application/json' -Method POST
```

```
PS C:\windows\system32> $John = @{"name" = "Jane Doe"}

$JohnJson = $John | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body $JohnJson -ContentType 'application/json' -Method POST



StatusCode        : 200
StatusDescription : OK
Content           : {"_index":"customer","_type":"external","_id":"1","_version":11,"result":"updated","_shards":{"total":2,"successful":1,"failed":0},"created":false}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 147
                    Content-Type: application/json; charset=UTF-8

                    {"_index":"customer","_type":"external","_id":"1","_version":11,"result":"updated","_shards":{"total":2,"successf...
Forms             : {}
Headers           : {[Content-Length, 147], [Content-Type, application/json; charset=UTF-8]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 147
```

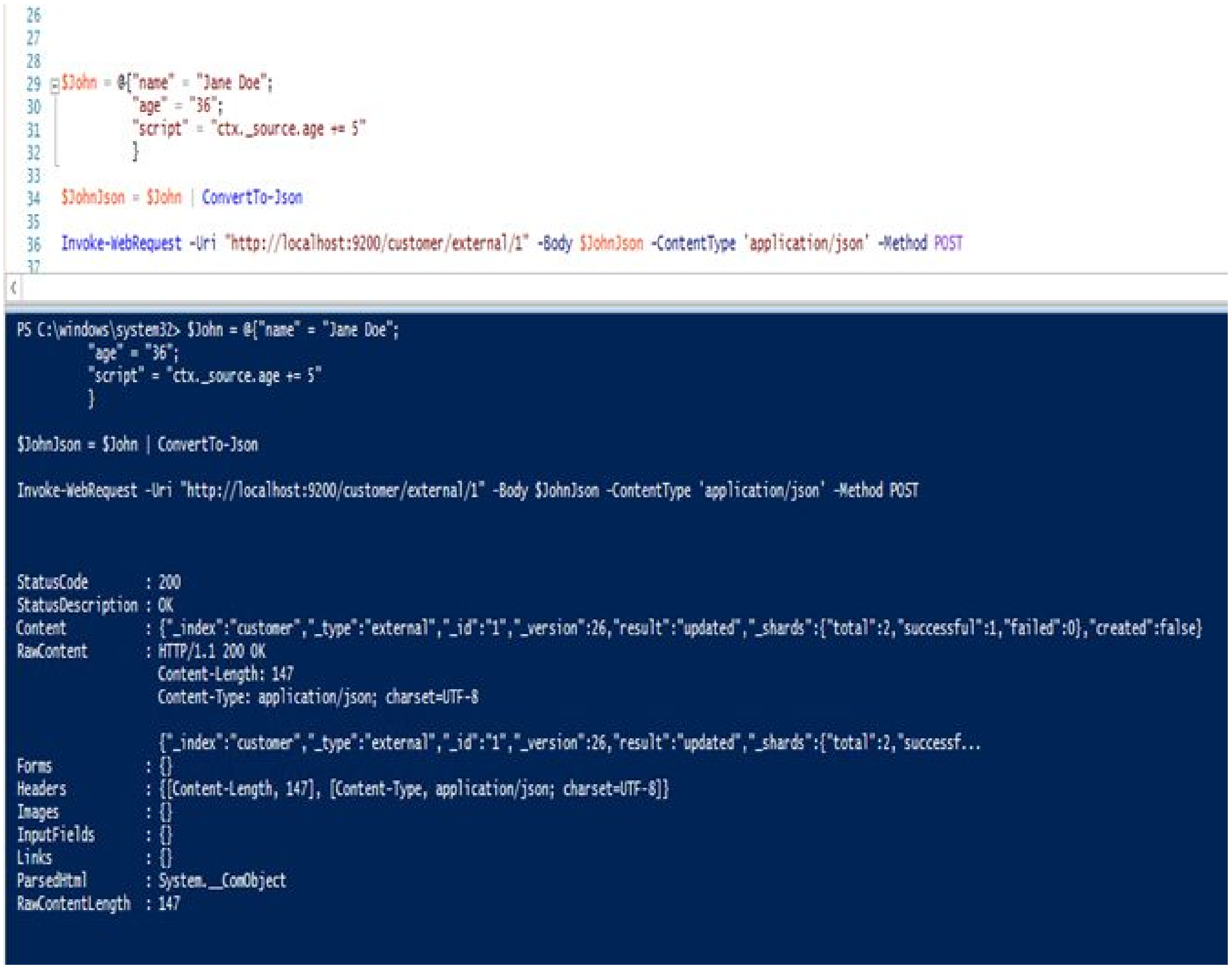
2. This example shows how to update our previous document (ID of 1) by changing the name field to "Jane Doe" and at the same time add an age field to it:



```
$John = @{"name" = "Jane Doe";
          "age" = "36"
}

$JohnJson = $John | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body $JohnJson -ContentType 'application/json' -Method POST
```

```
PS C:\windows\system32> $John = @{"name" = "Jane Doe";
          "age" = "36"
}

$JohnJson = $John | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body $JohnJson -ContentType 'application/json' -Method POST



StatusCode        : 200
StatusDescription : OK
Content           : {"_index":"customer","_type":"external","_id":"1","_version":12,"result":"updated","_shards":{"total":2,"successful":1,"failed":0},"created":false}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 147
                    Content-Type: application/json; charset=UTF-8

                    {"_index":"customer","_type":"external","_id":"1","_version":12,"result":"updated","_shards":{"total":2,"successf...
Forms             : {}
Headers           : {[Content-Length, 147], [Content-Type, application/json; charset=UTF-8]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 147
```

3. Updates can also be performed by using simple scripts. This example uses a script to increment the age by 5:



```
$John = @{"name" = "Jane Doe";
          "age" = "36";
          "script" = "ctx._source.age += 5"
          }

