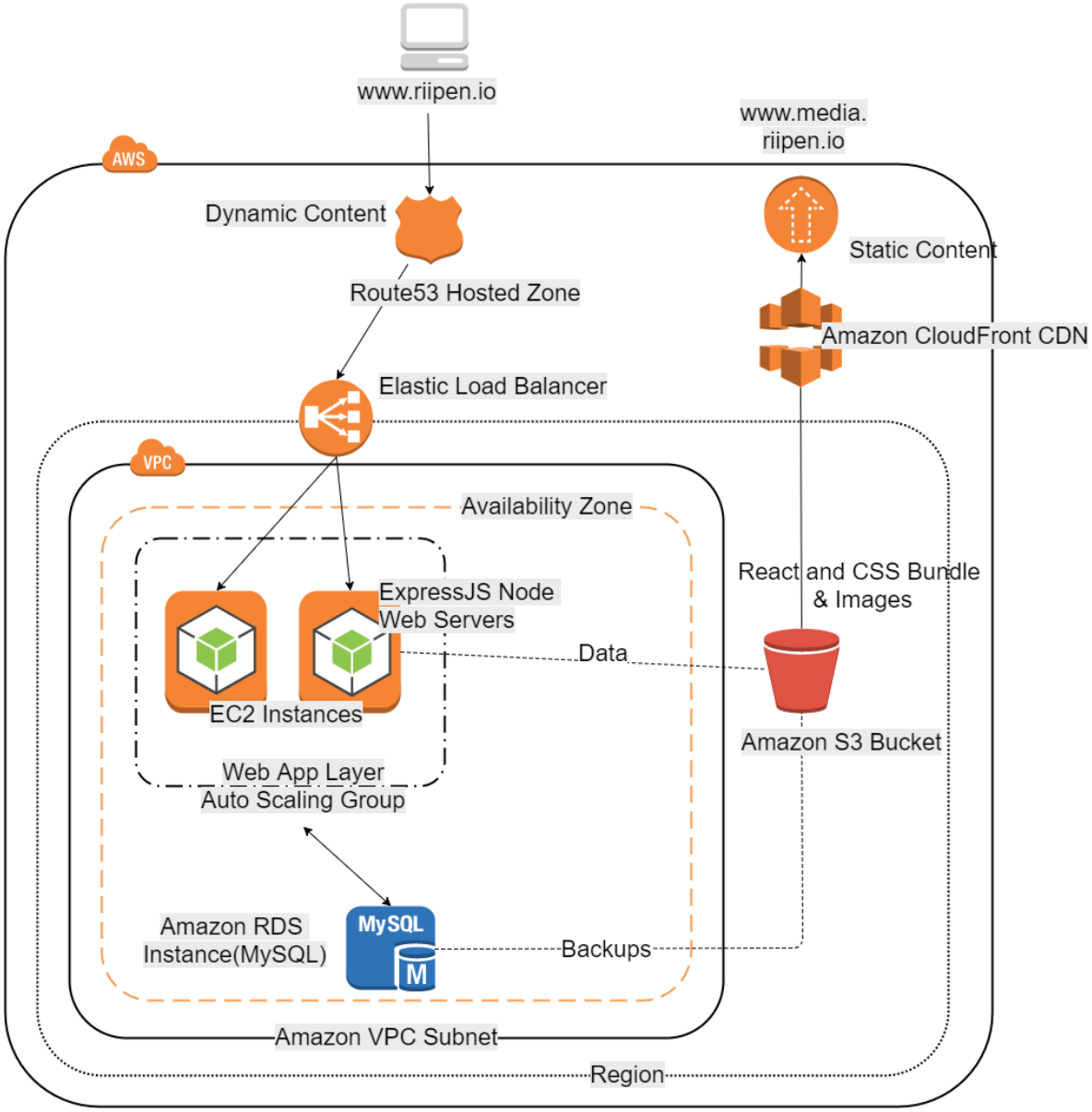
BIG DATA AND E-COMMERCE PROJECT

Group members:
Aaron Pereira
Adrian Texeira
Arya Mukherjee
Tarjani Shah

Table of Contents

1. RIIPEN'S CURRENT ARCHITECTURE



2. ELASTIC SEARCH

2.1 DESCRIPTION:

Elasticsearch is an open-source, distributed, readily-scalable, RESTful search and analytics engine. It centrally stores your data, so you can discover the expected and uncover the unexpected. Elasticsearch is accessible through an extensive and elaborate API, Elasticsearch helps extremely fast searches that support multiple data discovery applications.

It is easy to setup and use Elasticsearch. It ships with sensible defaults and hides complex search and distribution mechanics from beginners which means that the learning curve for grasping the basics is very short and you can become productive very quickly. It allows the user to store, search, and analyze big volumes of data quickly and in near real time

2.2 BASIC CONCEPTS

Cluster

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

Node

A node is a single server that is part of the cluster, stores 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. Node name can be defined as per the user's convenience. This name is important for administration purposes where user wants to identify which servers in your network correspond to which nodes in your Elasticsearch cluster.

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.

Type

Within an index, user can define one or more types. A type is a logical category/partition of the index whose semantics is completely up the user. In general, a type is defined for documents that have a set of common fields.

Document

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

Sharding

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

Replication

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:

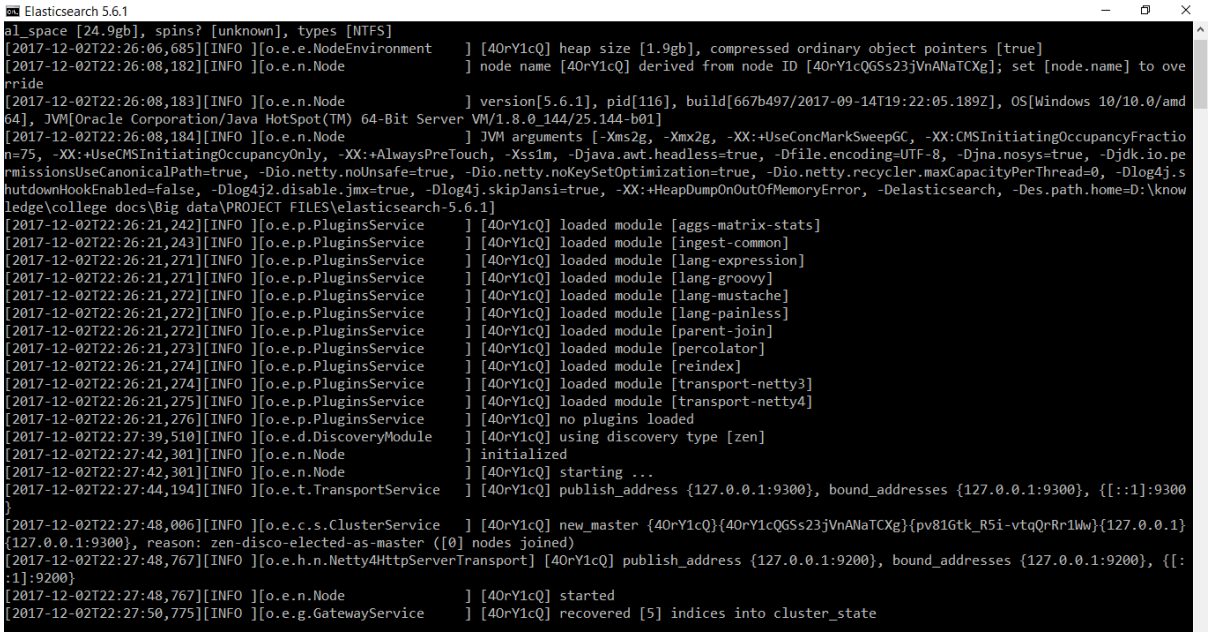
- It provides high availability in case a shard/node fails. For this reason, it is important to note that a replica shard is never allocated on the same node as the original/primary shard that it was copied from.
- It allows you to scale out your search volume/throughput since searches can be executed on all replicas in parallel.

