

CIS655/CSE661: Advanced Computer Architecture

Fall 2015

Final Project Report

Date: 12/11/2015

Submitted By

Name	SUID
Saurabh Gupta	428803631
Choudhary Abdulvafa	671290061
Patel Karankumar	874028780

Table of Contents

1. Topic : Wikipedia Trend Tracking with MapReduce	3
2. Design.....	3
3. Implementation	5
3.1. Script to Download Wiki Data.....	5
3.2. Code to upload files to S3	7
3.3. Map-Reduce job.....	8
3.3.1. Job 1	9
3.3.2. Job 2	11
4. Running job in Elastic MapReduce.....	13
5. Visualizing Top 10 Trending Topics using Kibana	16
5.1 Settings to Open Kibana in Web Browser on Windows	16
5.2 Visualize Data on Kibana.....	19
6. Conclusion	23
7. References	24

1. Topic : Wikipedia Trend Tracking with MapReduce

This application will find the most trending topics on Wikipedia for a date specified by the user. Wikipedia keeps hourly logs of hit counts for every page present on Wikipedia. We will use this data, preprocess it and run Map Reduce batch jobs to get trending topics for the day. This data will be stored in indexed format, so that a user can query trends for a specific topic.

Wiki dumps are available on <http://dumps.wikimedia.org/other/pagecounts-raw/>

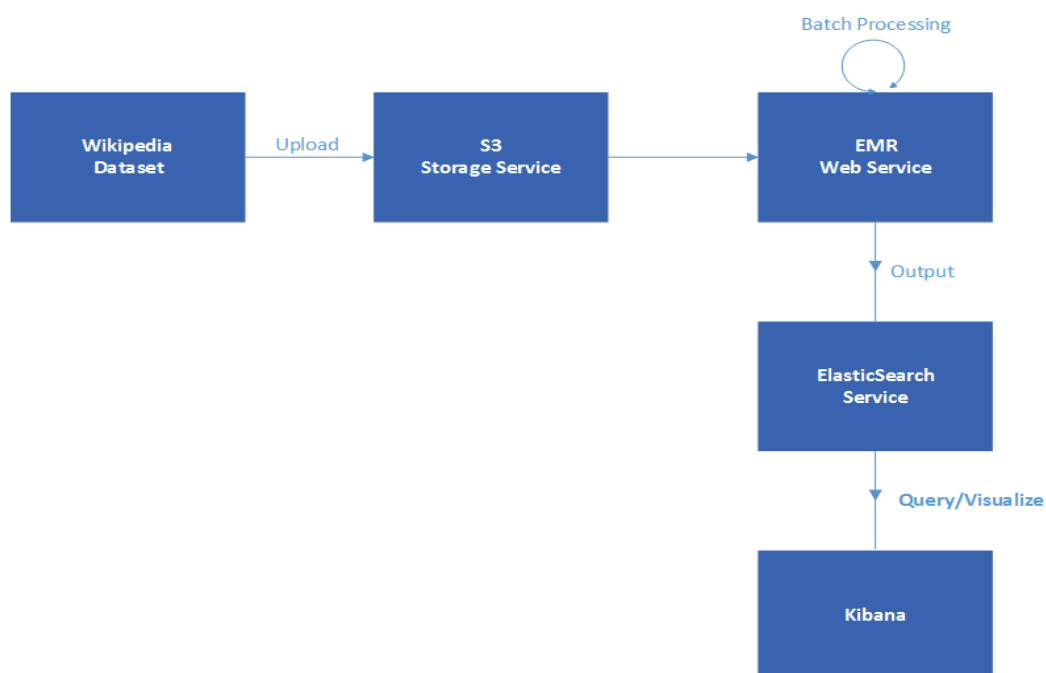
Wiki dumps have following format

projectcode	Pagename	pageviews	bytes
ep	Michael_Jackson	12	557854
es	James_Bond	6	55753
ed	Ronaldo	32	877989

We will run Map Reduce batch jobs on Wiki dumps to get daily counts of Wikipedia article.

2. Design

The following section describes the design of the project in details and explaining each aspect of to how to implement it.



The general design and flow of the application is explain in following steps.

1) The user enters the date for running the analysis.

2) The Wikipedia data dumps downloaded by python script for the day entered by the user.

- The script first forms a URL string using the date entered by the user.
- The format of the urlstring is shown:
#url example <http://dumps.wikimedia.org/other/pagecounts-raw/2015/2015-11/pagecounts-20151101-000000.gz>
The above url is generated for every hour for that day.
- This url is then used to download the files into a input folder on the local machine.

3) Data downloaded in the above step is then uploaded to S3 storage service.

- Amazon S3 is a web service interface designed to store and retrieve large amount of data.
- Amazon S3 allows us to configure lifecycle policies for managing the stored dataset throughout its lifecycle.
- For this project we are uploading the Wikipedia page count file for a single day using an Amazon S3 bucket.
- The java SDK provided by AWS is used to automatically transfer the files to created bucket.
- All this files on S3 ae stored in “input” folder which will be used to run Map Reduce jobs. In order to use java SDK a user has to be added in the Identity and Access Management, the steps of which are shown in the section below.

4) Amazon Elastic MapReduce (Amazon EMR) enables us to run a script at any time during step processing in our cluster.

- We use Hadoop Streaming program that enables us to develop MapReduce executables. These executables reading input from S3 buckets and output data through to Elasticsearch.
- We test these executables locally and upload them to Amazon S3.
- Using the Amazon EMR command line interface or Amazon EMR console to we run this application.
-

5) We are using Logstash, Elasticsearch and Kibana to create an interactive dashboard from output given by Amazon EMR service.

- We are to streaming data from the text source file that we have got from EMR into the database and for that we have wrote configuration file to tell Logstash where it gets the data from, how the data should be transformed and where the data should be inserted.
- We have specified index name in this config file which is important for. This index will be used later for configuring Kibana to visualize the dataset.
- We need to specify the name of the index which we want to use for our visualizations using Kibana.

3. Implementation

3.1. Script to Download Wiki Data

We have written a python script to download the Wikipedia data dumps for the entire day and store it in a folder name "Input" on local desktop.