$JohnJson = $John | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body $JohnJson -ContentType 'application/json' -Method POST
```

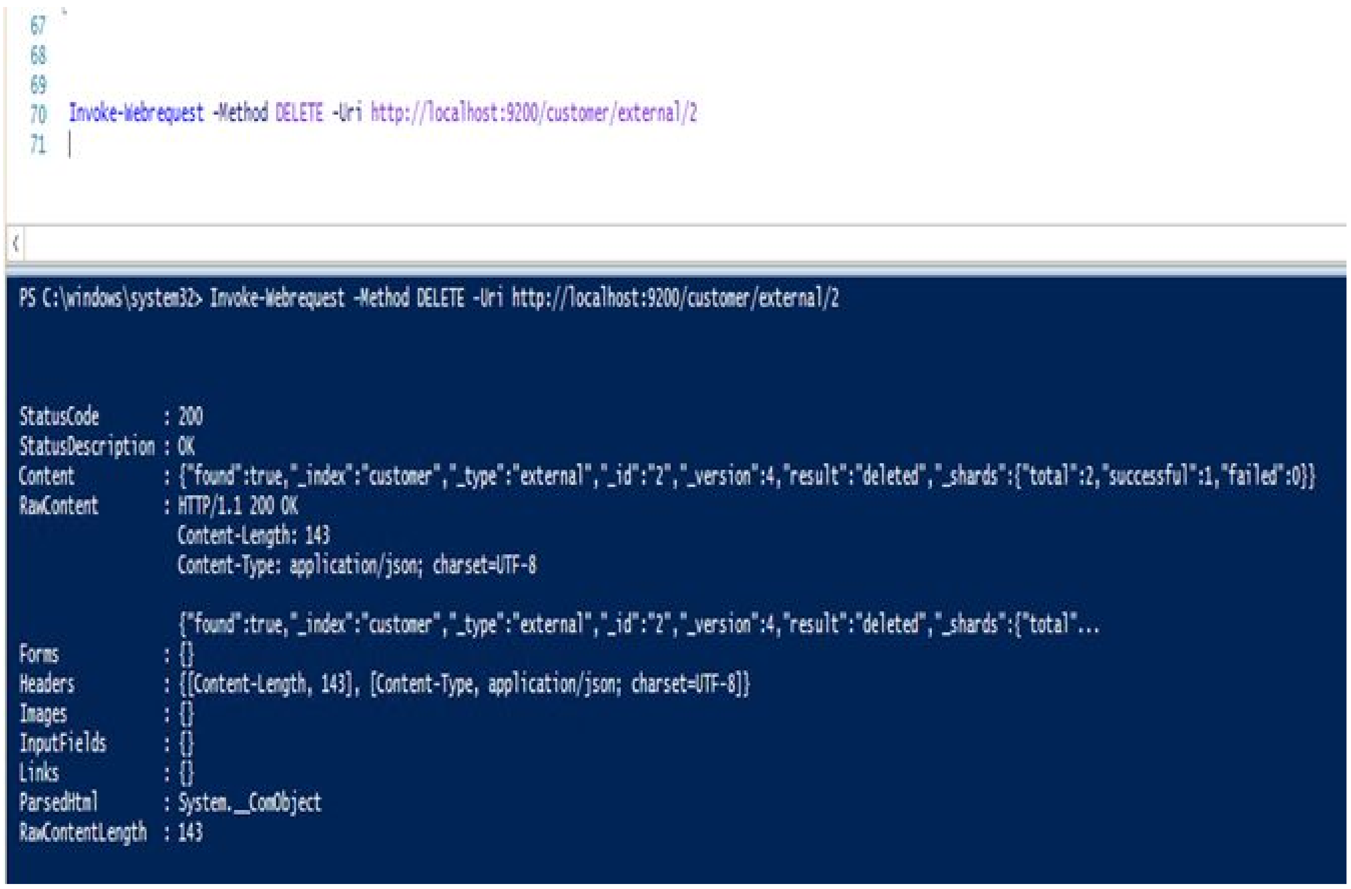
```
PS C:\windows\system32> $John = @{"name" = "Jane Doe";
          "age" = "36";
          "script" = "ctx._source.age += 5"
          }

$JohnJson = $John | ConvertTo-Json

Invoke-WebRequest -Uri "http://localhost:9200/customer/external/1" -Body $JohnJson -ContentType 'application/json' -Method POST



StatusCode        : 200
StatusDescription : OK
Content           : {"_index":"customer","_type":"external","_id":"1","_version":26,"result":"updated","_shards":{"total":2,"successful":1,"failed":0},"created":false}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 147
                    Content-Type: application/json; charset=UTF-8

                    {"_index":"customer","_type":"external","_id":"1","_version":26,"result":"updated","_shards":{"total":2,"successf...
Forms             : {}
Headers           : {[Content-Length, 147], [Content-Type, application/json; charset=UTF-8]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 147
```

In the above, ctx._source refers to the current source document that is about to be updated. Note: As of writing, updates can only be performed on single document at a time.

Deleting Documents:

https://www.elastic.co/guide/en/elasticsearch/reference/current/_deleting_documents.html

1. Deleting documents is fairly straight forward. This example shows how to delete our previous customer with ID of 2:



```
Invoke-Webrequest -Method DELETE -Uri http://localhost:9200/customer/external/2
```

```
PS C:\windows\system32> Invoke-Webrequest -Method DELETE -Uri http://localhost:9200/customer/external/2


StatusCode        : 200
StatusDescription : OK
Content           : {"found":true,"_index":"customer","_type":"external","_id":"2","_version":4,"result":"deleted","_shards":{"total":2,"successful":1,"failed":0}}
RawContent        : HTTP/1.1 200 OK
                    Content-Length: 143
                    Content-Type: application/json; charset=UTF-8