2.3 INSTALLATION:

One of the most important pre-requisites for Elastic search is JAVA version 8. Once this is installed, one can go to the elastic search website and download it. It is available for download in various options which include .zip, .tar, .msi, etc. As windows user we downloaded .zip file for elastic search and ran the elasticsearch.bat after unzipping the file.

Following link can be referred for the same:

https://www.elastic.co/guide/en/elasticsearch/reference/5.0/_installation.html/



Once elastic search service is up and running, next thing to check configurations, configuration setting can be found in **elasticsearch.yml.** The node name given by elastic search, in our case '4OrY1cQ' can be update with any name of user's choice by running the following command.

./elasticsearch -Ecluster.name=riipencluster -Enode.name=riipennode

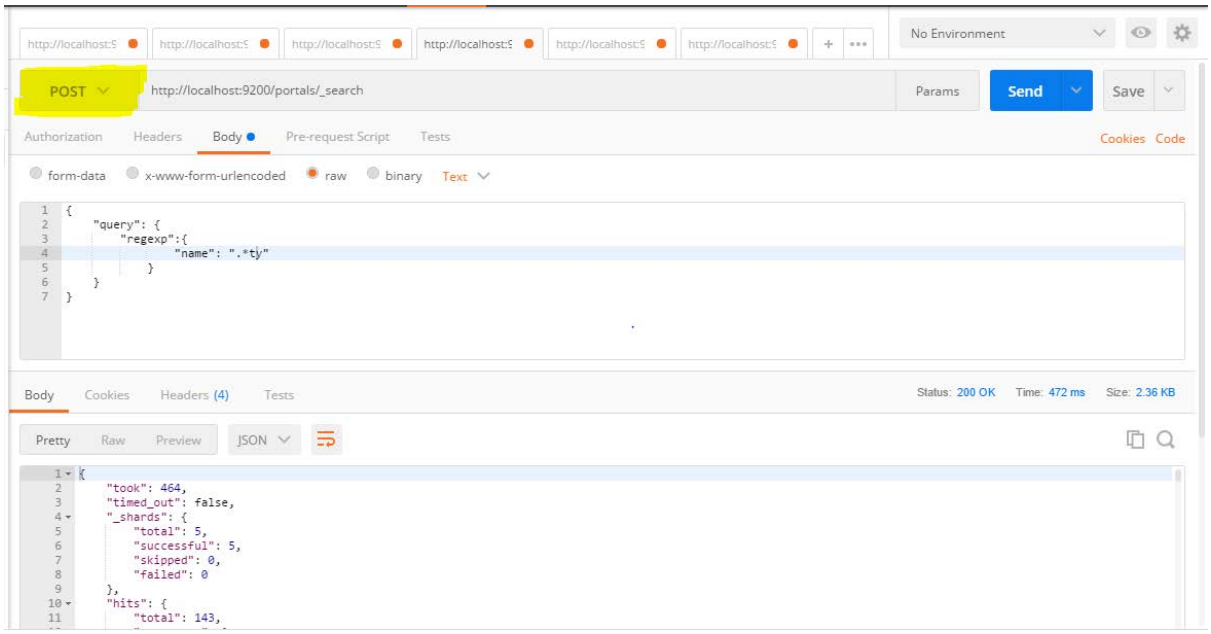
By default Elastic Search creates 5 shards for a default cluster, this can be modified by making changes to the configuration file.

2.4 API USED: *POSTMAN or SENSE PLUGIN*

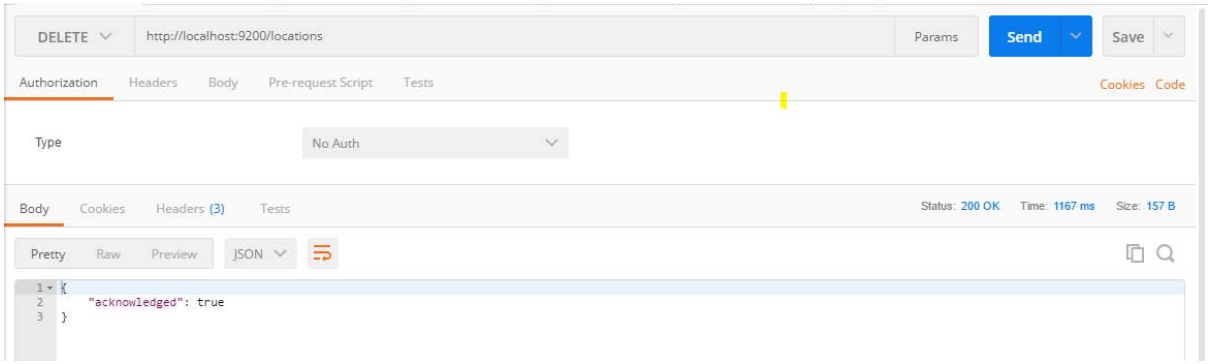
As shown in the screenshot below, query can be written in the body of the postman api, by clicking the '*SEND*' button, one will get results from the elastic search query.

As highlighted, the dropdown provides options like GET, POST, PUT, DELETE, etc.

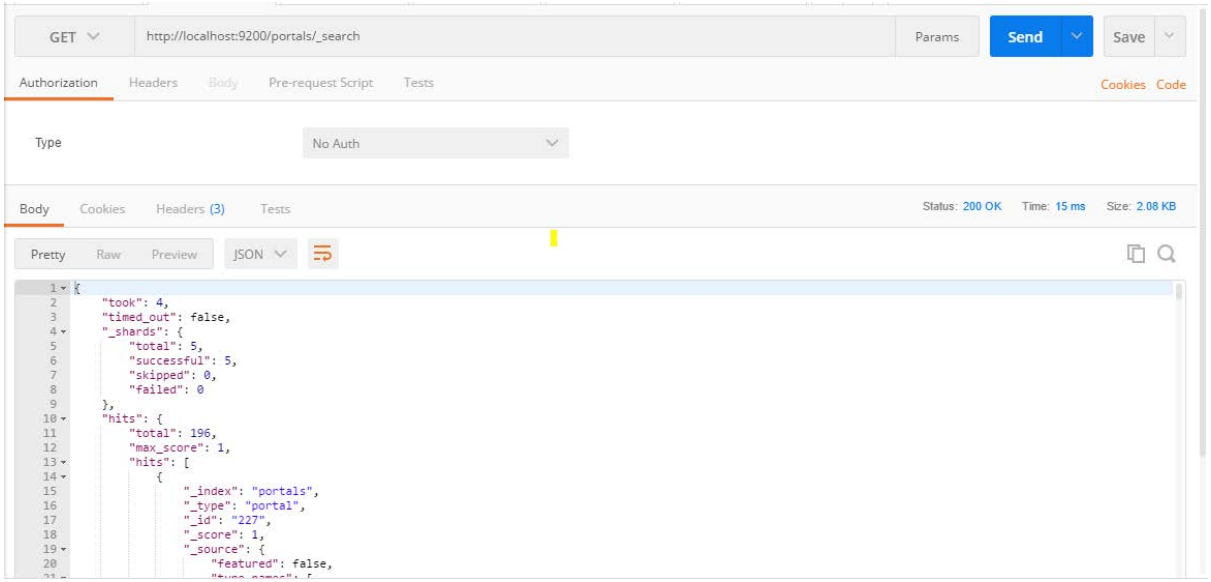
Executing POST command for executing a query



DELETE command screenshot



GET command screenshot



2.5 HANDLING RELATIONSHIP IN ELASTIC SEARCH VIA INDEXING

There are four ways by which association in RDBMS can be carried out to Elastic Search indexes

- Application-side joins
- Data denormalization
- Nested objects
- Parent/child relationships

In our case, we have specified the query with join in the input section of logstah config file. Using this query we are able to carry out the relationship from RDBMS into Elastic search index. This will help in storing the documents in appropriate order and querying would take less time. Hence planning on how to store the index is of utmost importance in Elastic search.

