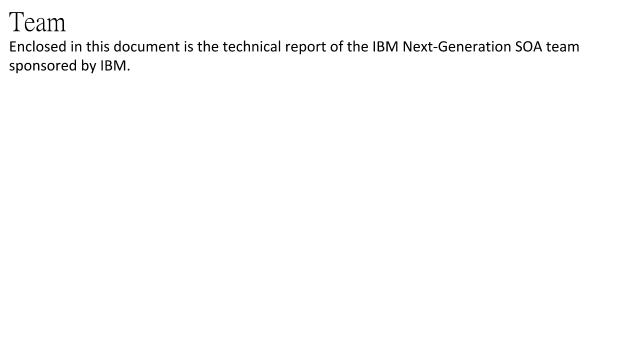
Analysis Me





# Table of Contents

1. **Introduction**
2. **Motivation**
3. **Related work**
4. **System design**
5. **System implementation**
6. **Experiments and analysis**
7. **Conclusions and future work**
8. **Contribution of each team member**

## **Introducti**on

SOA (Service Oriented Architecture) has been widely adopted as a architecture to address the requirement of modern organizations. In the current economic climate where new IT technics are blooming, the flexibility of SOA makes it possible to deal with the quick emerging and evolving of all the new technics and enterprises to keep pace with new trend.

The Next Generation SOA propose several aspects.

* Cloud Computing: how to realize quick connection of internal apps and external cloud apps;
* Business Mobility: how to extend applications to mobile devices;
* Internet of Things: how to link to IOT and make in-time reaction;
* API: how to publish API and connect enterprises to Sociotechnical system which is growing into a global trend;
* Big Data: how to provide services that meet the requirement for scale and quality of big data.

BlueMix is an open-standards, cloud-based platform for building, managing, and running apps of all types, such as web, mobile, big data, and smart devices. Capabilities include Java, mobile back-end development, and application monitoring, as well as features from ecosystem partners and open source—all provided as-a-service in the cloud.

Our project leverages IBM BlueMix as a future next generation SOA platform for developing fast-to-market web applications.

## **Motivation**

The traditional IT development cycle is too slow and expensive, which requires hardware, and monolithic chunk of softwares. You need database, cloud instances, deployment, etc. Although now we have infrastructure as a service, platform as a service, software as a service, back-end or mobile back-end as service, we want to rapidly create powerful applications with the efficiency of a cold-blooded cyborg, and even starters can kick off quickly.

Now IBM BlueMix has provided this future next generation SOA platform, and our project aims at serving as a killer application to advocate this new technology.

## **Related work**

Our project is based on the previous project from another team. As the foundation of our project and the first concept we tried to understand, the previous project did a good job in system and module design.

This previous project’s research goal is to explore a way to advance the traditional service oriented architecture (SOA) to the next level, cloud-powered SOA. As a proof of concept, we have designed and developed Enterprise Risk Advisor (ERA) - a Web 2.0 application that provides web services to analyze, visualize and predict company’s financial situation. ERA applies big data analytics technology to extract and analyze the “Risk factors” for US publicly traded companies, crawlable from 10k forms.

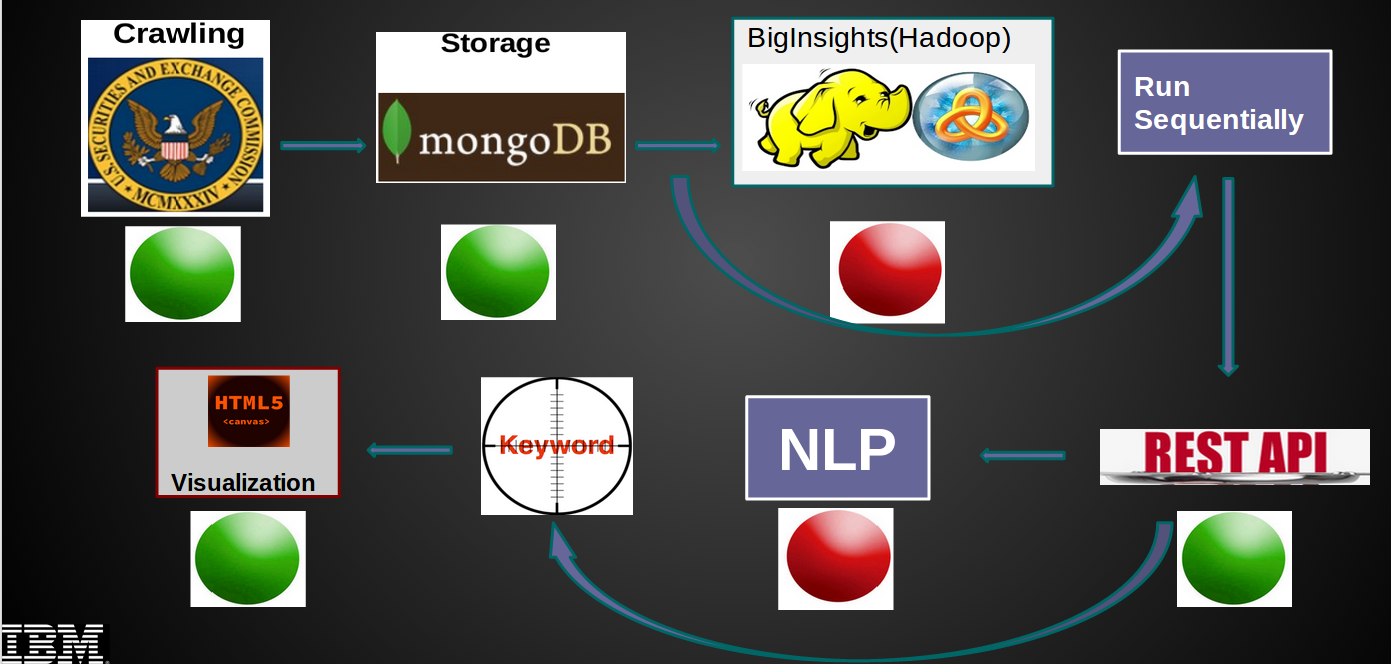


Figure 3.1: The road map for the previous project.