                    {"found":true,"_index":"customer","_type":"external","_id":"2","_version":4,"result":"deleted","_shards":{"total"...
Forms             : {}
Headers           : {[Content-Length, 143], [Content-Type, application/json; charset=UTF-8]}
Images            : {}
InputFields       : {}
Links             : {}
ParsedHtml        : System.__ComObject
RawContentLength  : 143
```

See the Delete By Query API to delete all documents matching a specific query. It is much more efficient to delete a whole index instead of just deleting all documents with the Delete by Query API.

Batch Processing:

https://www.elastic.co/guide/en/elasticsearch/reference/current/_batch_processing.html

In addition to being able to index, update and delete individual documents, Elasticsearch also provides the ability to perform any of the above operations in batches using the _bulk API. This functionality is important in that it provides a very efficient mechanism to do multiple operations as fast as possible with as little network round trips as possible.

NOTE: I'm not able to find any easy Powershell equivalent method to do bulk processing as yet. Once I have managed to resolve this, I will update the document accordingly.

Exploring Your Data:

Sample Dataset:

We will now work with a more realistic dataset, through a sample of fictitious JSON documents of customer bank account information. Each document has the following schema:

```json
{
    "account_number": 0,
    "balance": 16623,
    "firstname": "Bradshaw",
    "lastname": "Mckenzie",
    "age": 29,
    "gender": "F",
    "address": "244 Columbus Place",
    "employer": "Euron",
    "email": "bradshawmckenzie@euron.com",
    "city": "Hobucken",
    "state": "CO"
}
```

Loading the Sample Dataset:

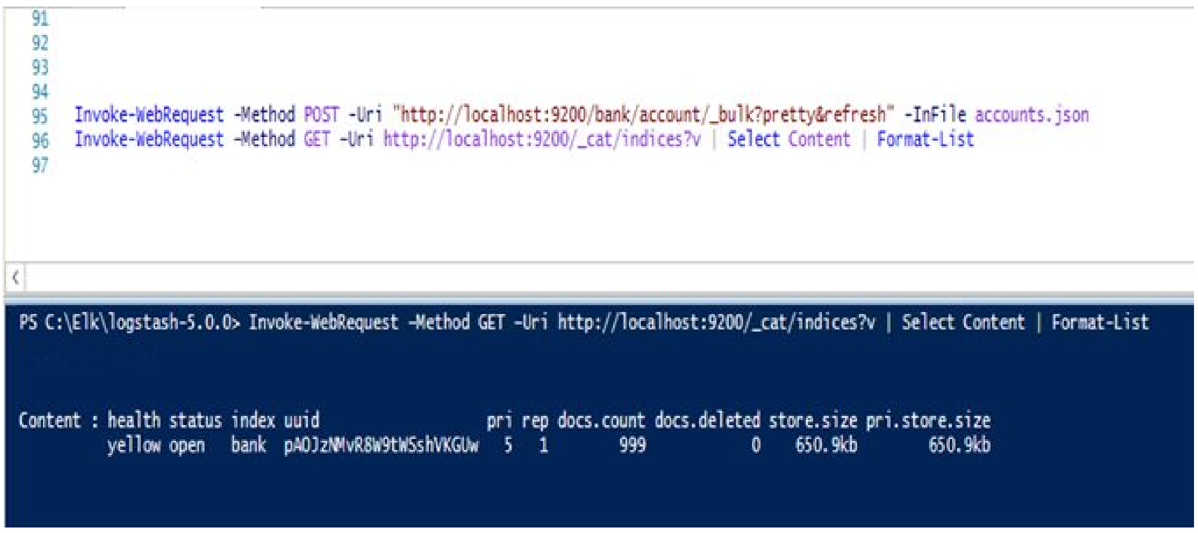
1. Download the sample dataset (accounts.json) here. Extract the file to your current directory and load the file into the cluster with the following command:

```
Invoke-WebRequest
-Method POST
-Uri "http://localhost:9200/bank/account/_bulk?pretty&refresh"
-InFile accounts.json
```

2. List and review the imported indices with the following command:

```
Invoke-WebRequest –Method GET –Uri http://localhost:9200/_cat/indices?v | Select Content | Format-List
```

3. The output will look similar to the following:



```
Invoke-WebRequest -Method POST -Uri "http://localhost:9200/bank/account/_bulk?pretty&refresh" -InFile accounts.json
Invoke-WebRequest -Method GET -Uri http://localhost:9200/_cat/indices?v | Select Content | Format-List
```

```
PS C:\Elk\logstash-5.0.0> Invoke-WebRequest -Method GET -Uri http://localhost:9200/_cat/indices?v | Select Content | Format-List