The python script is shown below:

```
import sys
import os
import zipfile
from urllib2 import urlopen, HTTPError, URLError
from urllib import urlretrieve
from datetime import datetime

def download_file(target,url):
    try:
        data = urlopen(url)
        print "downloading " + url

        with open(os.path.join(target, os.path.basename(url)), "wb") as data_file:
            data_file.write(data.read())
        return data_file
    except HTTPError, e:
        print "Http Error:",e.code, url
    except URLError, e:
        print "URL Error:",e.code, url

var = sys.argv[1]
#Date format = mm-dd-yyyy
date = datetime.strptime(var, '%m-%d-%Y')
print date
#url example http://dumps.wikimedia.org/other/pagecounts-raw/2015/2015-11/pagecounts-20151101-000000.gz
base_url = "http://dumps.wikimedia.org/other/pagecounts-raw/"

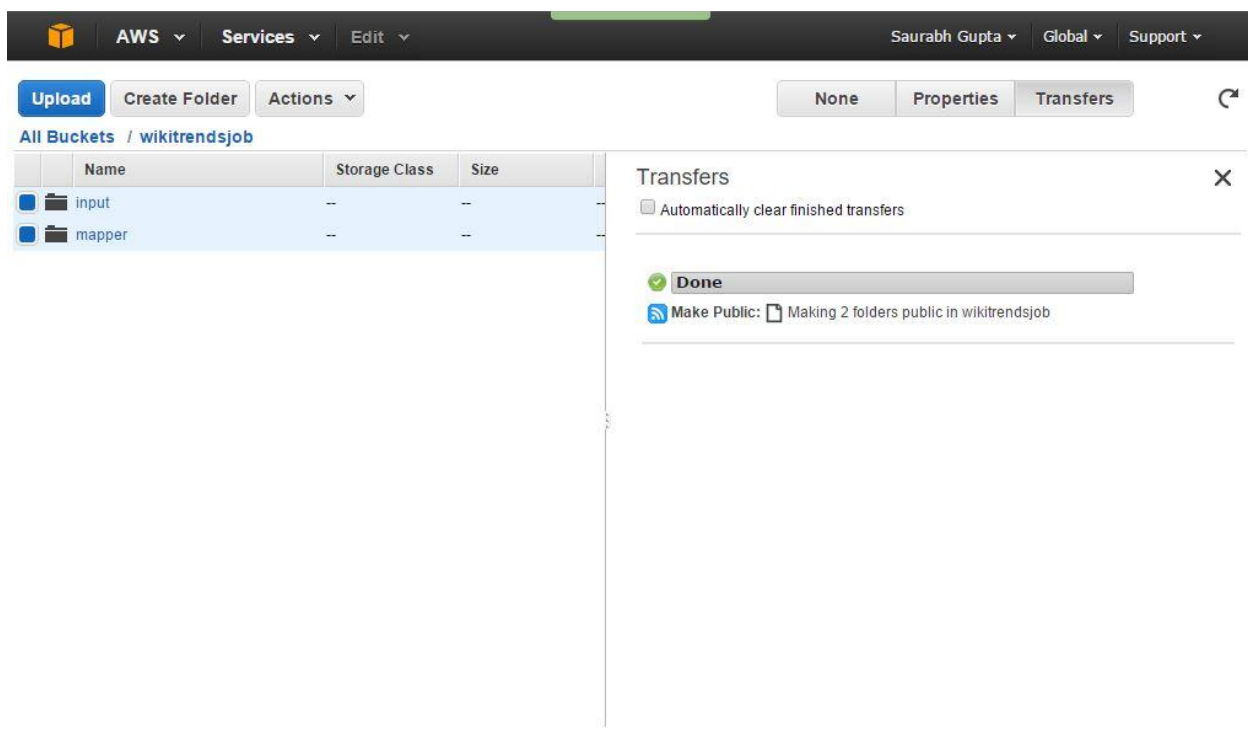
for hour in range(0,24):
    url_string = ( base_url + "%d/%d-%d/pagecounts-%d%d%02d-%02d0000.gz" %
        (date.year, date.year, date.month, date.year, date.month, date.day, hour));

    if not os.path.exists('input'):
        os.makedirs('input');
    download_file('input',url_string)
    print url_string
```

```
C:\Users\Saurabh Gupta\Desktop>python getwikifiles.py 11-02-2015
2015-11-02 00:00:00
downloading http://dumps.wikimedia.org/other/pagecounts-raw/2015/2015-11/pagecou
nts-20151102-000000.gz
http://dumps.wikimedia.org/other/pagecounts-raw/2015/2015-11/pagecounts-20151102
-000000.gz
downloading http://dumps.wikimedia.org/other/pagecounts-raw/2015/2015-11/pagecou
nts-20151102-010000.gz
http://dumps.wikimedia.org/other/pagecounts-raw/2015/2015-11/pagecounts-20151102
-010000.gz
downloading http://dumps.wikimedia.org/other/pagecounts-raw/2015/2015-11/pagecou
nts-20151102-020000.gz
http://dumps.wikimedia.org/other/pagecounts-raw/2015/2015-11/pagecounts-20151102
-020000.gz
```

Running of Python Script to download files

We have also created an Amazon S3 bucket and uploaded the dataset on the S3 storage service.



Folder created in Bucket & Making Folder Public

3.2. Code to upload files to S3

```
import java.io.File;
import java.nio.file.Paths;

import com.amazonaws.AmazonClientException;
import com.amazonaws.AmazonServiceException;
import com.amazonaws.auth.profile.ProfileCredentialsProvider;
import com.amazonaws.event.ProgressEvent;
import com.amazonaws.event.ProgressListener;
import com.amazonaws.services.s3.transfer.MultipleFileUpload;
import com.amazonaws.services.s3.transfer.TransferManager;

public class UploadWikiFiles {
    private static String bucketName = "wikitrendsjob/input";

    public static void main(String[] args) {
        try {
            TransferManager manager = new TransferManager(
                new ProfileCredentialsProvider());
            String dir = new File(Paths.get(".").toAbsolutePath().normalize()
                .toString()).getParent();
            String inputDir = new File(dir).getParent();
            File folder = new File(inputDir + "/input");

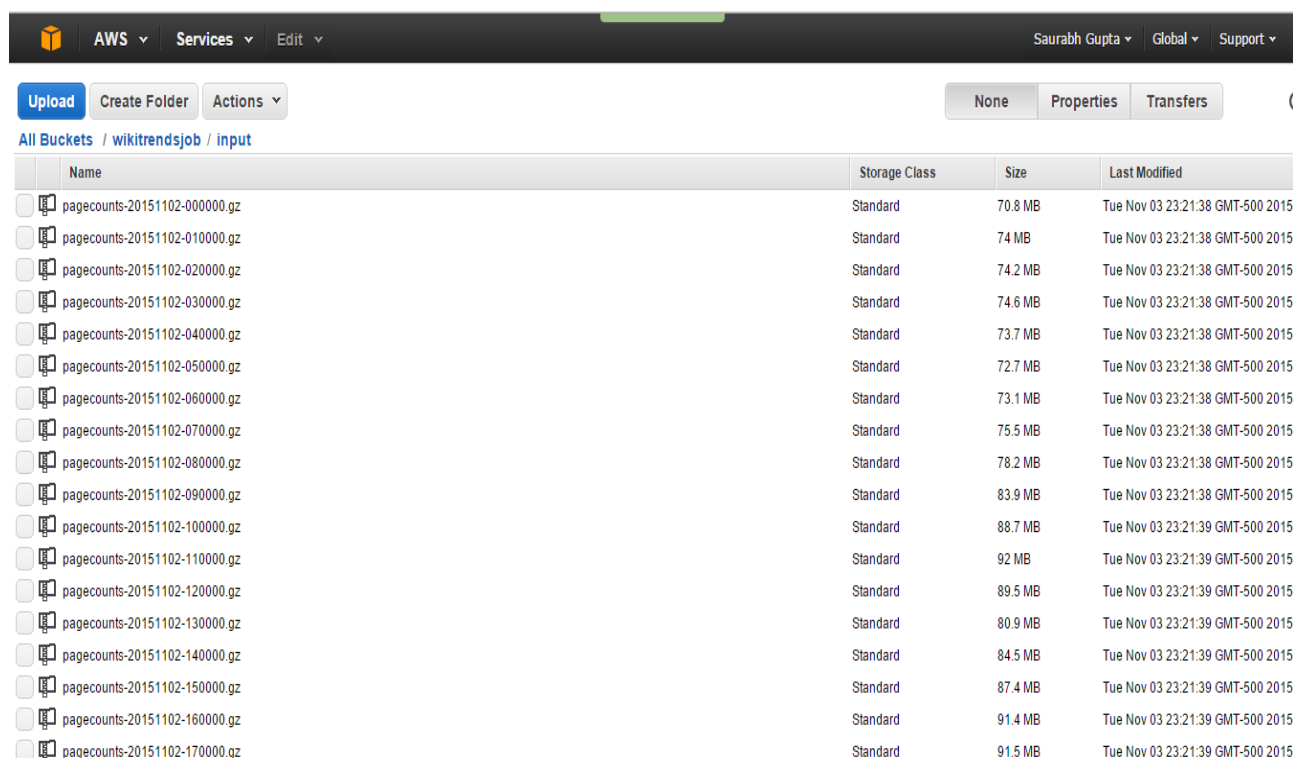
            MultipleFileUpload upload = manager.uploadDirectory(bucketName,
                null, folder, false);
            while (upload.isDone() == false) {

                System.out.println(Math.round(new
                Double(upload.getProgress().getPercentTransferred())) + "%");
                Thread.sleep(5000);
            }

            upload.waitForCompletion();
            manager.shutdownNow();

        } catch (AmazonServiceException e) {
            e.getMessage();
            e.printStackTrace();
        } catch (AmazonClientException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
}
```