The above image shows the road map of the previous project, the arrow means they went from a development stage to the next. There are two main workarounds helping them move forward. The following table to explain the original plans and why we adopted the workarounds.

|  |  |  |
| --- | --- | --- |
| Original Plan | Workaround | Reason |
| BigInsights - The Hadoop Service in Bluemix | Run sequentially | IBM BigInsights is not available in IBM Bluemix |
| Natural Language Processing Tools | Use keywords and phrases extraction | NLP is not suitable for this application |

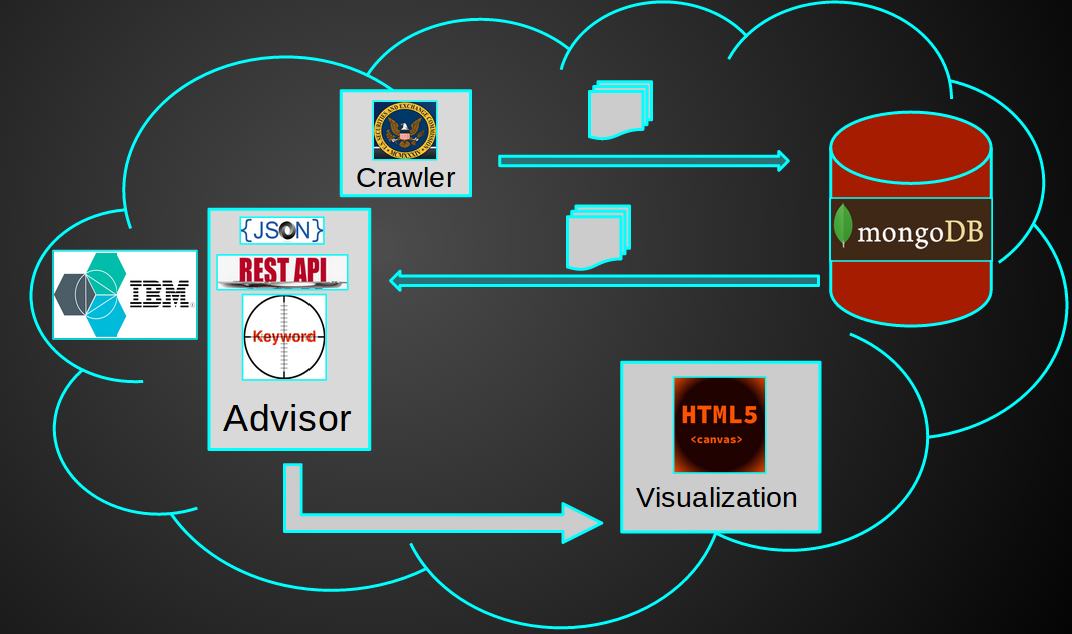


Figure 3.2: The system architecture of the previous project.

The above image displays the system architecture, there are several modules in the system:

1. Knowledge crawler: use crawler to download 10-k documents of all public companies
2. Knowledge repository: downloaded data is stored in the repository
3. Advisor: defined 25 types of financial risks and the keyword set for each type of risk. Extract keywords from the “Risk Factor” sections and find risks for the companies.
4. Web APIs & Visualization: Build a website using Java EE and provide web APIs to other developers. Visualize comparison of the risk factors by year and for different companies.

There are three layers in the system: the data layer, the analytical layer and the presentation layer. They adopt MVC architecture, M is the data layer, V is the presentation layer, C is the analytical layer. The following picture show the layers in the system.

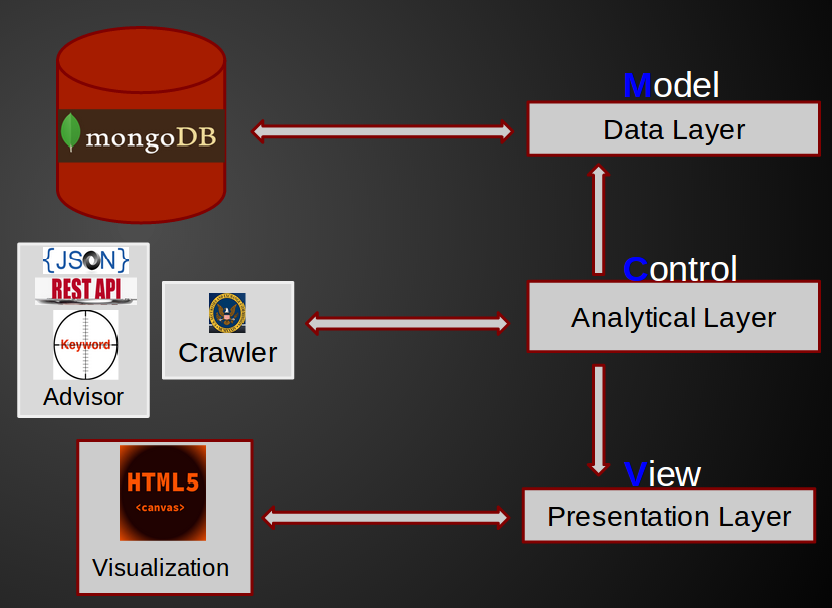


Figure 3.3: The MVC layer structure of the previous project.