Content : health status index uuid                   pri rep docs.count docs.deleted store.size pri.store.size
          yellow open   bank  pAOJzNMvR8W9tWSshVKGUw   5   1        999            0    650.9kb        650.9kb
```

Which means that we successfully bulk indexed 1000 documents into the bank index.

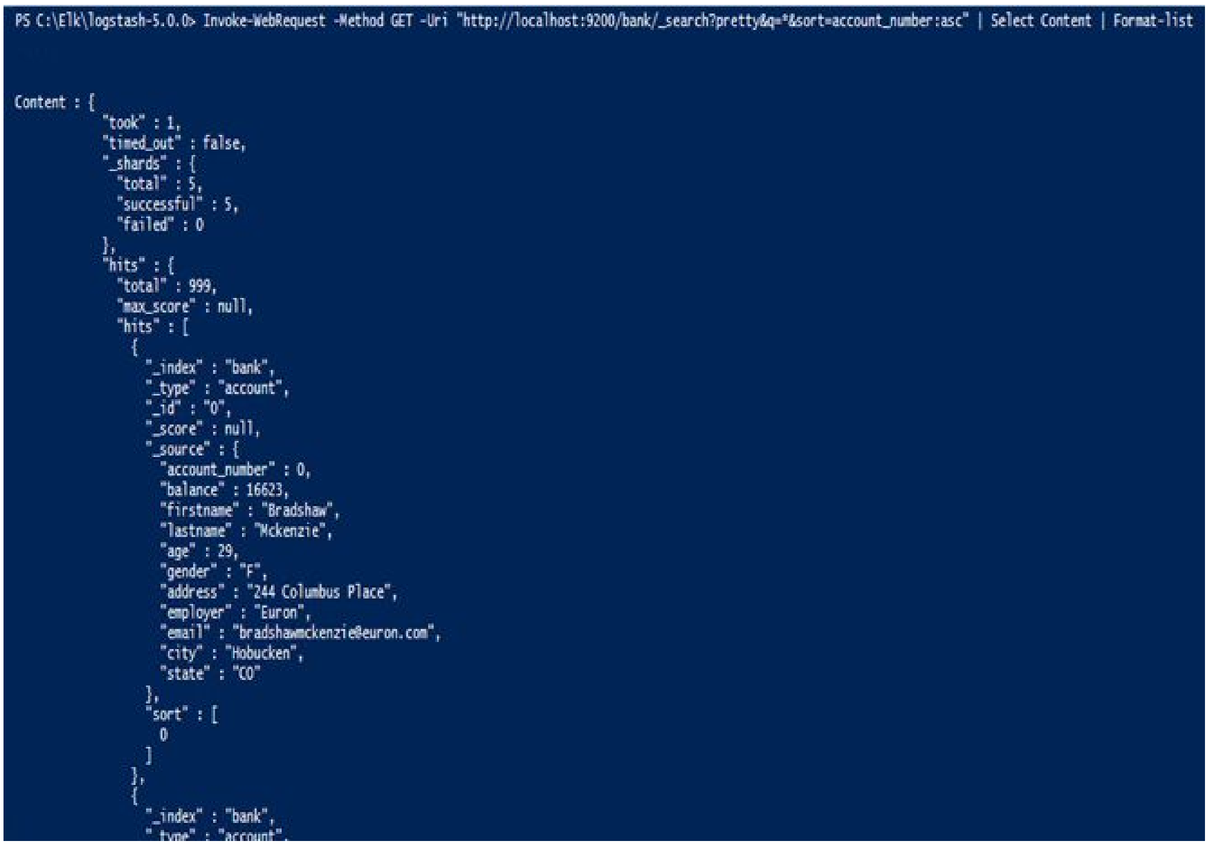
The Search API:

There are two basic ways to run searches: one is by sending search parameters through the REST request URI, and the other by sending them through the REST request body. The request body method allows you to be more expensive and also to define your searches in a more readable JSON format. ~~We will do one example of the request URI method, but the remainder of the tutorial will exclusively use the request body method.~~

1. The REST API for search is accessible from the _search endpoint. This command returns all documents in the bank index:

```
Invoke-WebRequest -Method GET -Uri "http://localhost:9200/bank/_search?pretty&q=*&sort=account_number:asc" |
Select Content | Format-list
```

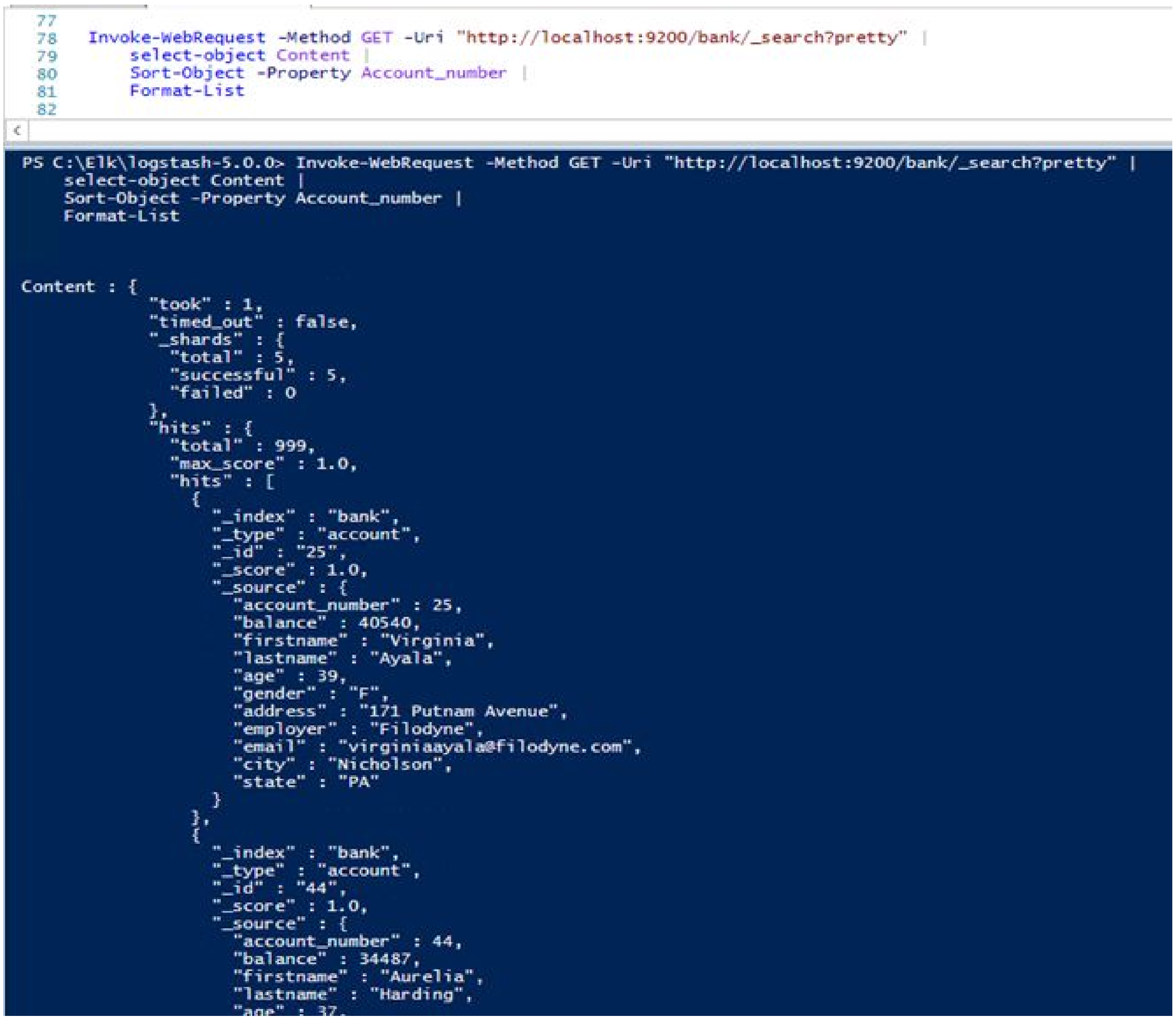
2. With the above command, we search (_search endpoint) in the bank index, and the q=* parameter instructs Elasticsearch to match all documents in the index. The pretty parameter tells Elasticsearch to return pretty-printed JSON results:



```
PS C:\Elk\logstash-5.0.0> Invoke-WebRequest -Method GET -Uri "http://localhost:9200/bank/_search?pretty&q=*&sort=account_number:asc" | Select Content | Format-list


Content : {
            "took" : 1,
            "timed_out" : false,
            "_shards" : {
              "total" : 5,
              "successful" : 5,
              "failed" : 0
            },
            "hits" : {
              "total" : 999,
              "max_score" : null,
              "hits" : [
                {
                  "_index" : "bank",
                  "_type" : "account",
                  "_id" : "0",
                  "_score" : null,
                  "_source" : {
                    "account_number" : 0,
                    "balance" : 16623,
                    "firstname" : "Bradshaw",
                    "lastname" : "Mckenzie",
                    "age" : 29,
                    "gender" : "F",
                    "address" : "244 Columbus Place",
                    "employer" : "Euron",
                    "email" : "bradshawmckenzie@euron.com",
                    "city" : "Hobucken",
                    "state" : "CO"
                  },
                  "sort" : [
                    0
                  ]
                },
                {
                  "_index" : "bank",
                  "_type" : "account",
```

3. In the response, we see the following parts:
   - *took* – the time in milliseconds for Elasticsearch to execute the search.
   - *timed_out* – indicates if search timed out or not.
   - *_shards* – how many shards were searched, as well as a count of the successful/failed search shards.
   - *hits* – search results.
   - *hits.total* – total number of documents matching the search criteria.
   - *hits.hits* – actual array of search results (defaults to first 10 documents)
   - *sort* – sort key for results (missing if sorting by score)
   - *_score* and *max_score* – ignore these for now.

4. Here is the same exact search using the alternative request body method:



```
Invoke-WebRequest -Method GET -Uri "http://localhost:9200/bank/_search?pretty" |
    select-object Content |
    Sort-Object -Property Account_number |
    Format-List
```

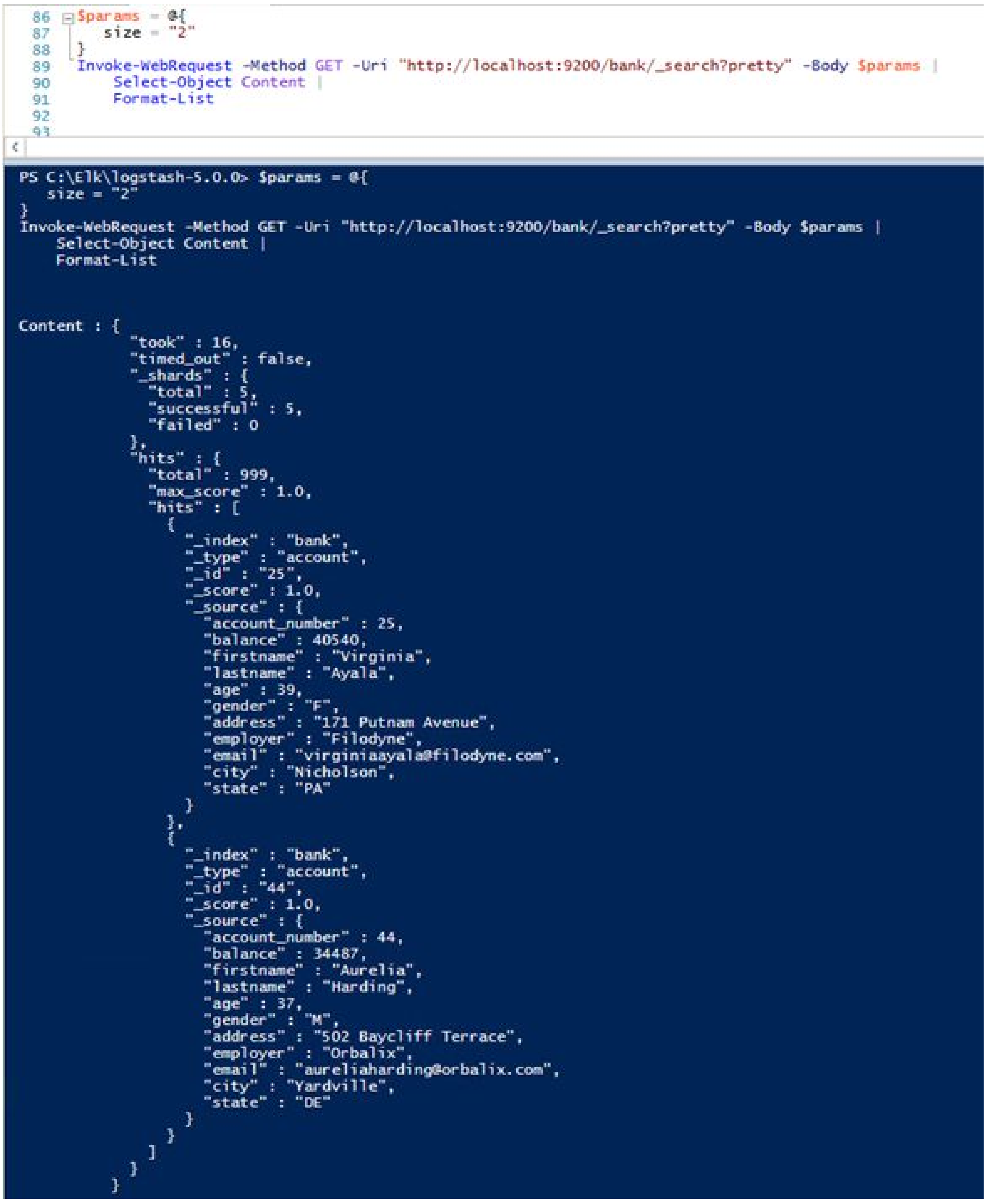
```
PS C:\Elk\logstash-5.0.0> Invoke-WebRequest -Method GET -Uri "http://localhost:9200/bank/_search?pretty" |
    select-object Content |
    Sort-Object -Property Account_number |
    Format-List