Screenshots below shows the uploaded data on the Amazon S3:



Name	Storage Class	Size	Last Modified
pagecounts-20151102-000000.gz	Standard	70.8 MB	Tue Nov 03 23:21:38 GMT-500 2015
pagecounts-20151102-010000.gz	Standard	74 MB	Tue Nov 03 23:21:38 GMT-500 2015
pagecounts-20151102-020000.gz	Standard	74.2 MB	Tue Nov 03 23:21:38 GMT-500 2015
pagecounts-20151102-030000.gz	Standard	74.6 MB	Tue Nov 03 23:21:38 GMT-500 2015
pagecounts-20151102-040000.gz	Standard	73.7 MB	Tue Nov 03 23:21:38 GMT-500 2015
pagecounts-20151102-050000.gz	Standard	72.7 MB	Tue Nov 03 23:21:38 GMT-500 2015
pagecounts-20151102-060000.gz	Standard	73.1 MB	Tue Nov 03 23:21:38 GMT-500 2015
pagecounts-20151102-070000.gz	Standard	75.5 MB	Tue Nov 03 23:21:38 GMT-500 2015
pagecounts-20151102-080000.gz	Standard	78.2 MB	Tue Nov 03 23:21:38 GMT-500 2015
pagecounts-20151102-090000.gz	Standard	83.9 MB	Tue Nov 03 23:21:38 GMT-500 2015
pagecounts-20151102-100000.gz	Standard	88.7 MB	Tue Nov 03 23:21:39 GMT-500 2015
pagecounts-20151102-110000.gz	Standard	92 MB	Tue Nov 03 23:21:39 GMT-500 2015
pagecounts-20151102-120000.gz	Standard	89.5 MB	Tue Nov 03 23:21:39 GMT-500 2015
pagecounts-20151102-130000.gz	Standard	80.9 MB	Tue Nov 03 23:21:39 GMT-500 2015
pagecounts-20151102-140000.gz	Standard	84.5 MB	Tue Nov 03 23:21:39 GMT-500 2015
pagecounts-20151102-150000.gz	Standard	87.4 MB	Tue Nov 03 23:21:39 GMT-500 2015
pagecounts-20151102-160000.gz	Standard	91.4 MB	Tue Nov 03 23:21:39 GMT-500 2015
pagecounts-20151102-170000.gz	Standard	91.5 MB	Tue Nov 03 23:21:39 GMT-500 2015

Data Upload for PageCount files using Java AWS API

3.3. Map-Reduce job

We have created a Map reduce job that takes Wikipedia files uploaded on S3 bucket as input and generates key value pairs, which contain page names and their hit counts over a day. This map reduce job consists of 2 jobs chained together. The first job, collects all the page names from the wiki files and associate their hit counts with them. But the output we get is sorted according to the page names instead of the count. So the output from the first job is fed to the second job. The second Map Reduce job sorts all the records in descending order of page hits. Hence it is now easier to get top 'n' trending pages by reading first n records from the output file generated from second job.

3.3.1. Job 1

Input: Wiki Files for page counts over a day.

File 1: pagecounts-20151102-000000	File 2: pagecounts-20151102-010000
en Christmas 11 1478 en Black_Hole 121 1347 en Barack_Obama 21 500	en Christmas 7 1478 en Black_Hole 12 1347 en Barack_Obama 36 500



Mapper



Christmas 11
Black_Hole 121
Barack_Obama 21
Christmas 7
Black_Hole 12
Barack_Obama 36

Reducer



Barack_Obama 57
Black_Hole 133
Christmas 17

Mapper Class Code :

```
public class WikiTrendsMapper extends Mapper<LongWritable, Text, Text, LongWritable>{

    @Override
    protected void map(LongWritable key, Text value,
                       Mapper<LongWritable, Text, Text, LongWritable>.Context context)
        throws IOException, InterruptedException {

        String recordArray[] = new String [ ] {};
        try {
            if(value.toString().startsWith("en")){
                recordArray = value.toString().split(" ");

                if(recordArray.length == 4) {
                    long pageHits = Long.parseLong(recordArray[2]);
                    context.write(new Text(recordArray[1]), new LongWritable(pageHits));
                }
            }
        } catch(Exception e) {
            System.out.println(value + "at record" + key.toString());
            System.out.println(recordArray[1] + "" + recordArray[1] + "" +
recordArray[2] + "" + recordArray[2]);
            throw e;
        }
    }
}
```

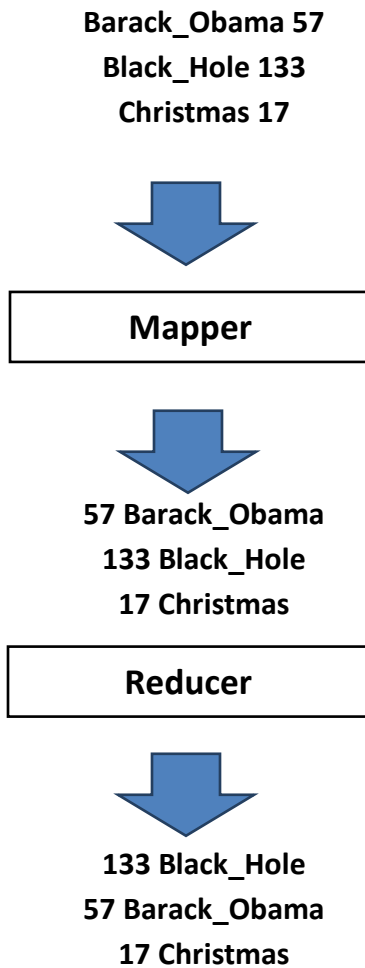
Reducer Class Code :

```
public class WikiTrendsReducer extends Reducer<Text, LongWritable, Text, LongWritable> {
    @Override
    protected void reduce(Text key, Iterable<LongWritable> values,
                          Reducer<Text, LongWritable, Text, LongWritable>.Context context)
        throws IOException, InterruptedException {
        Long totalHits = 0l;
        for(LongWritable value : values) {
            totalHits += value.get();
        }
        context.write(key, new LongWritable(totalHits));
    }
}
```

3.3.2. Job 2

This Map Reduce job sorts the results we get from first job, and gives us the sorted output. The default sort provided by Hadoop is for increasing order. But that can be changed by setting `SortComparatorClass` to `DecreasingComparator.class`