Due to the technical problem during the previous project, they failed to use IBM BigInsights to do the risk analysis, and they only integrated very few IBM Bluemix service with their project. So these two parts becomes our main goal.

## System design

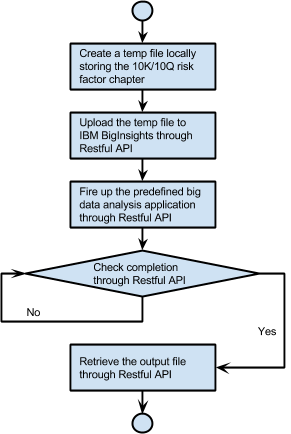
The main goal of this project is to modify and to extend the previous project to make it more integrated with IBM Bluemix. So our design includes two parts. The first part is to replace the original risk analysis algorithm by using IBM BigInsights, so except the web app system control, all the other components are based on IBM Bluemix. The second part is to add more related services from IBM Bluemix.

### Risk analysis & IBM BigInsights

The risk analysis algorithm is that, we have a certain company’s financial report as the input file, and a list of keywords which may indicate good/bad financial performance, then we compare the input file with the list and summarize the statistic of the company based on the keyword counts.

In the previous project, the team wrote their own code to implement this algorithm. After several trials, we found that IBM provides a service called BigInsights which fits our need for risk analysis. Here is the detailed procedure about how we replace the previous algorithm with IBM BigInsights.

The previous algorithm directly process the file inside the application. However, the purpose of IBM proposing Biginsights service is mainly on processing big data, which fit our needs. Therefore we changed our implementation to IBM Biginsights instead of implementing our own algorithms. Text Analytics application framework was utilized to develop our application for text processing in Biginsights. It uses AQL (Annotational Query Language) for information extraction. We adopted regular expression for text matching and processing. The whole Biginsights application was wrapped into a web service such that it can be directly called by our main application for use. Some Biginsights file system management and application running services were wrapped together with the Text Analytics web service into a java class for fully automatic control.



From the result, we can see that the IBM BigInsights does the same job, and we don’t need the application instance to run the real job. It makes both the development and the operation easier.

### Add-on IBM Bluemix services

Companies like Internet startup companies, will grow their IT resources rapidly during the beginning of their deployment. So in order to avoid the capital expense and make the web application more scalable, they need to increase and decrease the computation very quickly for responding related needs. And developers or administrators need to have the control of the whole system’s status and performance.

We found out that IBM Bluemix provides two very powerful services for these needs, one is Auto-Scaling and the other is Monitoring and Analytics.

For the Auto-Scaling, we can use that service to set up instance increase and decrease policies. Let’s see a screenshot of the Auto-Scaling control page.