Content : {
            "took" : 1,
            "timed_out" : false,
            "_shards" : {
              "total" : 5,
              "successful" : 5,
              "failed" : 0
            },
            "hits" : {
              "total" : 999,
              "max_score" : 1.0,
              "hits" : [
                {
                  "_index" : "bank",
                  "_type" : "account",
                  "_id" : "25",
                  "_score" : 1.0,
                  "_source" : {
                    "account_number" : 25,
                    "balance" : 40540,
                    "firstname" : "Virginia",
                    "lastname" : "Ayala",
                    "age" : 39,
                    "gender" : "F",
                    "address" : "171 Putnam Avenue",
                    "employer" : "Filodyne",
                    "email" : "virginiaayala@filodyne.com",
                    "city" : "Nicholson",
                    "state" : "PA"
                  }
                },
                {
                  "_index" : "bank",
                  "_type" : "account",
                  "_id" : "44",
                  "_score" : 1.0,
                  "_source" : {
                    "account_number" : 44,
                    "balance" : 34487,
                    "firstname" : "Aurelia",
                    "lastname" : "Harding",
                    "age" : 37,
```

The difference here is that instead of passing the q=* in the URI, we pipe the results of the index search into a Select cmdlet, then pipe it to a Sort cmdlet, and finally to a format-list. Once you get results back, Elasticsearch is completely done with the request and does not maintain any kind of server-side resources or open cursors into the results.

Introducing the Query Language:

Elasticsearch provides a JSON-style domain-specific language that you can use to execute queries. This is referred to as Query DSL, and the language is quite comprehensive and can be intimidating at first glance but the best way to actually learn it is to start with a few basic examples.

1. We can also create an array for the Body in the above command, where we specify various parameters. In the example below we pass the size parameter:

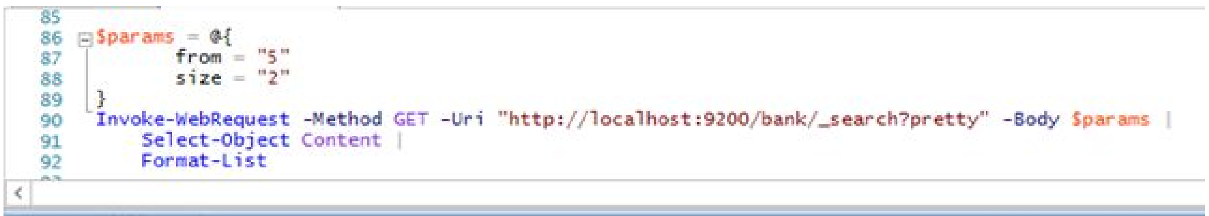


```
$params = @{
    size = "2"
}
Invoke-WebRequest -Method GET -Uri "http://localhost:9200/bank/_search?pretty" -Body $params |
    Select-Object Content |
    Format-List
```

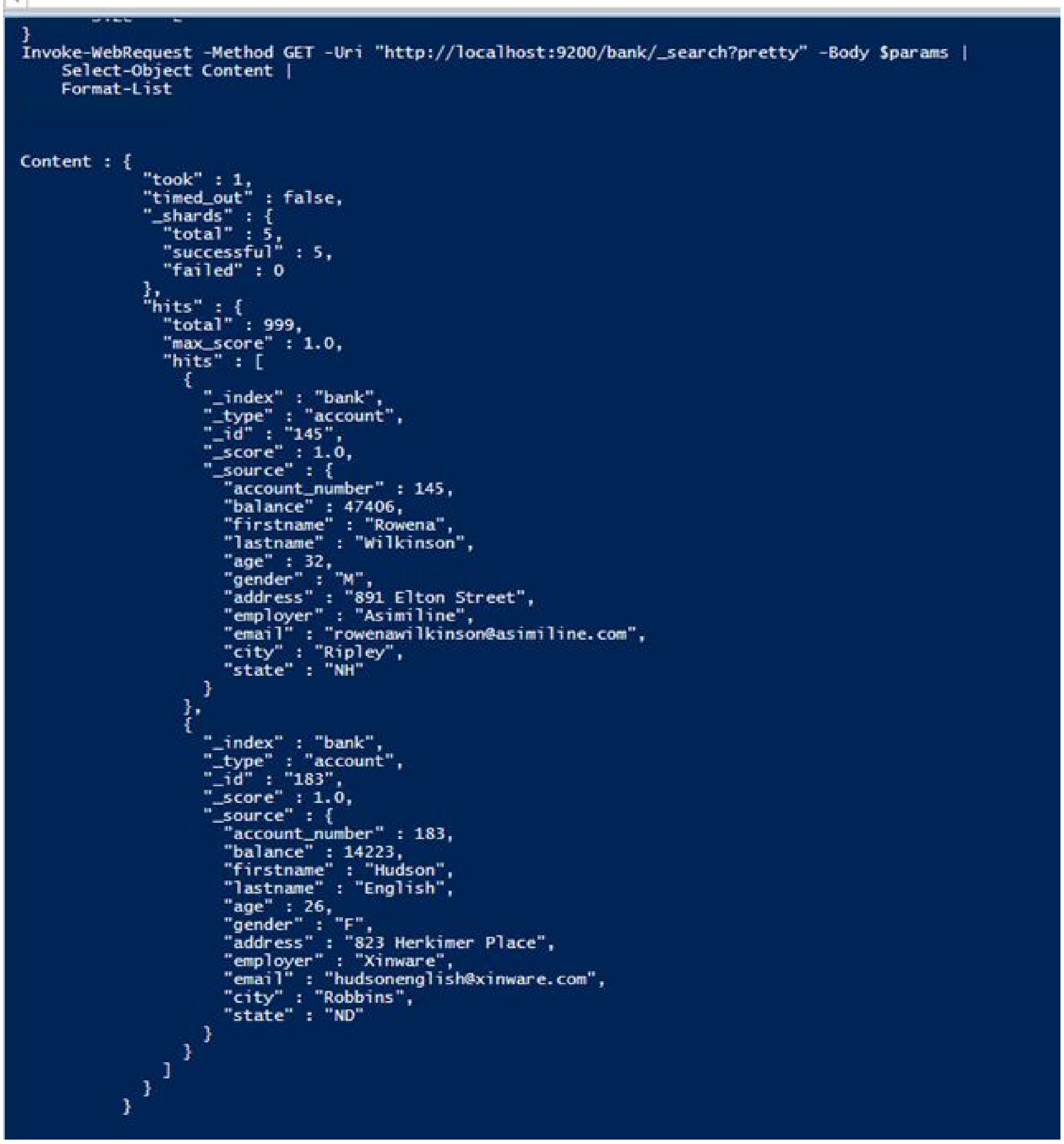
```
PS C:\Elk\logstash-5.0.0> $params = @{
   size = "2"
}
Invoke-WebRequest -Method GET -Uri "http://localhost:9200/bank/_search?pretty" -Body $params |
    Select-Object Content |
    Format-List


