```
# install elasticsearch
curl -L -O https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-7.6.1-linux-x86_64.tar.gz
tar -xvf elasticsearch-7.6.1-linux-x86_64.tar.gz

# run first instance (node)
cd elasticsearch-7.6.1/bin
./elasticsearch

# run second and third nodes (in order to test cluster)
./elasticsearch -Epath.data=data2 -Epath.logs=log2
./elasticsearch -Epath.data=data3 -Epath.logs=log3


# health check
curl -X GET "localhost:9200/_cat/health?v&pretty"

########################################################################
# Index some documents
curl -X PUT "localhost:9200/customer/_doc/1?pretty" -H 'Content-Type: application/json' -d'
{
  "name": "John Doe"
}
'

# retrieve document
curl -X GET "localhost:9200/customer/_doc/1?pretty"

# list indices
curl "localhost:9200/_cat/indices?v"

# index documents in bulk
# accounts.json is located in ~/Desktop/Books/elasticsearch/elastic_search_official_page/ folder
curl -H "Content-Type: application/json" -XPOST "localhost:9200/bank/_bulk?pretty&refresh"
--data-binary "@accounts.json"
curl "localhost:9200/_cat/indices?v"

########################################################################
Start searching

# The following request retrieves all documents in the bank index sorted by account number
# By default, the hits section of the response includes the first 10 documents that match the search
criteria
curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": { "match_all": {} },
  "sort": [
    { "account_number": "asc" }
  ]
}
'
```

```
# Pagination example
# Hits from 10 through 19
curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": { "match_all": {} },
  "sort": [
    { "account_number": "asc" }
  ],
  "from": 10,
  "size": 10
}
'

# To search for specific terms within a field, you can use a match query. For example,
# the following request searches the address field to find customers whose addresses contain mill or lane

curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": { "match": { "address": "mill lane" } }
}
'


# To perform a phrase search rather than matching individual terms, you use match_phrase instead of match.
# For example, the following request only matches addresses that contain the phrase mill lane

curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": { "match_phrase": { "address": "mill lane" } }
}
'


# To construct more complex queries, you can use a bool query to combine multiple query criteria.
# You can designate criteria as required (must match), desirable (should match),
# or undesirable (must not match).

# For example, the following request searches the bank index for accounts that belong to customers
# who are 40 years old, but excludes anyone who lives in Idaho (ID):

curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": {
    "bool": {
      "must": [
        { "match": { "age": "40" } }
      ],
      "must_not": [
        { "match": { "state": "ID" } }
      ]
```

```
    }
  }
}
'

# Must be 40 years old and must live in Idaho (ID) state
curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": {
    "bool": {
      "must": [
        { "match": { "age": "40" } },
        { "match": { "state": "ID" }}
      ]
    }
  }
}
'

# example to should
curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": {
     "bool": {
       "must": [
         { "match": {"age": "40" }}
       ],
       "should": [
         { "match": {"state": "ID" }}
       ]
     }
  }
}
'



# The following request uses a range filter to limit the results to accounts
# with a balance between $20,000 and $30,000 (inclusive).

curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "query": {
    "bool": {
      "must": { "match_all": {} },
      "filter": {
        "range": {
          "balance": {
            "gte": 20000,
            "lte": 30000
          }
        }
```

```
      }
    }
  }
}
'

#########################################################################
Analyze results with aggregations
You can search documents, filter hits, and use aggregations to analyze the results all in one request.

# The following request uses a terms aggregation to group all of the accounts in the bank
# index by state, and returns the ten states with the most accounts in descending order:

curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "state.keyword"
      }
    }
  }
}
'

#output
...
  "buckets" : [
        {
          "key" : "TX",
          "doc_count" : 30
        },
...


# You can combine aggregations to build more complex summaries of your data.
# For example, the following request nests an avg aggregation within the previous group_by_state
# aggregation to calculate the average account balances for each state.

curl -X GET "localhost:9200/bank/_search?pretty" -H 'Content-Type: application/json' -d'
{
  "size": 0,
  "aggs": {
    "group_by_state": {
      "terms": {
        "field": "state.keyword"
      },
      "aggs": {
        "average_balance": {
          "avg": {
            "field": "balance"
```

```
        }
       }
      }
    }
  }
}
'

# output
...
    "buckets" : [
      {
        "key" : "TX",
        "doc_count" : 30,
        "average_balance" : {
          "value" : 26073.3
        }
      },
...
```