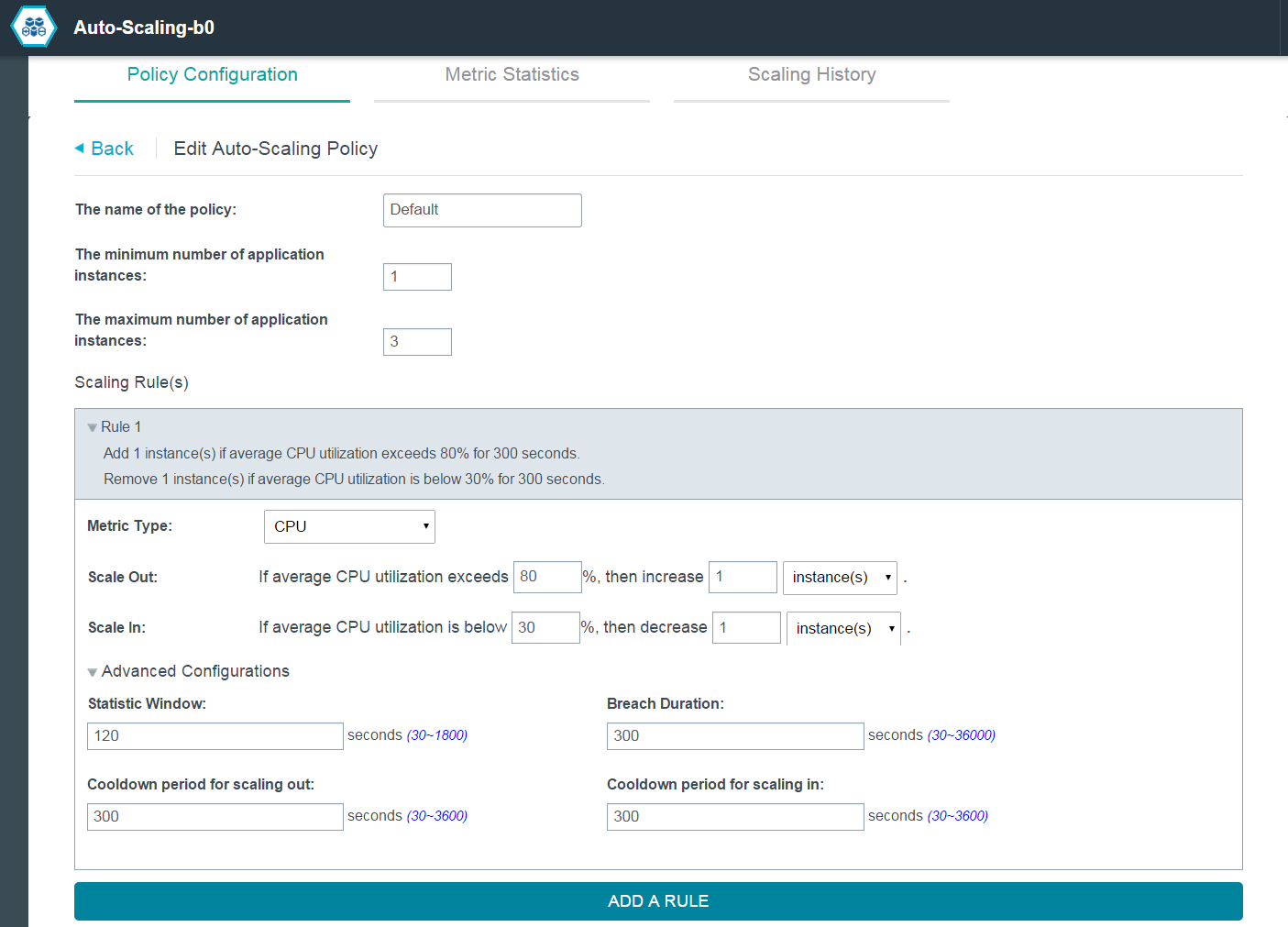


Figure 4.1: Auto-Scaling’s Policy Configuration Tab.

We can set up the minimum/maximum number of application instances. And there are four metric types for the scaling rules, CPU utilization, Memory utilization, JVM heap utilization(since we’re using Java as the backend language), and throughput.

After setting the metric type, we can set the threshold of the scale out/in rules. The normal rule is that the whole system should not stay in an idle state or low state too long or not enough to handle the current load.

If the developers need some advanced configuration, they can also set the related setting for trigue timing. For example, the Statistic Windows is the length of the time slot for calculating the average metric type, so the longer you set, the more precise statistics you’ll get, while the application will respond slowly to some fast change. And there are also Breach Duration and Cooldown period. What’s more, it also provides Metric Statistics for monitoring the real time performance of instances to help you get a better understanding of the current status, and Scaling History for logging the change of the instances.

Besides the Auto-Scaling, there are another service for administator, which is called Monitoring and Analytics. This service provides three functionalities.

The Availability tab shows the health of the whole system.

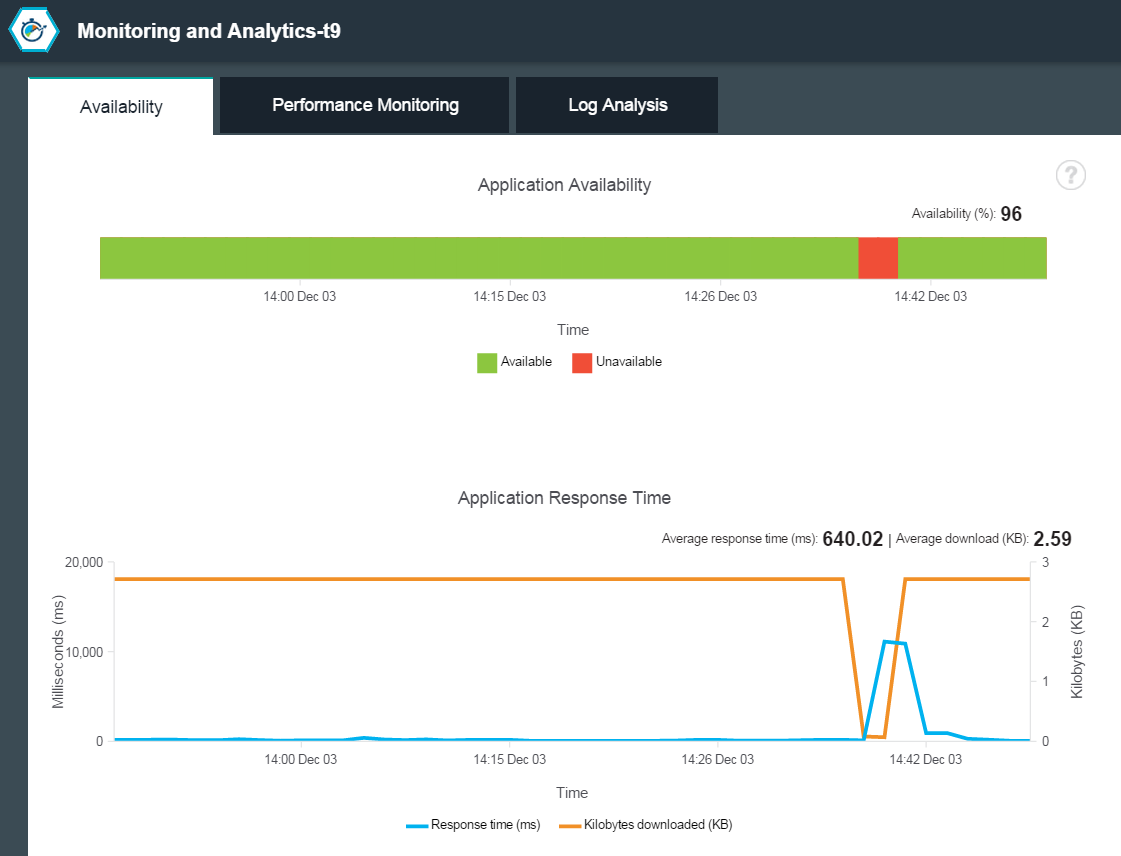


Figure 4.2: Monitoring and Analytics’s Availablity Tab.