Content : {
            "took" : 16,
            "timed_out" : false,
            "_shards" : {
              "total" : 5,
              "successful" : 5,
              "failed" : 0
            },
            "hits" : {
              "total" : 999,
              "max_score" : 1.0,
              "hits" : [
                {
                  "_index" : "bank",
                  "_type" : "account",
                  "_id" : "25",
                  "_score" : 1.0,
                  "_source" : {
                    "account_number" : 25,
                    "balance" : 40540,
                    "firstname" : "Virginia",
                    "lastname" : "Ayala",
                    "age" : 39,
                    "gender" : "F",
                    "address" : "171 Putnam Avenue",
                    "employer" : "Filodyne",
                    "email" : "virginiaayala@filodyne.com",
                    "city" : "Nicholson",
                    "state" : "PA"
                  }
                },
                {
                  "_index" : "bank",
                  "_type" : "account",
                  "_id" : "44",
                  "_score" : 1.0,
                  "_source" : {
                    "account_number" : 44,
                    "balance" : 34487,
                    "firstname" : "Aurelia",
                    "lastname" : "Harding",
                    "age" : 37,
                    "gender" : "M",
                    "address" : "502 Baycliff Terrace",
                    "employer" : "Orbalix",
                    "email" : "aureliaharding@orbalix.com",
                    "city" : "Yardville",
                    "state" : "DE"
                  }
                }
              ]
            }
          }
```

This will only list the first 2 results. Note, that if you do not specify a size parameter, it defaults to 10.

2. In the following example we return documents 5 through to 7:



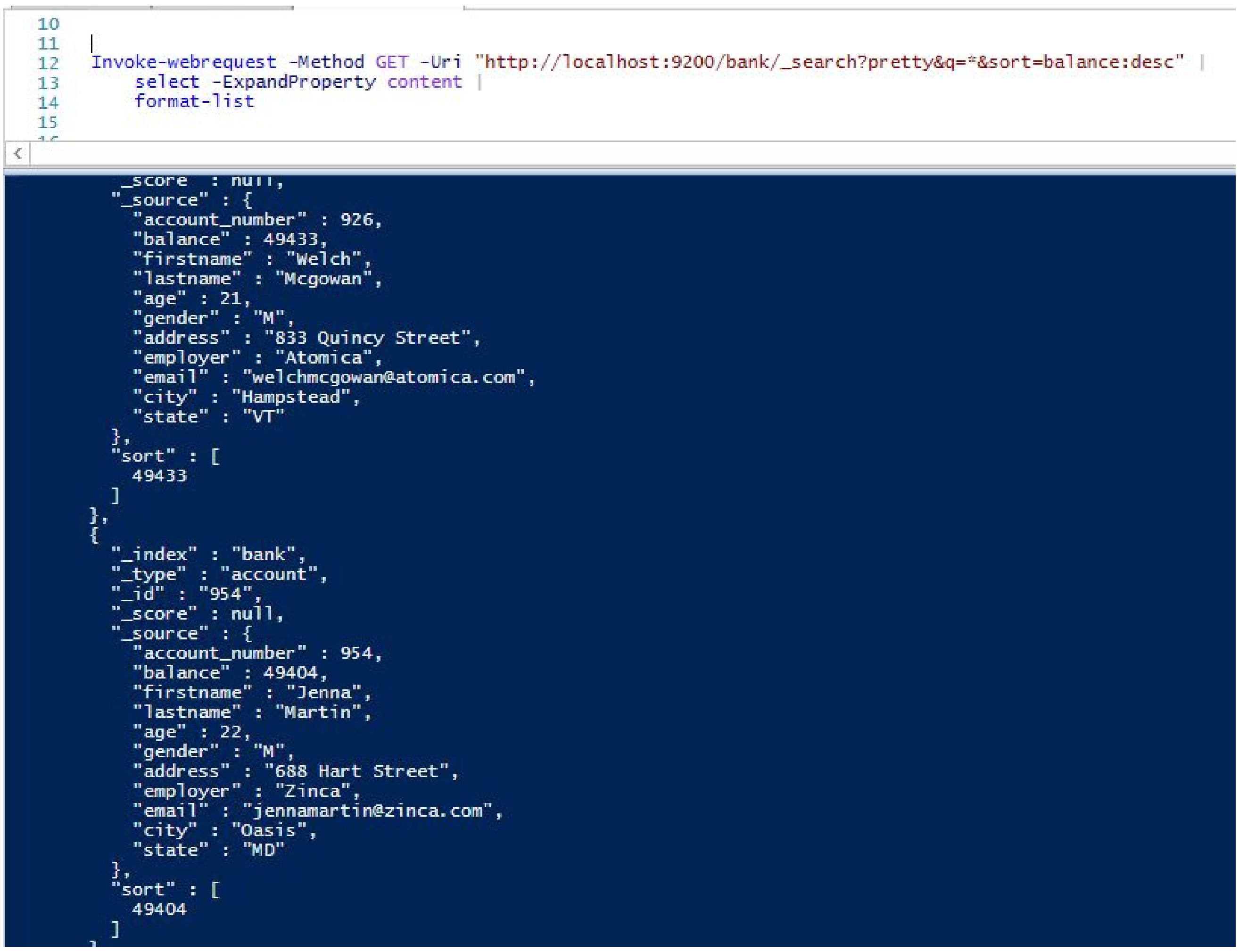
```
$params = @{
        from = "5"
        size = "2"
}
Invoke-WebRequest -Method GET -Uri "http://localhost:9200/bank/_search?pretty" -Body $params |
    Select-Object Content |
    Format-List
```