References:

https://www.elastic.co/guide/en/elasticsearch/reference/5.0/

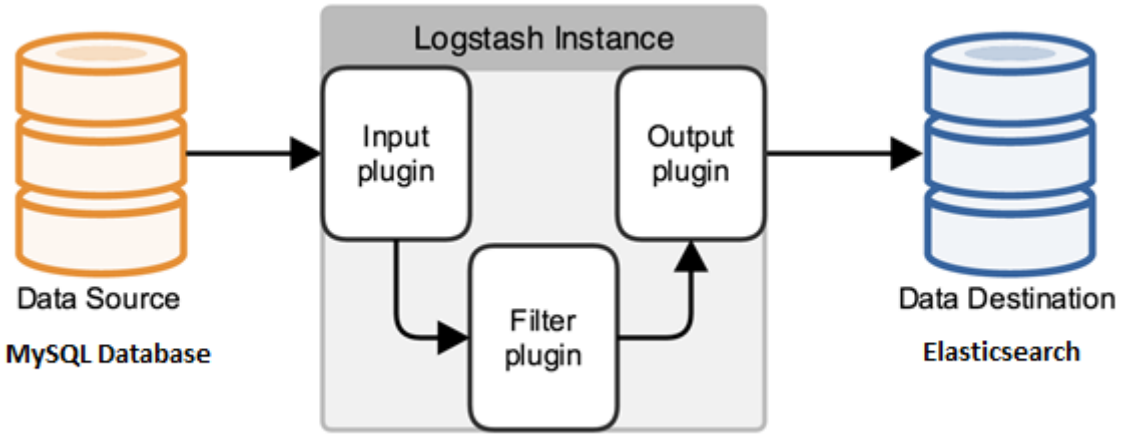
3. LOGSTASH

3.1 OVERVIEW:

Logstash is a server-side data processing pipeline used to extract the data from different sources (MySQL database in our project) and convert them to documents in Elasticsearch. It is a tool that collects, processes and forwards events.

Data is collected via configurable input plugins. Once an input plugin has collected data it can be processed by the filters which modify and validate the event data. Finally, logstash routes events to output plugins which can forward the events to an external program like Elasticsearch.

3.2 PLACEMENT OF LOGSTASH IN THE RIIPEN ARCHITECTURE.



Features:

a) Queueing – If there is a failure between any of the stages with in the pipeline of logstash then the data is saved in the queue log and then reinjected in the pipeline.
b) Clustering – Logstash can coordinate between multiple nodes or pipeline.
c) Introspection – We can verify how many requests in queue are pending or is the queue full.

3.3 INSTALLATION

i. Pre-requisite: Java 8. Note: Java 9 is not supported.
ii. Download Logstash from the link: https://www.elastic.co/downloads/logstash
iii. Unzip the file "logstash-x.x.x" file.

3.4 CONFIG FILE

The Logstash config file ("logstash.conf") will be present in the bin folder. This file is used to specify the plugin and the setting for those plugins.

Structure of the config file: The config file consists of 3 sections. Each section can contain single or multiple plugin along with its parameters.

a) **Input** - Events are generated using the input plugin. E.g.: a line/row in a document/table may be an event.
b) **Filter** - Filters are used to transform, drop and validate an event. It is an optional section.

c) **Output** – Output is used to send an event to the outside world (i.e. Elasticsearch)

```
Input {

    }

Filter {

    }

Output {

    }
```

a) **Input**: A source of event to be read by Logstash will be mentioned in the Input section. Since Riipen's data resides on MySQL database we have used the Java database connectivity (JDBC), which act as an interface to ingest data into logstash.

We have used the following version of jdbc plugin: **mysql-connector-java-5.1.44**

Below is a sample setting of the jdbc plugin to fetch data from MySQL database:

```
input{
        jdbc{
                jdbc_connection_string => "jdbc:mysql://localhost:3306/riipendb"
                jdbc_user => "*****"
                jdbc_password => "******"
                jdbc_driver_library => "F:/Softwares/logstash-6.0.0/bin/mysql-connector-
java-5.1.44/mysql-connector-java-5.1.44-bin.jar"
                jdbc_driver_class => "com.mysql.jdbc.Driver"
                statement => "Select eg.* from assignments a,
                                (select   a.id ,
                                        a.name as name,
                                        a.summary as summary,
                                        t.type,
                                        t1.name as tag_name
                                from    assignments a,
                                        taggings t,
                                        tags t1,
                                where   a.id = t.taggable_id
                                and t.tag_id = t1.id)
                                eg where a.id=eg.id;"

                schedule => "1 * * * *"

  }
}
```

Parameters used in Input section:
jdbc_connection_string: MySQL jdbc connection to our database (riipendb).
jdbc_user: Database Username
jdbc_password: Database Password
jdbc_driver_library: Directory/path of the downloaded jdbc plugin.

jdbc_driver_class: JDBC driver class to load
Statement: Query to create documents in Elasticsearch.
Schedule: The scheduling syntax is powered by rufus-scheduler. Each asterisk ‘ * ‘ has a definition and a value that it stands for.
1st asterisk: minute of the hour the input from the plugin will execute.
2nd asterisk: hour of the day in 24hr format.
3rd asterisk: Day of the week
4th asterisk: Month
5th asterisk: Time zone

E.g.: 5 10 * * * the input to the plugin will execute every 5th minute of 10am every day
1 * * * * the input to the plugin will execute every minute.
6 8 * 1-5 * the input to the plugin will execute on 6th minute of 8am every day between the month January – May.

b) **Filter**: A filter performs intermediary processing like transforming and validation of an event
Below is a sample setting of the filter stage:

```
filter {
    aggregate {
      task_id => "%{id}"
      code => "
                    map['id'] = event.get('id')
                   map['name'] = event.get('name')
                    map['summary'] = event.get('summary')
                    map['type_names'] ||= []
                    map['type_names'] << {'type_name' => event.get('type'),'tag_name' =>
event.get('tag_name')}
        event.cancel()
      "
      push_previous_map_as_event => true
      timeout => 5
    }
}
```

Aggregate plugin: It will aggregate the data from several events of the same task and then send the aggregated data to the output section.

“The filter needs a task_id (i.e id) to correlate events of a same task. At the beginning of the task the filter creates a map attached to task_id (id). For each event, you can execute code using event and map. After the final event, the map attached to task is deleted. In one filter configuration, it is recommended to define a timeout option to protect the feature against unterminated tasks. It tells the filter to delete expired maps. If no timeout is defined, by default, all maps older than 1800 seconds are automatically deleted. All timeout options have to be defined in only one aggregate filter per task_id pattern.”[1]

Since an ID can have many tags, we have created an array “type_names” that contains all the tags associated to a particular id. In this way we are creating nested objects.

push_previous_map_as_event: When this option is enabled, each time aggregate plugin detects a new task id, it will push the previous aggregate map as a new event, and then creates a new empty map for the next task.

c) **Output**: This is the final stage in the pipeline. An output plugin sends event data to the specified destination.
Below is a sample setting of the output plugin:

```
output{
        stdout{
                codec => json_lines
        }
        elasticsearch{
                "hosts" => "localhost:9200"
                "index" => "talent"
                "document_type" => "skills"
                "document_id" => "%{id}"
        }
}
```

Stdout: used to print a simple output on the shell running Logstash.
Json_lines: This codec will decode streamed JSON
Elasticsearch: This plugin is used for storing logs in Elasticsearch.
Hosts: Specify the conection details of Elasticsearch.
Index: Name of the index to be created.
Document_type: This is the document type to write events to.
Document_id: The document ID for the indexes

3.5 CONFIGURE MYSQL TO ELASTICSEARCH THROUGH LOGSTASH

1. Download the logstash dependancies and set up folder from the elastic search website : https://www.elastic.co/downloads/logstash

2. Download the mysql jdbc jar and add in the bin folder(mysql-connector-java-5.1.44-bin.jar)

3. Create the logstash.conf file with all the configuration details and add it in the bin folder.

4. Install logstash jdbc by following command (run in bin folder)
--> bin\logstash-plugin install logstash-input-jdbc

5. Install the configuration file created above by following command (run in bin folder)
--> logstash.bat -f .\logstash.conf

NOTE : The path of logstash folder should not have any spaces in the folder name
for eg c://downloads/big data/riipen project ----- such file path won't work where there is a space in folder name like big data or riipen project so remove spaces.
Tables will be loaded further
Go to postman check by firing queries

4. QUERY DSL

“Elasticsearch provides a full Query DSL based on JSON to define queries. It has two types of clauses:

Leaf query clauses

Leaf query clauses look for a particular value in a particular field, such as the match, term or range queries. These queries can be used by themselves.