THe Performance Monitoring tab helps administrator know the detailed statistic of the running instances.

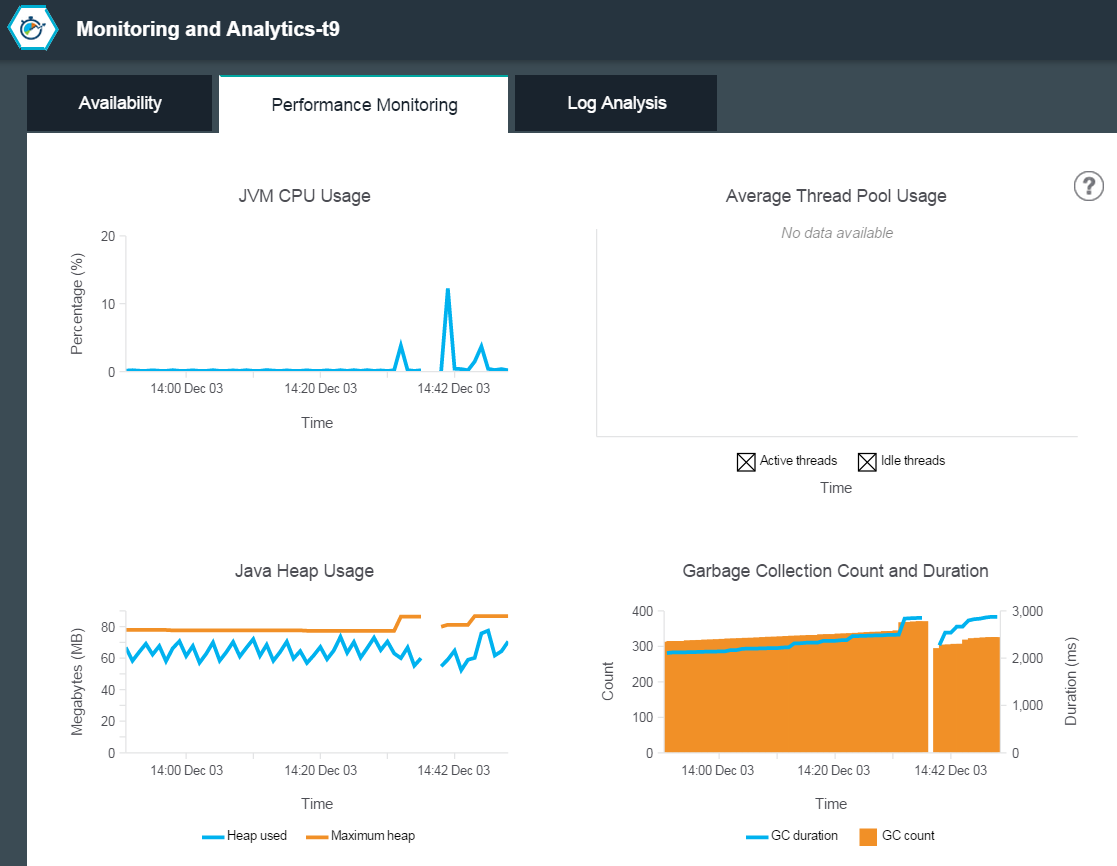


Figure 4.3: Monitoring and Analytics’s Performance Monitoring Tab.

And the Log Analysis collects specific logs generated during certain time. If something goes wrong, instead of checking logs instance by instance, the administrator has a centralized system for reviewing logs.