Input: Output files from job 1



Sorting Mapper Class Code :

```
public class SortingMapper extends Mapper<LongWritable, Text, LongWritable, Text> {

    @Override
    protected void map(LongWritable key, Text value,
        Mapper<LongWritable, Text, LongWritable, Text>.Context context)
        throws IOException, InterruptedException {

        try {
            System.out.println(value);
            String inputArray[] = value.toString().split("\\s+");
            if(inputArray.length == 2) {
                context.write(new LongWritable(Long.parseLong(inputArray[1])),
                    new Text(inputArray[0]));
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Sorting Reducer Class Code :

```
public class SortingReducer extends Reducer<LongWritable, Text, LongWritable, Text> {

    @Override
    protected void reduce(LongWritable key, Iterable<Text> values,
        Reducer<LongWritable, Text, LongWritable, Text>.Context context)
        throws IOException, InterruptedException {

        for(Text value: values) {
            context.write(key, value);
        }
    }
}
```

In this project we have used EMR service to run map reduce job. So a JAR file was created for the above code to act as an input to EMR job cluster.

4. Running job in Elastic MapReduce

EMR Cluster

In order to run Hadoop jobs, AWS provides Elastic Map Reduce (EMR) service. EMR uses Amazon EC2 instances to run managed Hadoop framework in an easy, fast and cost-effective way.

In order to utilize EMR service, the user needs to first create a cluster. This cluster actually creates an EC2 instance on which various services and frameworks can be added using bootstrap actions or steps. AWS provides an open source repository of bootstrap actions and related steps that can be added to configure already existing services. For example in our project we add bootstrap actions to configure Elastic Search, Kibana and Logstash on the EC2 instance.

The EMR clusters can be created and configured either by using GUI provided by AWS dashboard or by using AWS SDKs for various languages. We have decided to use AWS SDK for java in our project. The code for setting up and running EMR cluster involves following main steps:

- 1.) Setting up instance configurations for EC2 instance: This involves various parameter configurations like Instance Count for master and slave instances, Instance Types, actions to be taken if a step or boot strap action fails etc.**

The code below uses M1 Medium instances, with one instance each for master and slave. And it keeps the cluster alive after steps have completed or failed.

```
JobFlowInstancesConfig instanceConfig = new JobFlowInstancesConfig()
    .withEc2KeyName(keypair)
    .withInstanceCount(2)
    .withMasterInstanceType(InstanceType.M1Medium.toString())
    .withSlaveInstanceType(InstanceType.M1Medium.toString())
    .withKeepJobFlowAliveWhenNoSteps(true);
```

- 2.) Adding bootstrap actions : Here we have added bootstrap actions to set Elastic Search, Kibana and Logstash. Code snippet shows adding of Elastic Search to EMR cluster.**

```
//Bootstrap Actions add elastic search
String elasticSearchPath =
"s3://support.elasticmapreduce/bootstrapactions/other/elasticsearch_install.rb";
ScriptBootstrapActionConfig bootstrapScriptConfig = new
ScriptBootstrapActionConfig();
bootstrapScriptConfig.setPath(elasticSearchPath);
BootstrapActionConfig elasticSearchBootstrap = new BootstrapActionConfig();
elasticSearchBootstrap.setName("Install ElasticSearch");
elasticSearchBootstrap.setScriptBootstrapAction(bootstrapScriptConfig);
```

- 3.) **Adding steps:** The steps are added in the EMR cluster to enable debugging for the cluster and also for running Hadoop job. The jar file created in the previous step is used as an input for the Hadoop Job.

```
// Enable debugging
StepFactory stepFactory = new StepFactory();

StepConfig enabledebugging = new StepConfig()
    .withName("Enable debugging")
    .withActionOnFailure("TERMINATE_JOB_FLOW")
    .withHadoopJarStep(stepFactory.newEnableDebuggingStep());

//Hadoop Job Step
HadoopJarStepConfig hadoopStep = new
HadoopJarStepConfig().withJar("s3://wikitrendstracking/jar/WikiTrends.jar")
    .withArgs("s3://wikitrendstracking/input", //Input Path
        "s3://wikitrendstracking/output1_mid", //Output path for first job
        "s3://wikitrendstracking/output1"); //Output path for 2nd job

StepConfig hadoopJob = new StepConfig()
    .withHadoopJarStep(hadoopStep)
    .withName("HadoopJob")
    .withActionOnFailure(ActionOnFailure.CONTINUE);
```

- 4.) **Creating a JobFlowRequest:** All the above parameters are added to a JobFlowRequest object provided by AWS SDK. This request is then made to run by using EMR Client. (As shown in the code below)

```
RunJobFlowRequest runJobFlowRequest = new RunJobFlowRequest()
    .withAmiVersion("3.5.0")
    .withBootstrapActions(elasticSearchBootstrap, //adding Bootstrap actions
        logstashBootstrap,
        kibanaBootstrap)
    .withName("caprojecttest") //cluster name
    .withSteps(enabledebugging, hadoopJob) // adding steps
    .withLogUri("s3://aws-logs-393991342318-us-east-1/elasticmapreduce/")
    .withJobFlowRole("EMR_EC2_DefaultRole")
    .withServiceRole("EMR_DefaultRole")
    .withInstances(instanceConfig);