Compound query clauses

Compound query clauses wrap other leaf **or** compound queries and are used to combine multiple queries in a logical fashion (such as the bool or dis_max query), or to alter their behavior (such as the constant_score query).”[2]

4.1 RELEVANCE SCORING IN ELASTIC SEARCH

“Lucene (and thus Elasticsearch) uses the Boolean model to find matching documents, and a formula called the practical scoring function to calculate relevance. There are 2 main concepts involved

1. Term Frequency (TF): calculates how often does the term appears in the document
    - Calculated as : tf(t in d) = √frequency
    - The term frequency (tf) for term t in document d is the square root of the number of times the term appears in the document.
2. Inverse Document Frequency: calculates how often does the term appears in all the documents.
    - Calculated as : idf(t) = 1 + log (numDocs / (docFreq + 1))
    - The inverse document frequency (idf) of term t is the logarithm of the number of documents in the index, divided by the number of documents that contain the term”[3]

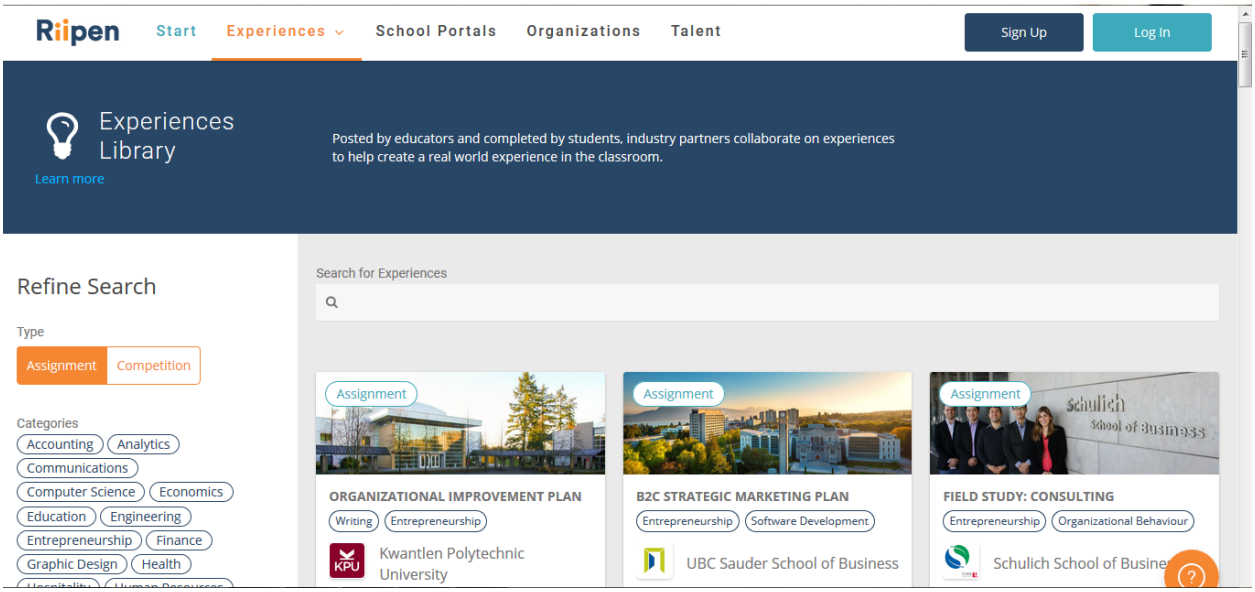
Scenarios/ use-cases

Following are the few scenarios/use-cases listed:

1. To return all the documents indexed in elastic search whenever any user hits the Riipen website and search for assignments. i.e. the first page that the user lands on hitting the website.

    Query1:

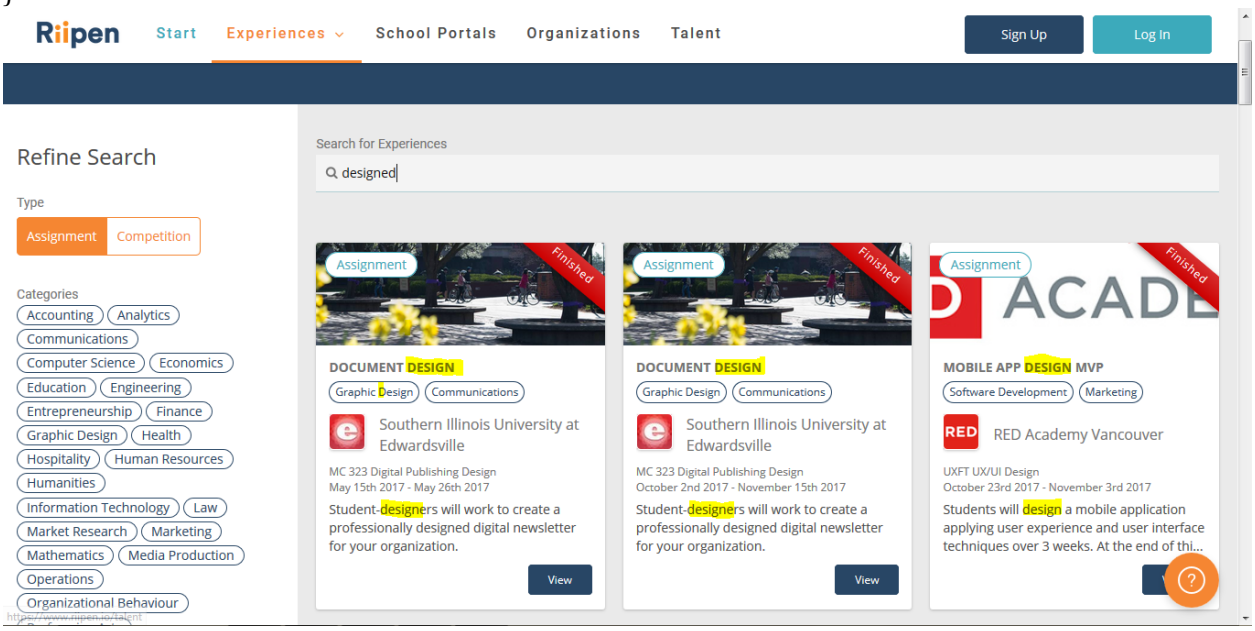
    ```
    {
       "query": {
          "match_all" : {}
       }
    }
    ```



2. To search for any specific term, say 'designed' in all the attributes across all the documents that has been indexed.

Query2:

```
{
   "query": {
      "query_string" : {
         "query" : "designed"
      }
}
```



3. To return the exact match that is mentioned in the search bar
   Query3a:

```
{
  "query": {
    "match_phrase" : {
      "type_names.tag_names" : "computer science"
    }
  }
}
```

   OR
   Query3b:

```
{
  "query": {
   "bool": {
    "must": [
     { "match": { "type_names.tag_names": "Computer Science" } }
    ],
    "must_not": [
     { "match": { "type_names.tag_names": "food" } }
                 { "match": { "type_names.tag_names": "env" } }
    ]
   }
  }
}
```

4. To allow fuzziness or exception while typing.
   The scenarios where it can be used is to allow some human error or typos while typing in the search bar. i.e if a user types scieces instead of science
   Query4:

```
{
  "query": {
    "fuzzy" : {
      "type_names.tag_names" : {
        "value" :"sciece",
        "fuzziness" :1
      }
    }
  }
}
```

5. To give more importance/priority/preference (termed as Boosting) to any field or attribute, in this case tag_name is given higher boosting of 5 as compared to name or summary
   Query5:

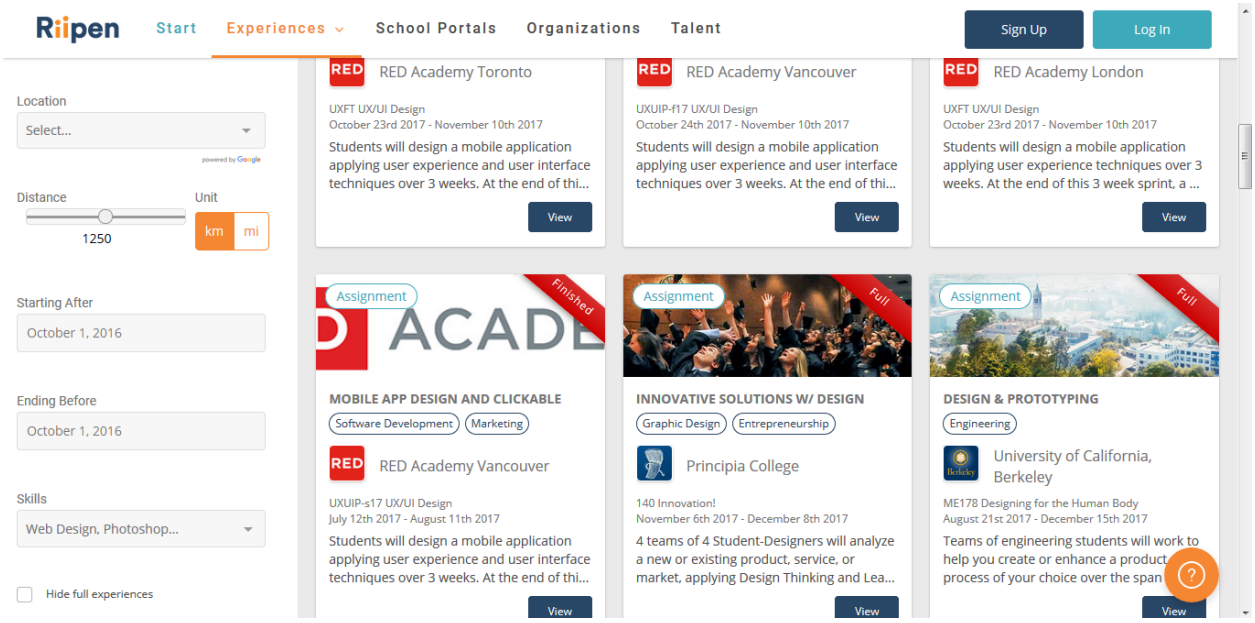
```
{
  "query": {
   "multi_match" : {
    "query" : "strategy" ,
    "fields" : [ "tag_name^5", "name^2", "summary" ]
   }
  }
}
```

In the above query, if the keyword “Strategy” if found in the tag_name will have higher score than name or summary as the score calculated for each document by Elasticsearch will be multiplied by 5 as the boosting given is 5 to the tag_name. Similarly the score of the document will be multiplied by 2 if the keyword is found in the “name” field. Since the documents are listed in the descending order of the score calculated by Elasticsearch, tag_name will have high score and hence appear much higher in the list

6. Location based search

Query6:

```
{
   "query": {
      "filtered" : {
         "query" : {
            "match_all" : {}
         },
         "filter" : {
            "geo_distance" : {
               "distance" : "20km",
               "Location" : {
                  "lat" : 37.776,
                  "lon" : -122.41
               }
            }
         }
      }
   }
}
```



7. To match a keyword and ignore certain other words while searching something very much specific.
Query7:

```
{
  "query": {
   "bool": {
    "must": [
     { "match": { "type_names.tag_names": "Computer Science" } }
    ],
    "must_not": [
     {    "match": { "summary": "science" }},
     {    "match": { "name": "science" }}
    ]
   }
  }
}
```

In this example the query returns value where the keyword “Computer Science” will be found in tag_name but ignored in summary and name fields

8. To view courses related to say business we can give more boost to the college having the word BUSINESS than any other school/university as other universities might have other courses than Business to offer but a Business school will only have business related courses and hence higher scoring and other colleges/universities can be listed later in the results.
Query8:

```
{
   "query": {
      "wildcard" :
         { "name" :
                  { "value" : "*business*" ,
                    "boost" : 2.0
                    }
         }

   }
}
```

In the above query we will be using wildcard and boost keywords.

9. To find school portals which are featured or not.
Query9:

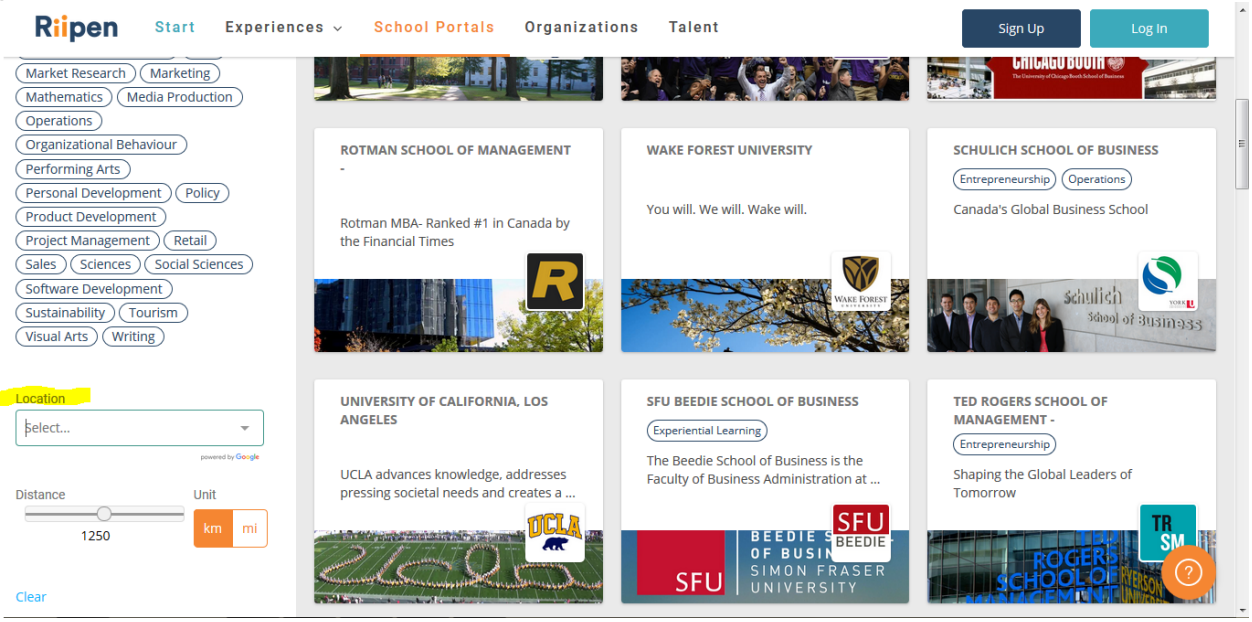
```
{
   "query": {
      "match": {
         "featured": false
      }
   }
}
```

10. To search portals based on locations and allows fuzziness i.e. some typos / human error

Query10:

```
{
   "query": {
      "match": {
         "city":{
         "query" : "bston",
          "fuzziness" : 2
          }

      }
   }
}
```