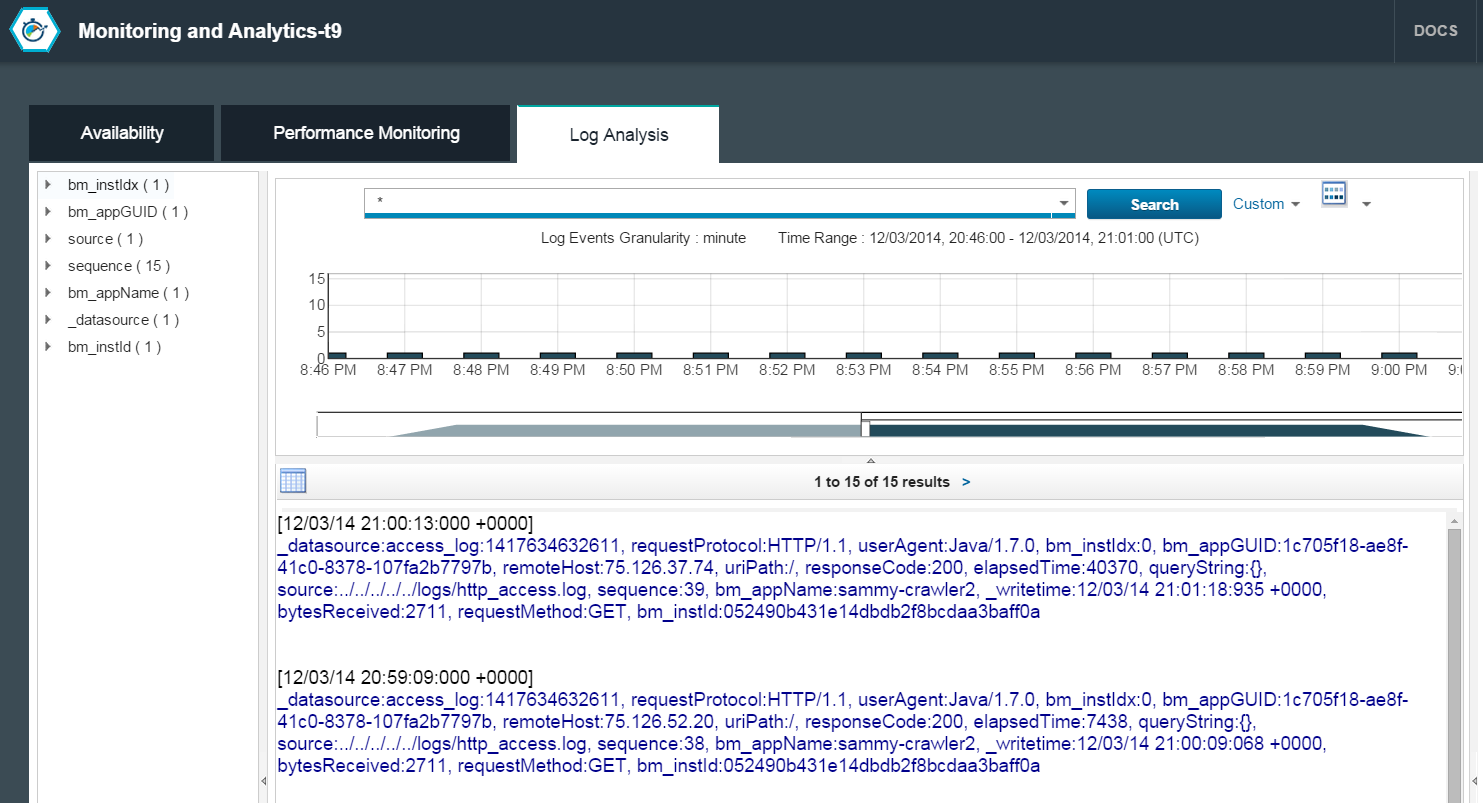


Figure 4.4: Monitoring and Analytics’s Log Analysis Tab.

## **System** Implementation

One of the goals of this project is to leverage IBM Bluemix as the platform for building the application. For that reason, we implemented the whole system exclusively on top of the IBM Bluemix platform.

In general, the system consists of four main parts: crawler, repository, advisor, and visualization. The crawler will retrieve the publicly available 10K and 10Q documents for a given company name from the US government website. It uses a web scraping technique to extract the textual part of the document for further processing by IBM BigInsights. The repository stores the crawled documents as well as the analysis result produced by IBM BigInsights. The advisor defines the 25 types of financial risks and the keyword set for each type of risk for text extraction. Finally, the visualization part displays a comparison of the risk factors by year and for different companies.

We implement the crawler using Java EE along with IBM BigInsights module. The part of the crawler which is written in Java will generate the list of URLs of the relevant 10K and 10Q documents. It generates the URLs based on the stock symbol of the desired company and the desired time range using a predefined pattern from the US government website. The retrieved documents which is in HTML format will be extracted to plain text which is more suitable for IBM BigInsights. The plain text document then is used as an input for IBM BigInsights analysis module for further text processing.

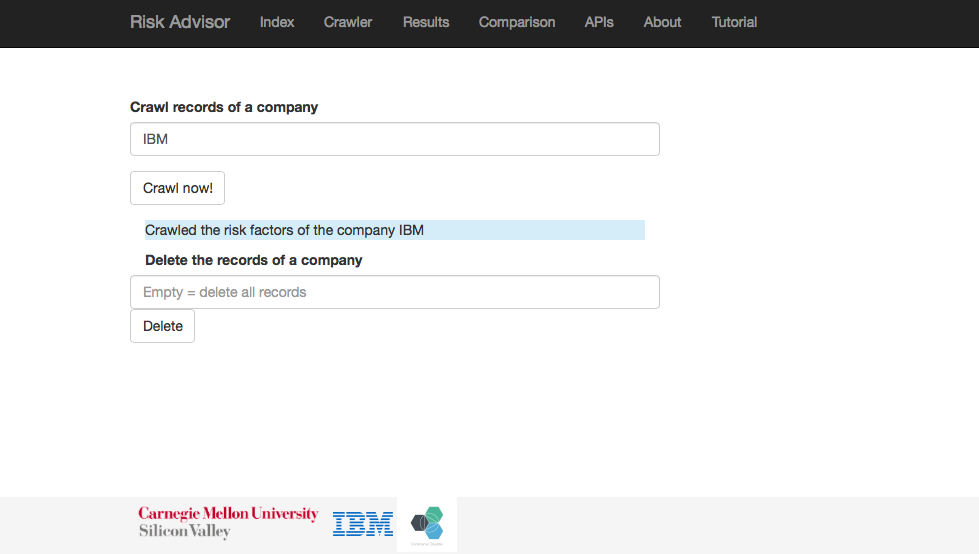


Figure 5.1: The application crawling for a company by its stock symbol.

The IBM BigInsights service module is used in the crawler to extract the relevant information from the plain text documents. It uses regular expressions (regex) which is expressed in an SQL-like query syntax. Currently, it uses the 25 keywords of financial risks defined in the advisor component to count the number of occurrence of each keywords. The whole IBM BigInsights operations are invoked from the crawler component via the provided REST API. The result then is retrieved by the crawler component from the IBM BigInsights module and stored in the repository component.

The repository component is actually an instance of MongoDB service on IBM Bluemix. It stores the plain text 10K or 10Q documents as well as the corresponding risk scores in JSON format. The MongoDB service has a binding support for Java programming language, more specifically for the Liberty for Java runtime environment on IBM Bluemix, which makes storing and retrieving data easier. The MongoDB service instance will not be scaled up or down, unlike the main Java application instance, so no data replication is required.