RunJobFlowResult runJobFlowResult = eMRClient.runJobFlow(runJobFlowRequest);
```

[Create cluster](#)
[View details](#)
[Clone](#)
[Terminate](#)

Filter: All clusters 10 clusters (all loaded)

	Name	ID	Status	Creation time (UTC-5)	Elapsed time	Normalized instance hours
	caprojecttest	j-26MGOF5YFEGPS	Starting	2015-12-11 01:02 (UTC-5)	24 seconds	0

Summary

Master public DNS: --

Termination protection: Off [Change](#)

Tags: -- [View All / Edit](#)

Hardware

Master: Provisioning 1 m1.medium

Core: Provisioning 1 m1.medium

Task: --

[View cluster details](#) [View monitoring details](#)

Steps
[Add Step](#) [View all interactive jobs](#)

Name	Status	Start time (UTC-5)	Elapsed time
HadoopJob	Pending		
Enable debugging	Pending		

Bootstrap Actions

Name
Install ElasticSearch
Install logstash
Install kibana ginx

After this code is successfully run, the 'caprojecttest' cluster is created and status is set to 'Starting'. It takes around 10 mins to start the cluster and execute the steps. We can see the bootstrap actions and steps added into the cluster. Current status for the steps is 'Pending', as they are waiting for cluster to start and execute the bootstrap actions. On the left side of image we can see the instance types for master and core instances that will run.

Eventually the cluster starts and Hadoop Job starts running. The time taken by Hadoop job depends on the size of instance we took. This is discussed in detail in further section. After the Hadoop Job is completed the output data is stored in S3.

[Upload](#)
[Create Folder](#)
[Actions](#)

None
Properties
Transfers

All Buckets / [wikitrendstracking](#) / [output1](#)

Name	Storage Class	Size	Last Modified
TopTrends.txt	Standard	1.1 KB	Tue Dec 08 01:23:12 GMT-500 2015
_SUCCESS	Standard	0 bytes	Tue Dec 08 01:06:33 GMT-500 2015
part-r-00000	Standard	689 MB	Tue Dec 08 01:03:19 GMT-500 2015

Above image for a previous run, shows the final output getting stored in bucket. The data from here is further read by a java job which gets us the TopTrends from the file (output of which was stored in TopTrends.txt).

5. Visualizing Top 10 Trending Topics using Kibana

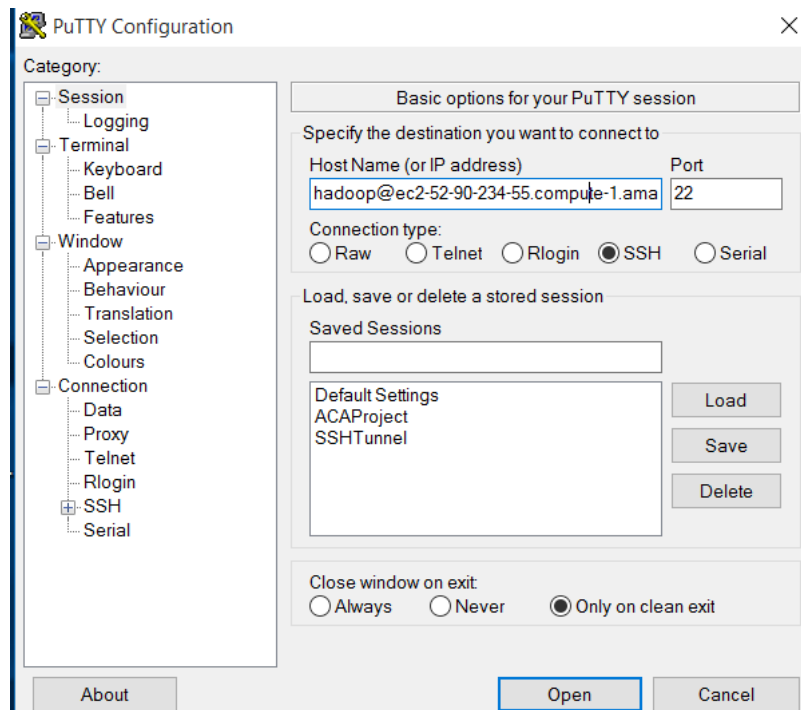
5.1 Settings to Open Kibana in Web Browser on Windows

When we want to connect to EC2 instance, which is a virtual server running in the cloud, we have to use SSH with AWS. After we make a connection, the terminal on our local computer behaves as if it is running on the remote computer.

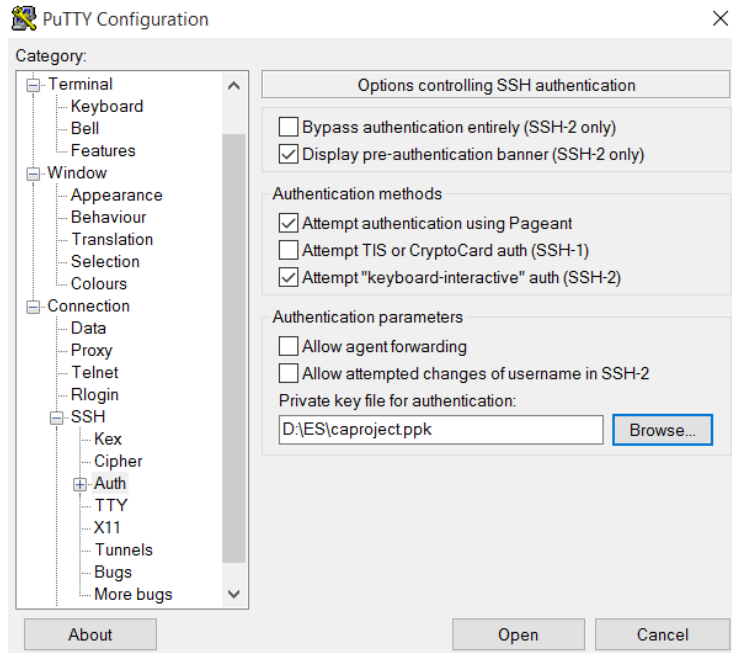
We have perform following steps on windows to use an SSH client such as PuTTY to create an SSH tunnel to the master node.

1) In the Host Name field of PuTTY, typed `hadoop@MasterPublicDNS`.

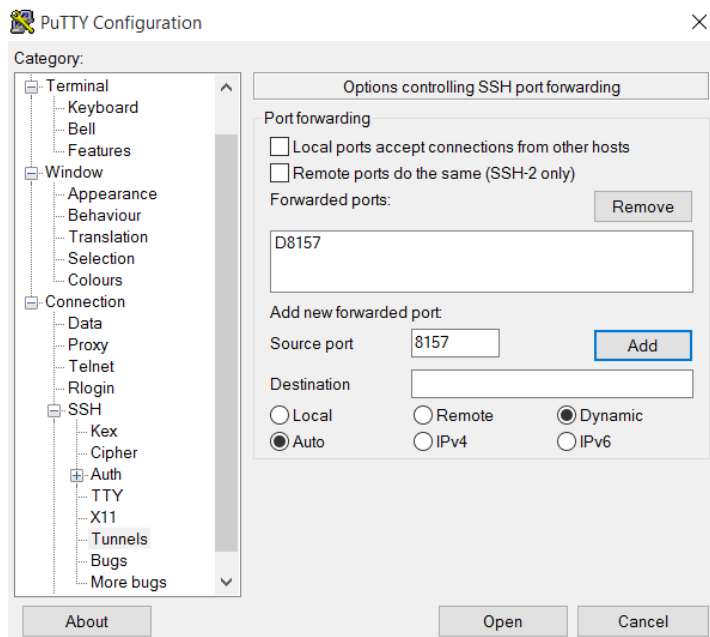
For example: `hadoop@ec2-###-##-###.compute-1.amazonaws.com`.



2) In the Category list, expand Connection > SSH, and then choose Auth. For Private key file for authentication, choose Browse and select the .ppk file that you generated using PuTTY Gen.



3) In the Category list, expand Connection > SSH, and then choose Tunnels. In the Source port field, typed 8157 (an unused local port). Selected the Dynamic and Auto options. Choose Add and Open.



4) Now you can configure you aws by typing aws configure command as shown in screenshot.

```
hadoop@ip-172-31-8-179:~
Using username "hadoop".
Authenticating with public key "imported-openssh-key"
Last login: Mon Dec 7 07:15:21 2015

 _ _ | _ _ | _ _ |
 _ | ( _ | /
 _ | \ _ | _ _ |

Amazon Linux AMI

https://aws.amazon.com/amazon-linux-ami/2014.09-release-notes/
9 package(s) needed for security, out of 27 available
Run "sudo yum update" to apply all updates.
Amazon Linux version 2015.09 is available.

-----

Welcome to Amazon Elastic MapReduce running Hadoop and Amazon Linux.

Hadoop is installed in /home/hadoop. Log files are in /mnt/var/log/hadoop. Check
/mnt/var/log/hadoop/steps for diagnosing step failures.