11. To use range query to boost the results if the distance is greater than or less than any given value

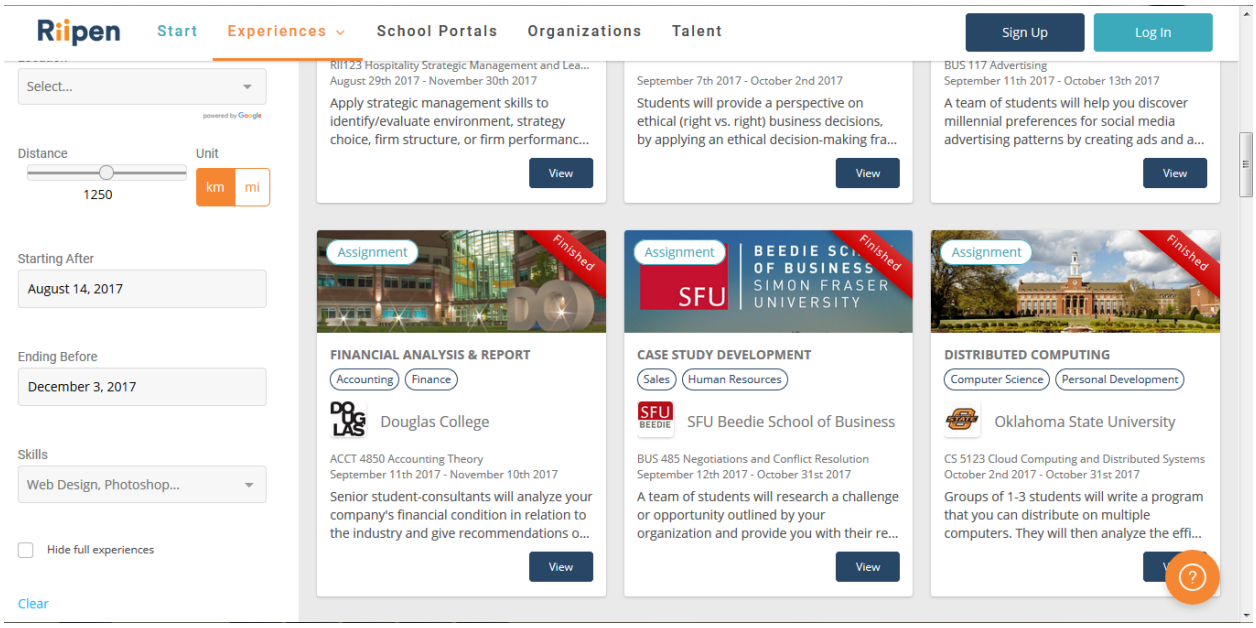
Query12:

```
{
   "query": {
      "range" : {
         "distance" : {
            "gte" : 0,
            "lte" : 5,
            "boost" : 2.0
         }
      }
   }
}
```

12. To find the assignments with the range of specific start and end date.

Query13:

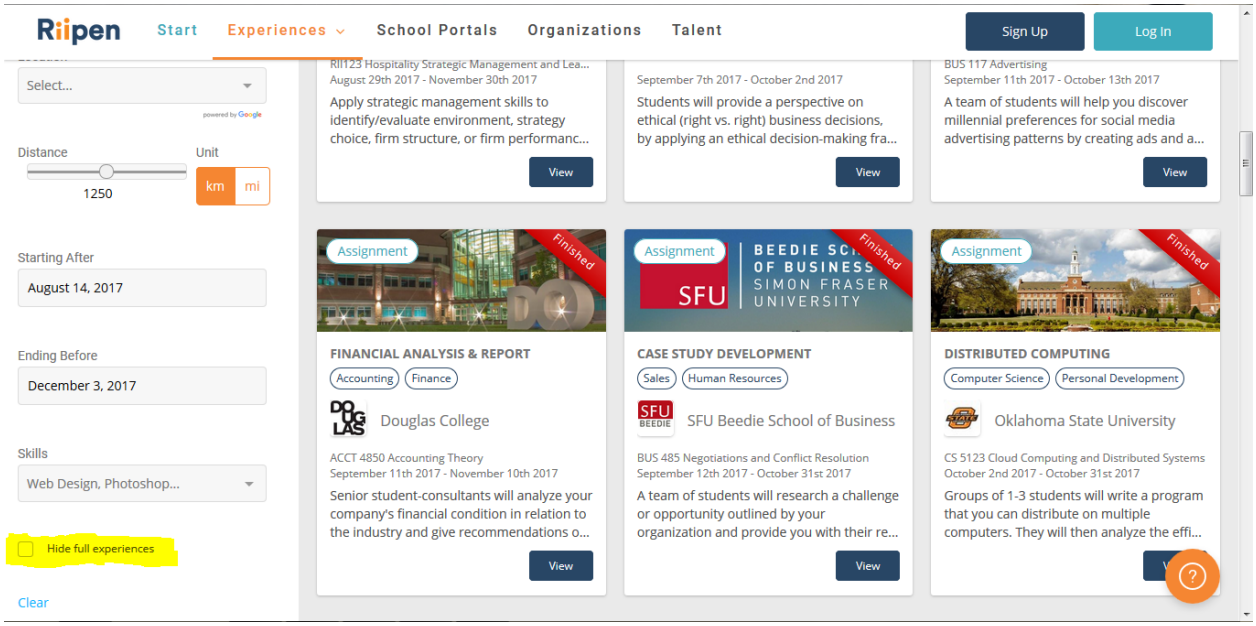
```
{
   "query": {
      "range" : {
         "start_date" : {
            "gte" : "2017-01-30T05:00:00.000",
            "lte" : "2017-10-30T05:00:00.000Z"
         }
      }
   }
}
```



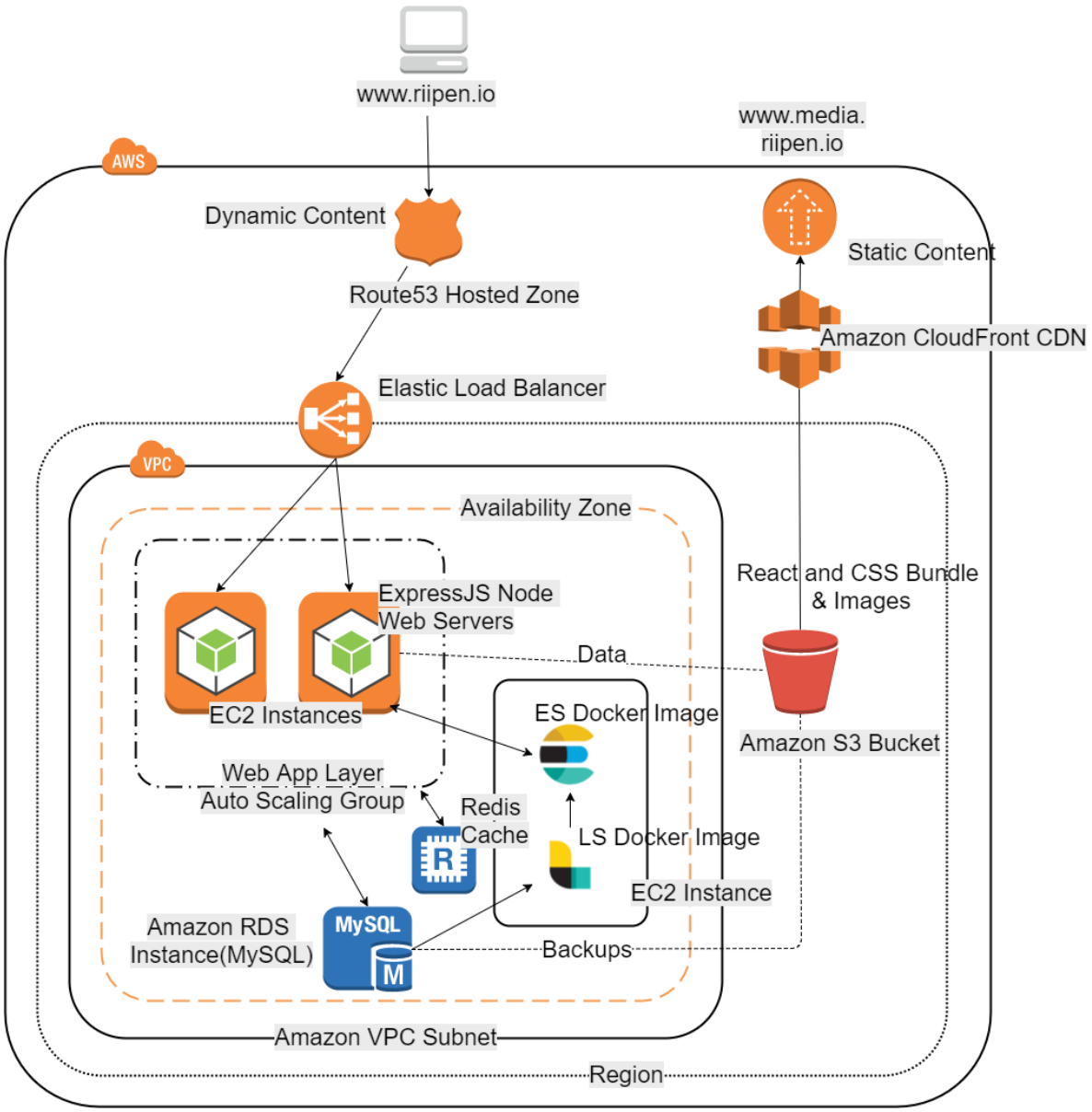
13. To get the list of all assignments those are not yet full/finished.