```
}
Invoke-WebRequest -Method GET -Uri "http://localhost:9200/bank/_search?pretty" -Body $params |
    Select-Object Content |
    Format-List


Content : {
            "took" : 1,
            "timed_out" : false,
            "_shards" : {
              "total" : 5,
              "successful" : 5,
              "failed" : 0
            },
            "hits" : {
              "total" : 999,
              "max_score" : 1.0,
              "hits" : [
                {
                  "_index" : "bank",
                  "_type" : "account",
                  "_id" : "145",
                  "_score" : 1.0,
                  "_source" : {
                    "account_number" : 145,
                    "balance" : 47406,
                    "firstname" : "Rowena",
                    "lastname" : "Wilkinson",
                    "age" : 32,
                    "gender" : "M",
                    "address" : "891 Elton Street",
                    "employer" : "Asimiline",
                    "email" : "rowenawilkinson@asimiline.com",
                    "city" : "Ripley",
                    "state" : "NH"
                  }
                },
                {
                  "_index" : "bank",
                  "_type" : "account",
                  "_id" : "183",
                  "_score" : 1.0,
                  "_source" : {
                    "account_number" : 183,
                    "balance" : 14223,
                    "firstname" : "Hudson",
                    "lastname" : "English",
                    "age" : 26,
                    "gender" : "F",
                    "address" : "823 Herkimer Place",
                    "employer" : "Xinware",
                    "email" : "hudsonenglish@xinware.com",
                    "city" : "Robbins",
                    "state" : "ND"
                  }
                }
              ]
            }
          }
```

The from parameter (0-based) specifies which document index to start from and the size parameter specifies how many documents to return starting at the from parameter. This feature is useful when implementing paging of search results. Note that if from is not specified, it defaults to 0.

3. This example does a match and sorts the results by account balance in descending order and returns the top 10 (default size) documents:



```
Invoke-webrequest -Method GET -Uri "http://localhost:9200/bank/_search?pretty&q=*&sort=balance:desc" |
    select -ExpandProperty content |
    format-list
```

```
          "_score" : null,
          "_source" : {
            "account_number" : 926,
            "balance" : 49433,
            "firstname" : "Welch",
            "lastname" : "Mcgowan",
            "age" : 21,
            "gender" : "M",
            "address" : "833 Quincy Street",
            "employer" : "Atomica",
            "email" : "welchmcgowan@atomica.com",
            "city" : "Hampstead",
            "state" : "VT"
          },
          "sort" : [
            49433
          ]
        },
        {
          "_index" : "bank",
          "_type" : "account",
          "_id" : "954",
          "_score" : null,
          "_source" : {
            "account_number" : 954,
            "balance" : 49404,
            "firstname" : "Jenna",
            "lastname" : "Martin",
            "age" : 22,
            "gender" : "M",
            "address" : "688 Hart Street",
            "employer" : "Zinca",
            "email" : "jennamartin@zinca.com",
            "city" : "Oasis",
            "state" : "MD"
          },
          "sort" : [
            49404
          ]
```

Executing Searches:

Let's dig some more into the Query DSL. By default, the full JSON document is returned as part of all searches. This is referred to as the source (_source field in the search hits). If we don't want the entire source document returned, we have the ability to request only a few fields from within the source to be returned.

1. Below shows how to return two fields, account_number and balance (inside _source), from the search:

Other Alternatives:

cURL:

Slightly out of scope for this document, you could install cURL for Windows by following these steps (for the Die-Hard *nix users):

1. Download the appropriate CAB file for your system architecture from the following link: https://curl.haxx.se/download.html.
2. Extract the file into the c:\windows\system32 folder.
3. Open a Command prompt, and navigate to c:\windows\system32, and issue the appropriate **cURL** commands.

Sense:

> *https://www.elastic.co/guide/en/sense/current/introduction.html*
> *Sense is a handy console for interacting with the REST API of Elasticsearch. As you can see below, Sense is composed of two main panes. The left pane, named the editor, is where you type the requests you will submit to Elasticsearch. The responses from Elasticsearch are shown on the right hand panel. The address of your Elasticsearch server should be entered in the text box on the top of screen (and defaults to `localhost:9200`).*

In this document, we are focusing on Elasticsearch, as well as creating and manipulating data via the command-line. In some cases, a 'nice-to-have' utility would be Sense (a Kibana4 application). In Elasticsearch v5/Kibana v5 Sense is considered legacy, and is now referred to as 'Console'.

We don't have Kibana installed. However, we could make use of a number of useful plugins for the Chrome browser:

Sense
Advanced REST client
POSTMAN

Powershell:

JSON to PowerShell Conversion Notes:

: becomes **=**
all ending commas go away
newlines denote new properties
@ before all new objects (e.g. **{}**)
[] becomes **@()**
@() is PowerShell for array
" becomes **""**
PowerShell escaping is double-doublequotes

DO NOT FORGET THE @ BEFORE {. If you do, it will sits there forever as it tries to serialize nothing into nothing. After a few minutes, you'll get hundreds of thousands of JSON entries. Seriously. I tries to serialize every aspect of every .NET property forever. This is why the -Depth defaults to 2.