The Hadoop UI can be accessed via the following commands:

ResourceManager      lynx http://ip-172-31-8-179.ec2.internal:9026/
NameNode              lynx http://ip-172-31-8-179.ec2.internal:9101/

-----

[hadoop@ip-172-31-8-179 ~]$ aws configure
AWS Access Key ID [*****4PNA]: AKIAJ4AWGHX63SUD4PNA
AWS Secret Access Key [*****kp/t]: vrtQ1bP2ZAQYUWnqBOYt7mEtFAzqZZScvg
wykp/t
Default region name [us-east-1]: us-east-1
Default output format [None]:
[hadoop@ip-172-31-8-179 ~]$ rm -rf TopTrends.txt
[hadoop@ip-172-31-8-179 ~]$ wget http://wikitrendstracking.s3.amazonaws.com/outp
```

5) Get Top 10 Trending Topics file from S3 by using following command :

`wget http://wikitrendstracking.s3.amazonaws.com/output/TopTrends.txt`

You can post this file to Elastic Search service by using following command :

`curl -XPOST "http://localhost:9200/_bulk" -data-binary "@TopTrends.txt"`

```
[hadoop@ip-172-31-8-179 ~]$ wget http://wikitrendstracking.s3.amazonaws.com/outp
ut/TopTrends.txt
--2015-12-07 07:43:46-- http://wikitrendstracking.s3.amazonaws.com/output/TopTr
ends.txt
Resolving wikipediasttracking.s3.amazonaws.com (wikipediasttracking.s3.amazonaws.c
om)... 54.231.65.49
Connecting to wikipediasttracking.s3.amazonaws.com (wikipediasttracking.s3.amazona
ws.com)|54.231.65.49|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1173 (1.1K) [text/plain]
Saving to: 'TopTrends.txt'

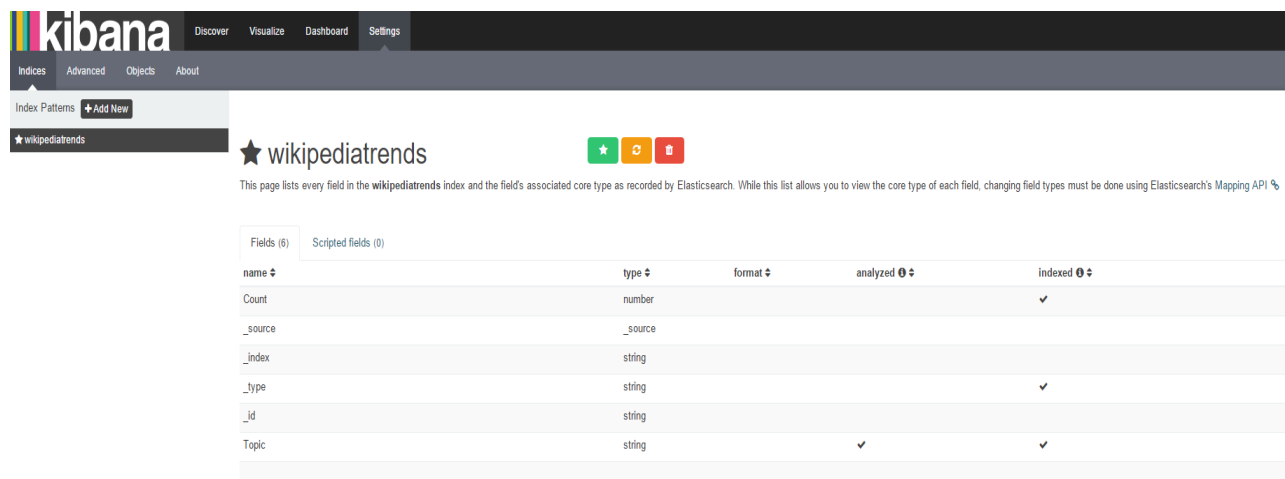
TopTrends.txt      100%[=====>]      1.15K  --.-KB/s    in 0s

2015-12-07 07:43:46 (83.9 MB/s) - 'TopTrends.txt' saved [1173/1173]