The advisor and the visualization component are programmed in Java and HTML5, respectively. They work together to present the analysis results to the user. The advisor component takes the stock symbol of the desired company and retrieves the corresponding records from the MongoDB service. As the analysis is done during the crawling activity, the advisor component can directly pass the records to the visualization component on the frontend. The visualization component takes the records as the input and displays the risk categories in form of histogram and pie chart.

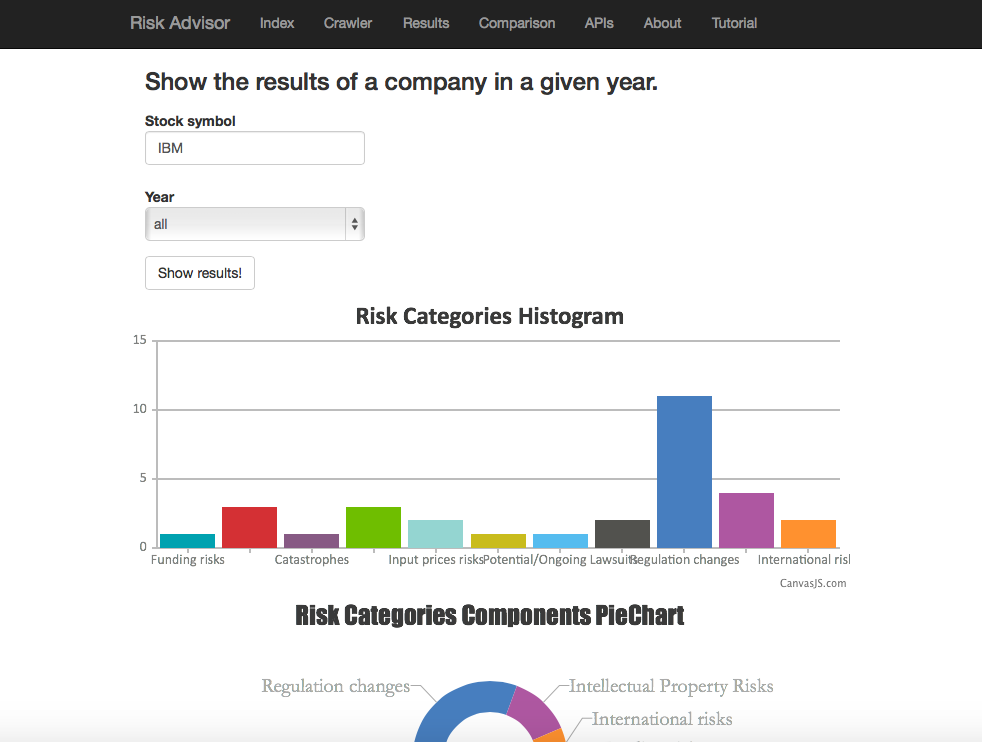


Figure 5.2: The application displays the analysis result for a company with histogram and pie chart

## Experiments and Analysis

Our team has done several experiments with the available services on IBM Bluemix throughout the project. We have explored the IBM BigInsight module, IBM Watson Relationship Extraction and Concept Expansion modules, Auto-Scaling, and finally the Monitoring and Analytics module. Each of these modules has their own specific functionalities, which will be described in detail in the next subsections.

### Watson Relationship Extraction

The Watson Relationship Extraction service provides a parser to detect and extract entities for a given sentence (string input). It will return an XML or JSON document which contains all the detected entities and terms from the sentence, such as people’s names, name of places, date, numbers, and so on. The entity detection uses contextual analysis to split the sentence into parts of speech (noun, verb, adjective, conjunction, etc.) and functions (subjects, objects, predicates, etc.). It provides a way for the developer to understand the meaning of individual sentences and documents more easily.

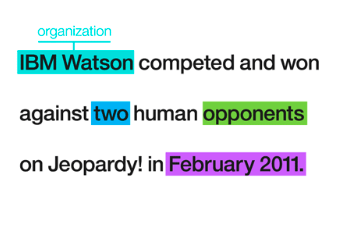


Figure 6.1: A sample result from Watson Relationship Extraction service

The service can be invoked by first binding it into our application from the IBM Bluemix dashboard. Once we bind the service, it will generate the authentication details, such as username, password, and the API endpoint for invoking the service. The next step is to call the service via the provided REST API. For this project, we use Java Apache HTTPClient to issue a POST request to the API endpoint. The POST request contains the string to be analyzed, as well as the desired return format (XML or JSON).