Query14:

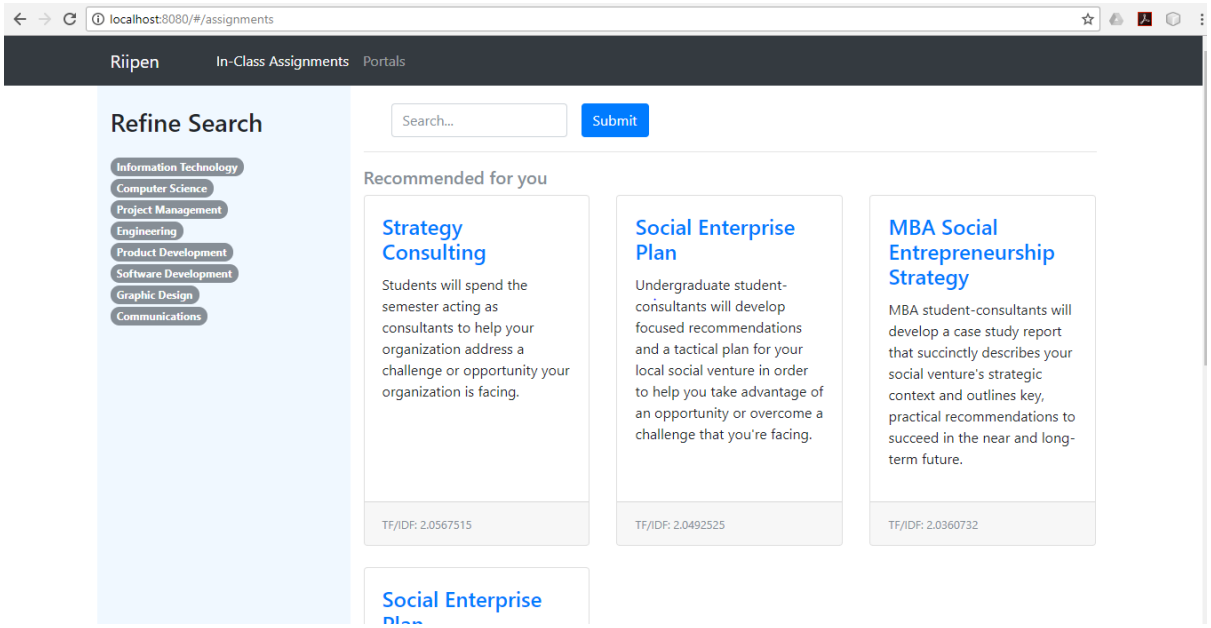
```
{
   "query": {
      "match": {
         "full": false
      }
   }
}
```


Riipen
Start
Experiences
School Portals
Organizations
Talent
Sign Up
Log In
Select...
Distance
Unit
1250
km
mi
Starting After
August 14, 2017
Ending Before
December 3, 2017
Skills
Web Design, Photoshop...
Hide full experiences
Clear
August 29th 2017 - November 30th 2017
Apply strategic management skills to identify/evaluate environment, strategy choice, firm structure, or firm performanc...
View
September 7th 2017 - October 2nd 2017
Students will provide a perspective on ethical (right vs. right) business decisions, by applying an ethical decision-making fra...
View
BUS 117 Advertising
September 11th 2017 - October 13th 2017
A team of students will help you discover millennial preferences for social media advertising patterns by creating ads and a...
View
Assignment
Finished
FINANCIAL ANALYSIS & REPORT
Accounting
Finance
Douglas College
ACCT 4850 Accounting Theory
September 11th 2017 - November 10th 2017
Senior student-consultants will analyze your company's financial condition in relation to the industry and give recommendations o...
View
Assignment
Finished
SFU
SIMON FRASER UNIVERSITY
CASE STUDY DEVELOPMENT
Sales
Human Resources
SFU Beedie School of Business
BUS 485 Negotiations and Conflict Resolution
September 12th 2017 - October 31st 2017
A team of students will research a challenge or opportunity outlined by your organization and provide you with their re...
View
Assignment
Finished
DISTRIBUTED COMPUTING
Computer Science
Personal Development
Oklahoma State University
CS 5123 Cloud Computing and Distributed Systems
October 2nd 2017 - October 31st 2017
Groups of 1-3 students will write a program that you can distribute on multiple computers. They will then analyze the effi...