[hadoop@ip-172-31-8-179 ~]$ curl -XPOST "http://localhost:9200/_bulk" --data-bin
ary "@TopTrends.txt"
{"took":434,"errors":false,"items":[{"_index":"wikipediastrends","_type":
"wikipediadata","_id":"AVF7ZY7du6PelgMq7lmR","_version":1,"status":201}},{_cre
ate":{"_index":"wikipediastrends","_type":"wikipediadata","_id":"AVF7ZY7du6PelgMq
7lms","_version":1,"status":201}},{_create":{"_index":"wikipediastrends","_type":
"wikipediadata","_id":"AVF7ZY7du6PelgMq7lmT","_version":1,"status":201}},{_creat
e":{"_index":"wikipediastrends","_type":"wikipediadata","_id":"AVF7ZY7du6PelgMq7l
mU","_version":1,"status":201}},{_create":{"_index":"wikipediastrends","_type":"w
ikipediadata","_id":"AVF7ZY7du6PelgMq7lmV","_version":1,"status":201}},{_create":
{"_index":"wikipediastrends","_type":"wikipediadata","_id":"AVF7ZY7du6PelgMq7lmW
","_version":1,"status":201}},{_create":{"_index":"wikipediastrends","_type":"wikip
ediadata","_id":"AVF7ZY7du6PelgMq7lmX","_version":1,"status":201}},{_create":{"
_index":"wikipediastrends","_type":"wikipediadata","_id":"AVF7ZY7du6PelgMq7lmY",
"_version":1,"status":201}},{_create":{"_index":"wikipediastrends","_type":"wikip
ediadata","_id":"AVF7ZY7du6PelgMq7lmZ","_version":1,"status":201}},{_create":{"_
index":"wikipediastrends","_type":"wikipediadata","_id":"AVF7ZY7du6PelgMq7lma","_
```

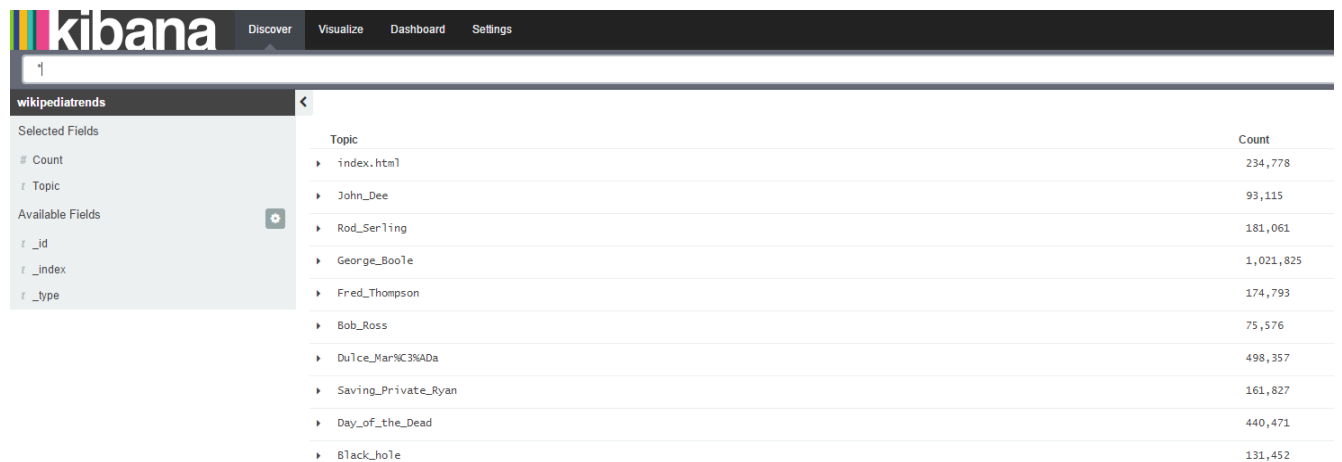
5.2 Visualize Data on Kibana

Since bootstrap action configures Kibana to listen at port 9200 on the master node, you can point the browser to the Master Node DNS public address (example: <http://ec2-54-77-165-138.eu-west-1.compute.amazonaws.com>) and get the Kibana console running. We need to specify the name of the index which we want to use for our visualizations. We have entered the name of the index that was specified in the final output of the top 10 trends "wikipediatrends". Now create the index pattern and choose "Visualize" from the top menu to create a new visualization.



Index pattern "wikipediatrends" added to Kibana

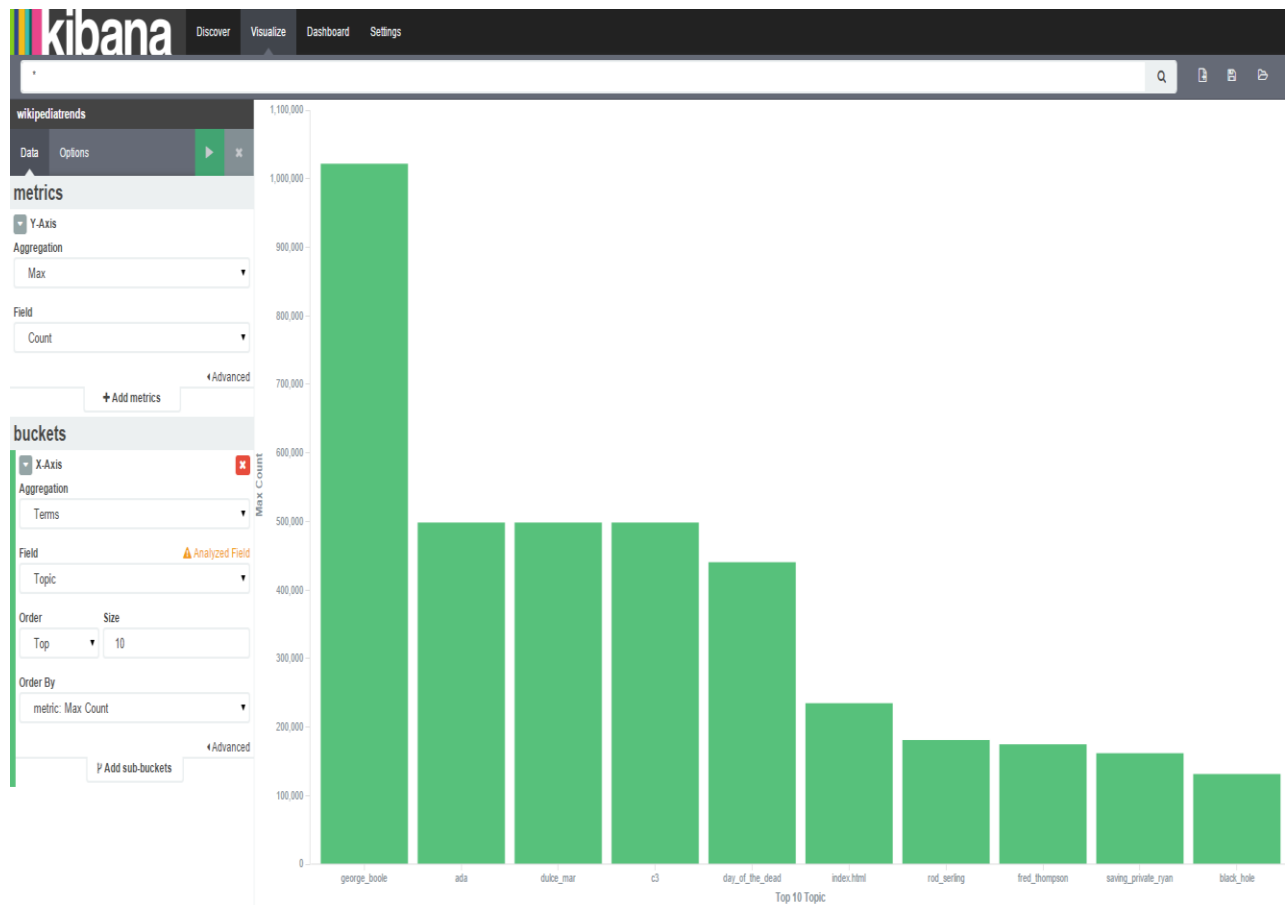
Discover page on Kibana allows us to explore our data with selected index. We will use our predefined index "wikipediatrends" to get our data. We can submit search queries and also filter the search results.



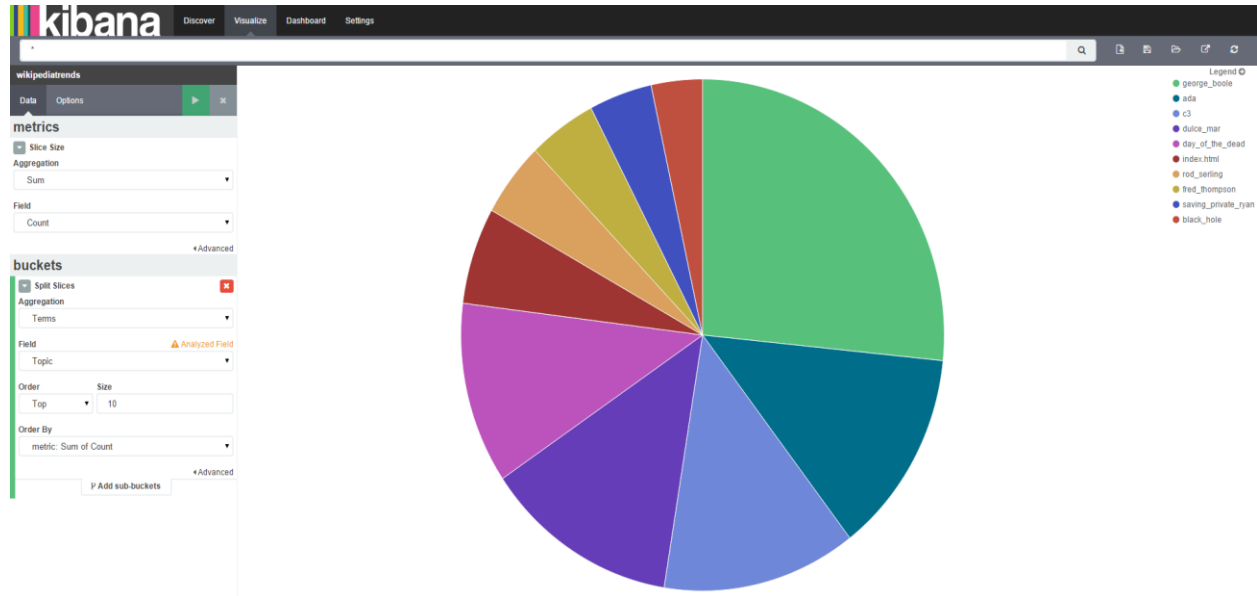
Top 10 Wiki Data fetched according to index "wikipediatrends"

You can use the Visualize page to design data visualizations. We have choose multiple virtualization types to display our data.

- 1) Vertical Bar Chart
- 2) Pie Chart
- 3) Metric Visualization in Ascending Order
- 4) Line Chart
- 5) Area Chart



Top trending topics visualization using Vertical Bar Chart

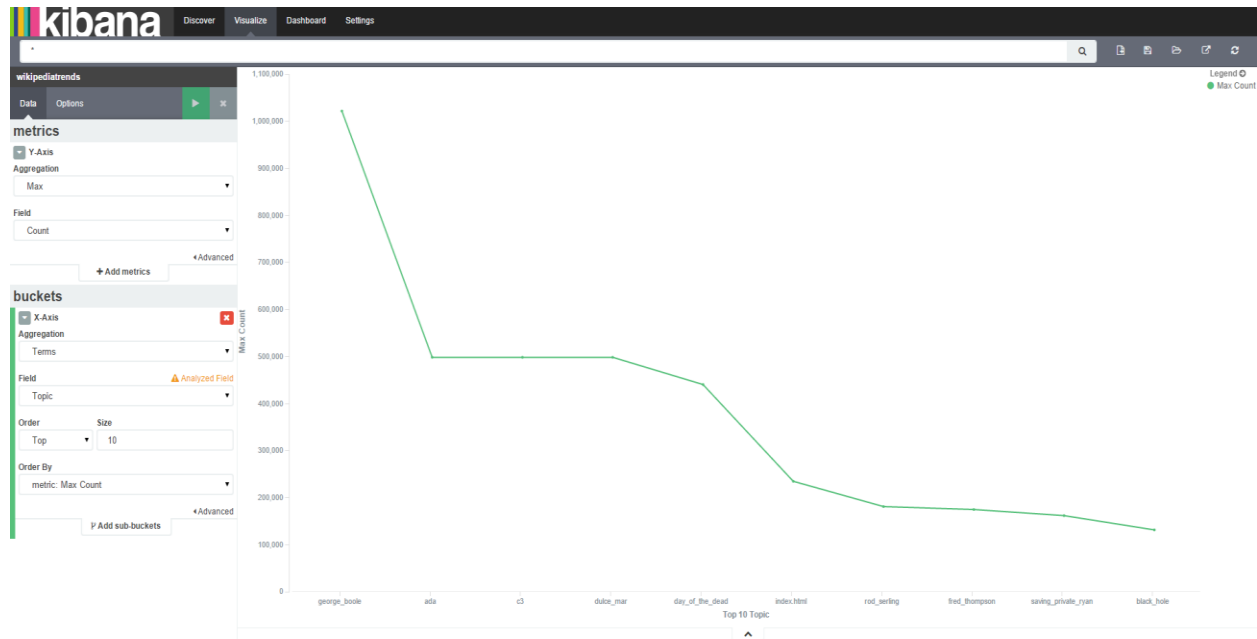


Top trending topics visualization using Pie Chart

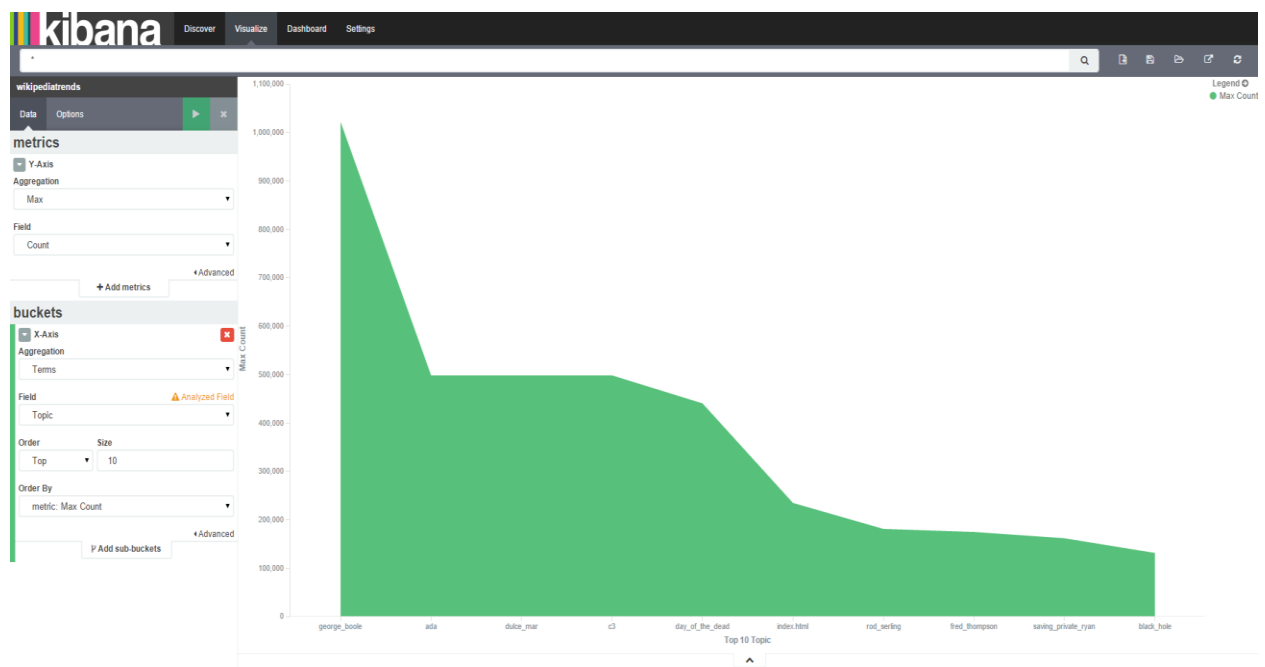
Top 10 Topic	Max Count
george_boole	1,021,825
ada	498,357
dulce_mar	498,357
c3	498,357
day_of_the_dead	440,471
index.html	234,778
rod_serling	181,061
fred_thompson	174,793
saving_private_ryan	161,827
black_hole	131,452

Export: [Raw](#) [Formatted](#)

Top trending topics visualization using Metrics "Max Count" (in Ascending Order)



Top trending topics visualization using Line Chart



Top trending topics visualization using Area Chart

6. Conclusion

We have developed the application that gives up top 10 trending topics from Wikipedia. Throughout the development of this project we learned a lot about multiple services that are provided by AWS. We learned how to use S3 service which provides cost-effective object storage for a wide variety of use cases including cloud applications, backup and archiving and big data analytics. We have also used EMR service which simplifies big data processing by providing Hadoop framework that makes it easy, fast, and cost-effective for us to process vast amounts of your data across dynamically scalable Amazon EC2 instances. We have also took benefits of Elasticsearch and Kibana which are data manipulation and data virtualization platforms that allow us to interact with data with stunning powerful graphics.

We have also compared the performance difference by running the same application on local machine and Amazon web services. The outcome is very interesting & inspiring, local machine takes long time to run this application while the same application on Amazon web services takes less amount of time. The performance difference of both is listed below in table.

Platform	Instance Type	Execution Time
Local Machine	<i>Technical Details:</i> Memory : 8 GiB Processor : 2.4 GHz Core i5	21 Minutes
Amazon Web Services	m1.medium <i>Technical Details:</i> vCPU : 1 Memory :3.75 GiB Instance Storage : 1 x 410 GB Network Operation : Mod (M1 instance is considered as previous gen instances. It does not provides SSD storage)	43 Minutes
Amazon Web Services	m3.xLarge <i>Technical Details:</i> vCPU : 4 Memory :15 GiB Instance Storage : 2x40GB SSD Network Operation : High (M3 instance offer SSD backed instance storage that delivers higher I/O performance.)	6 Minutes

7. References

- <http://docs.aws.amazon.com/gettingstarted/latest/emr/getting-started-emr-sentiment-tutorial.html>
- <http://blog.cloudera.com/blog/2009/07/tracking-trends-with-hadoop-and-hive-on-ec2>
- <https://aws.amazon.com/ec2/instance-types/>
- <https://aws.amazon.com/elasticsearch-service/>
- <http://aws.amazon.com/s3/>
- <https://aws.amazon.com/elasticmapreduce/>
- <https://www.elastic.co/guide/en/kibana/current/setup.html>