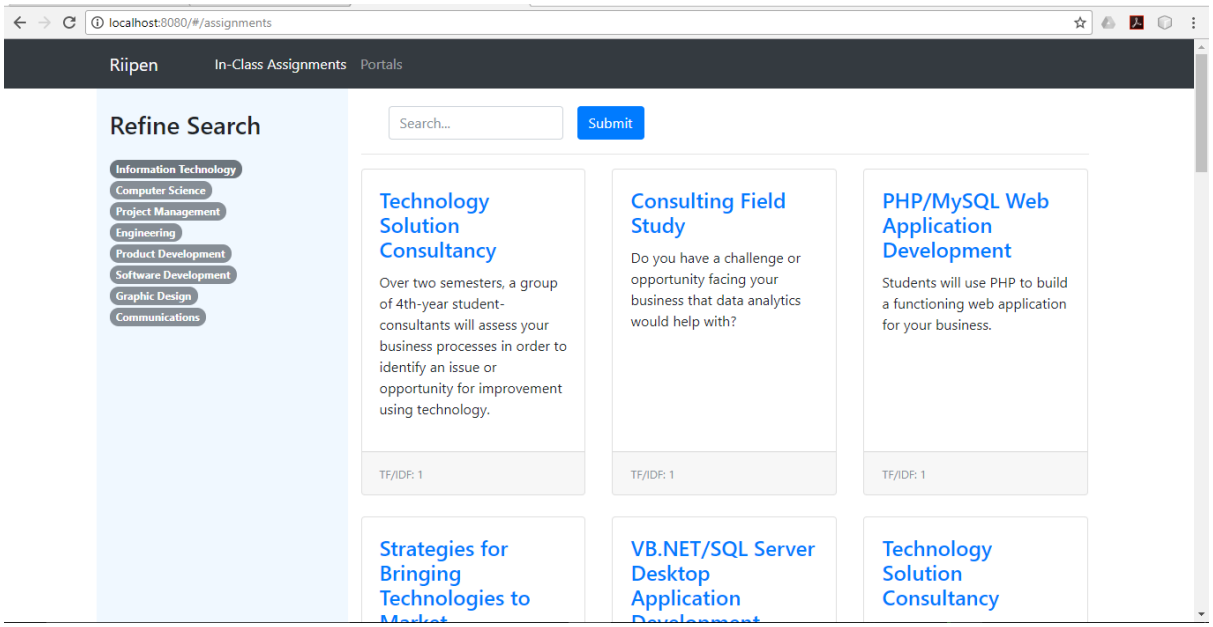
5. PROPOSED ARCHITECTURE



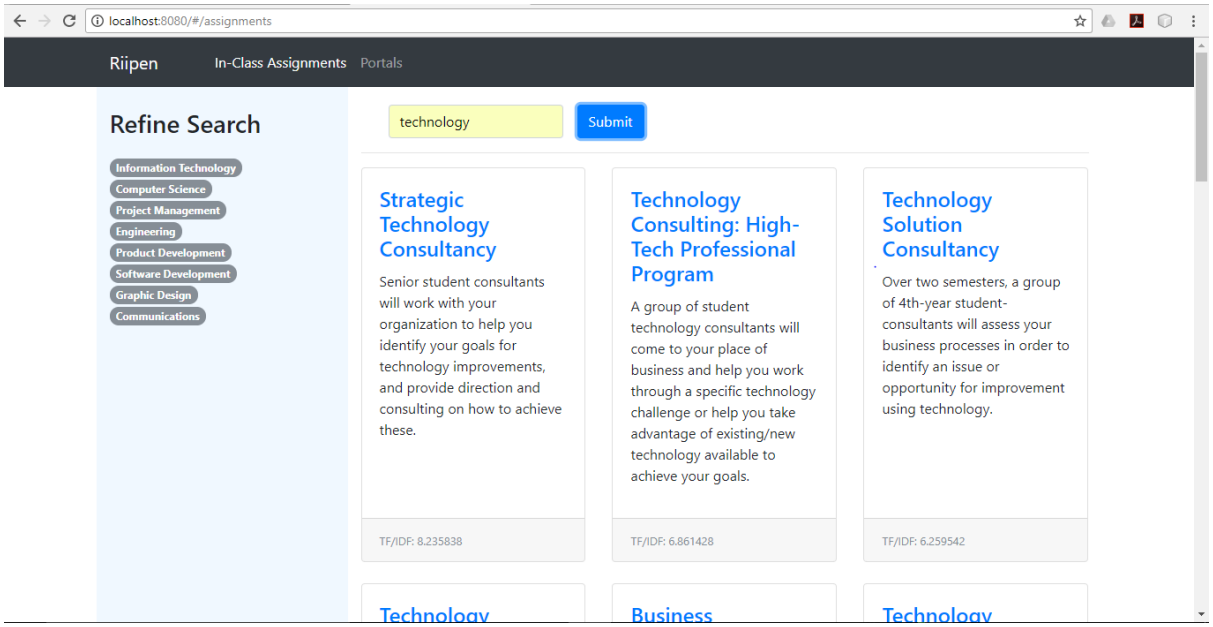
6. APPLICATION PROTOTYPE



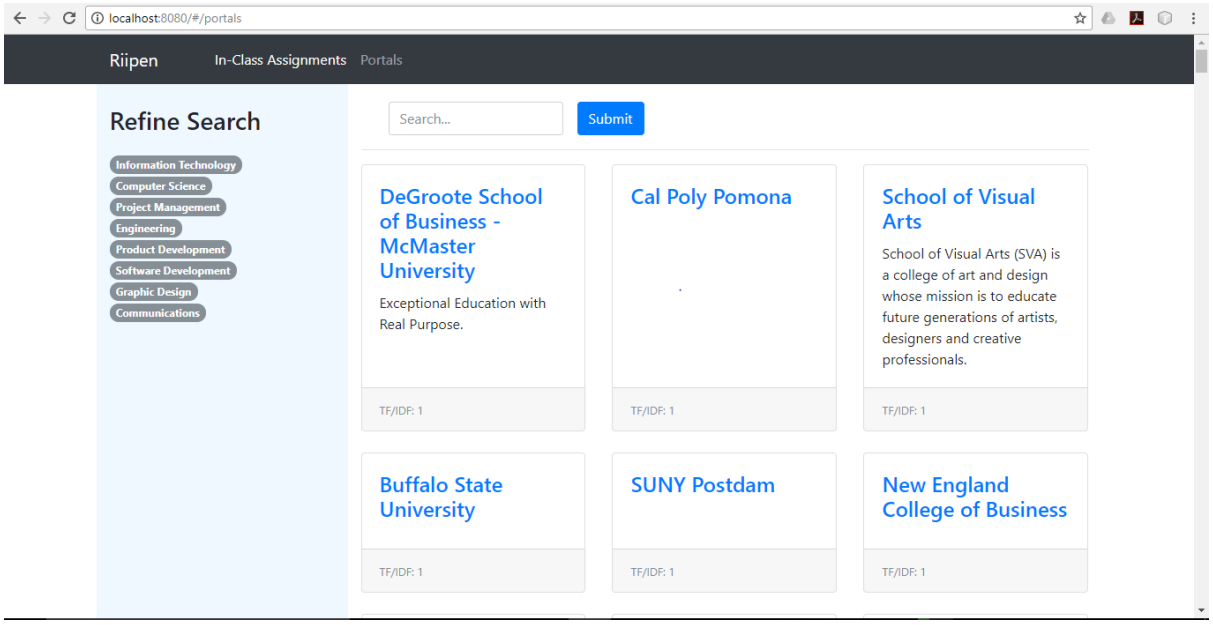
On clicking Information Technology keyword



Search keyword technology



Portals page



7. PROPOSED FUNCTIONALITIES

7.1 REDIS CACHING

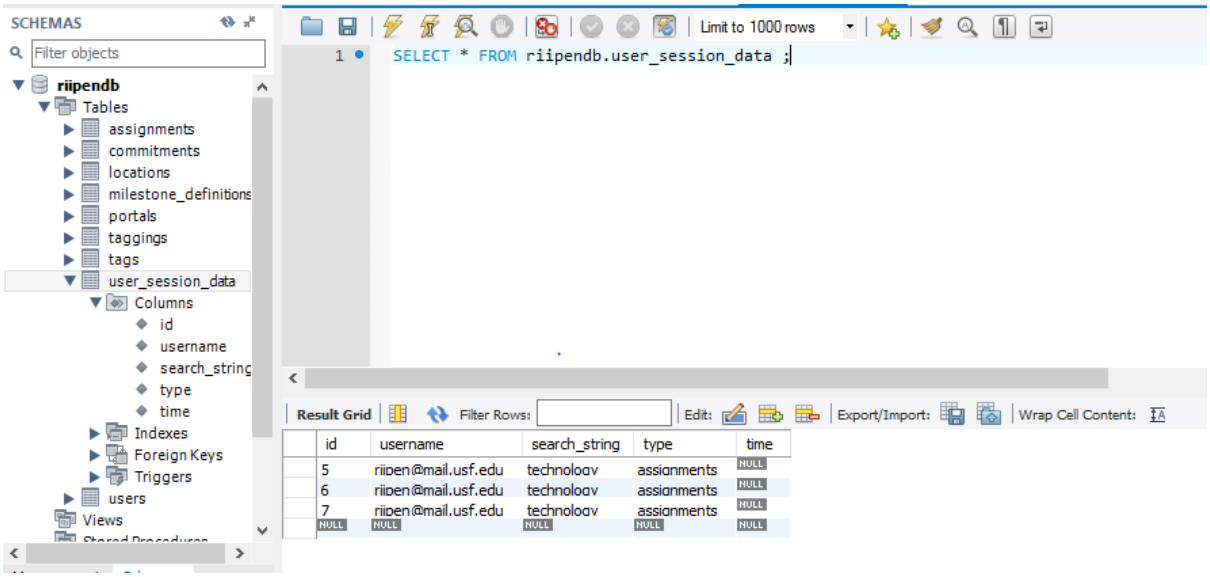
Redis is a key value in-memory data structure store used for caching. We will be caching the first 20 records in Redis whenever the user logs in for the first time. Due to the caching mechanism we save a lot of time in fetching and retrieving data.

7.2 RECOMMENDATION

The recommendation functionality is also included which is based on the location_id of the user. We will be recommending the user with assignment nearby the location_id.

7.3 LOGGING FUNCTIONALITY

A table user_session_data has been created to log users search activities. Whenever a user searches for any keyword it is recorded in the user_session_data table. Using this data, we can provide better recommendations using various analytical tools to the users based on the history of searches.



Reference:
[1] https://www.elastic.co/guide/en/logstash/5.1/plugins-filters-aggregate.html
[2] https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl.html
[3] https://www.elastic.co/guide/en/elasticsearch/guide/current/scoring-theory.html