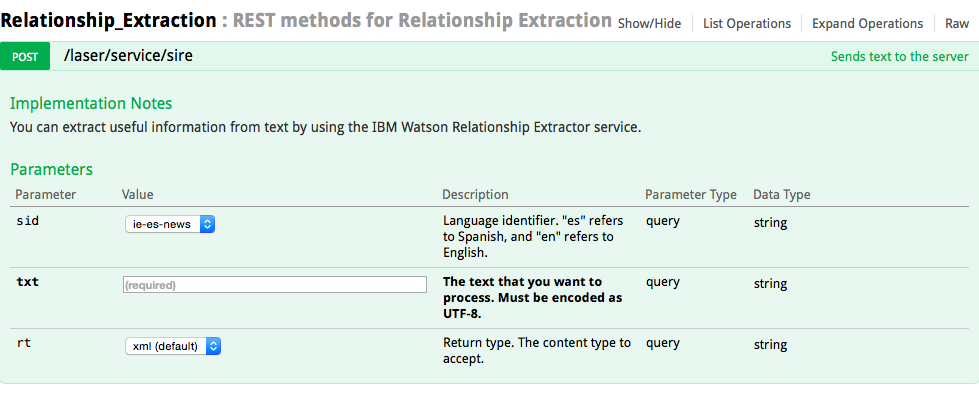


Figure 6.2: The REST API endpoint for Watson Relationship Extraction

According to our observation, this service should be useful for extracting relevant entities for risk analysis on 10K and 10Q documents. However, apparently the service is not able to detect the relevant “risk keywords”. We tried to pass several 10K and 10Q documents to the service and it just returns entities such as company names and figures without any of the keywords. Therefore, we were not able to do the risk analysis because our method is based on the number of occurrence of each keyword. Furthermore, it has some limitations on the maximum length of string we can pass to the API. Both 10K and 10Q documents are too long to be directly passed to the API. Even if we split the document into several smaller parts, the result will not be correct because the service should be executed to a whole document at once in order to correctly detect the relationship between words. Finally, the API takes a considerable amount of time to process. For the above reasons, we decided not to use this service for this project.

### Watson **Concept Expansion**

Another Watson service that came to our attention is the Concept Expansion service. For a given input text, the service is able to analyze the text and interprets its meaning based on usage in other similar contexts. For example, it could interpret 'The Big Apple' as meaning 'New York City'. It can be used to create a dictionary of related words and concepts so that euphemisms, colloquialisms, or otherwise unclear phrases can be better understood and analyzed.



Figure 6.3: The Watson Concept Expansion service can expand a term into other similar terms.

The service provides a REST API endpoint for easy invocation. We use the Java Apache HTTPClient to create a POST request with the required parameters.

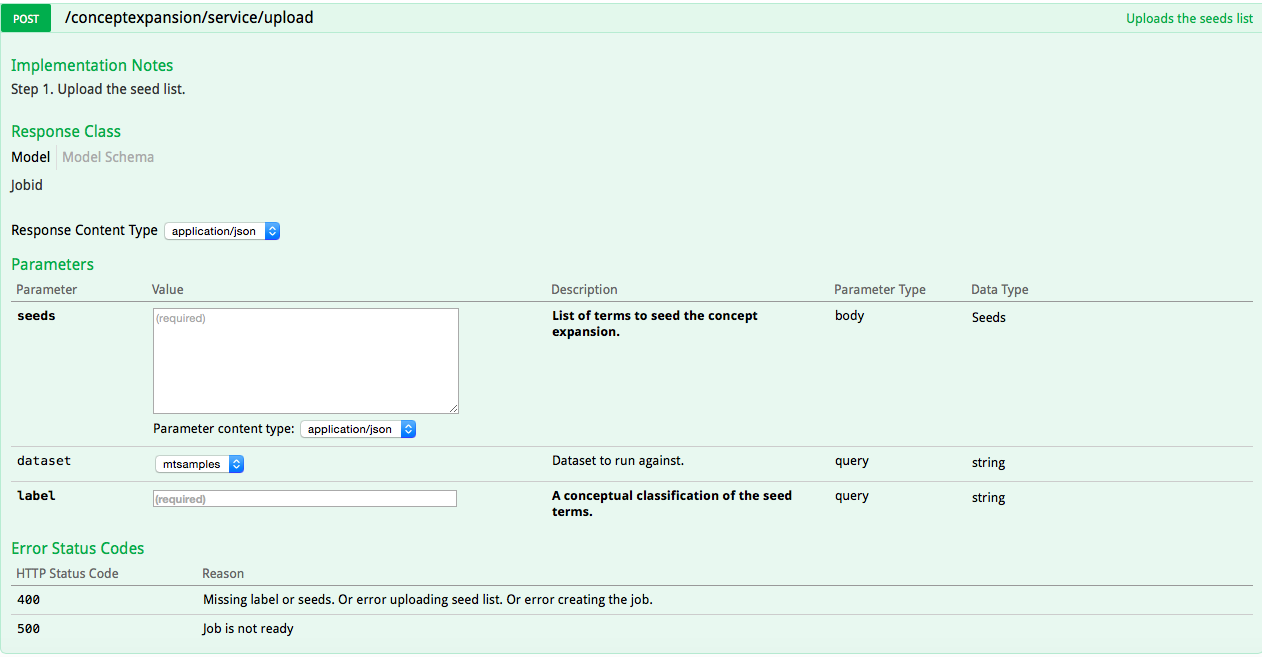


Figure 6.4: The REST API endpoint for Watson Concept Expansion

There are several reasons why we choose not to use this service for this project. First, at the time of writing, it only supports two data sets: periodically updated random tweets and medical transcript samples from MTSamples. As the input text should be related to either one of the data sets, then it will not produce a correct result if we put some financial risk keywords. Furthermore, it is still in beta version and we could not expand or modify the data set. Therefore, we only use this service for experimental purposes.

### IBM Analytics for Hadoop

BigInsights includes pre-built extractor libraries that you can use to extract a set of entities from input text, or extend the views of these extractors to develop custom extractors. This is very useful in analyzing big data. We can also use Text Analytics Java API to run the text analytics runtime component from a Java-based application. For example, we can invoke the Text Analytics API from a basic MapReduce job. One advantage for BigInsights Text Analytics is that regular expression can be used to match and extract the words from text files, which makes processing 10-K and 10-W data more efficiently. We first downloaded the BigInsight Eclipse plugin on local machine to test the word extraction to see if it works. Then we have confident that this can be integrated in our risk analysis project. The Figure X shows the successful word extraction from the financial report.

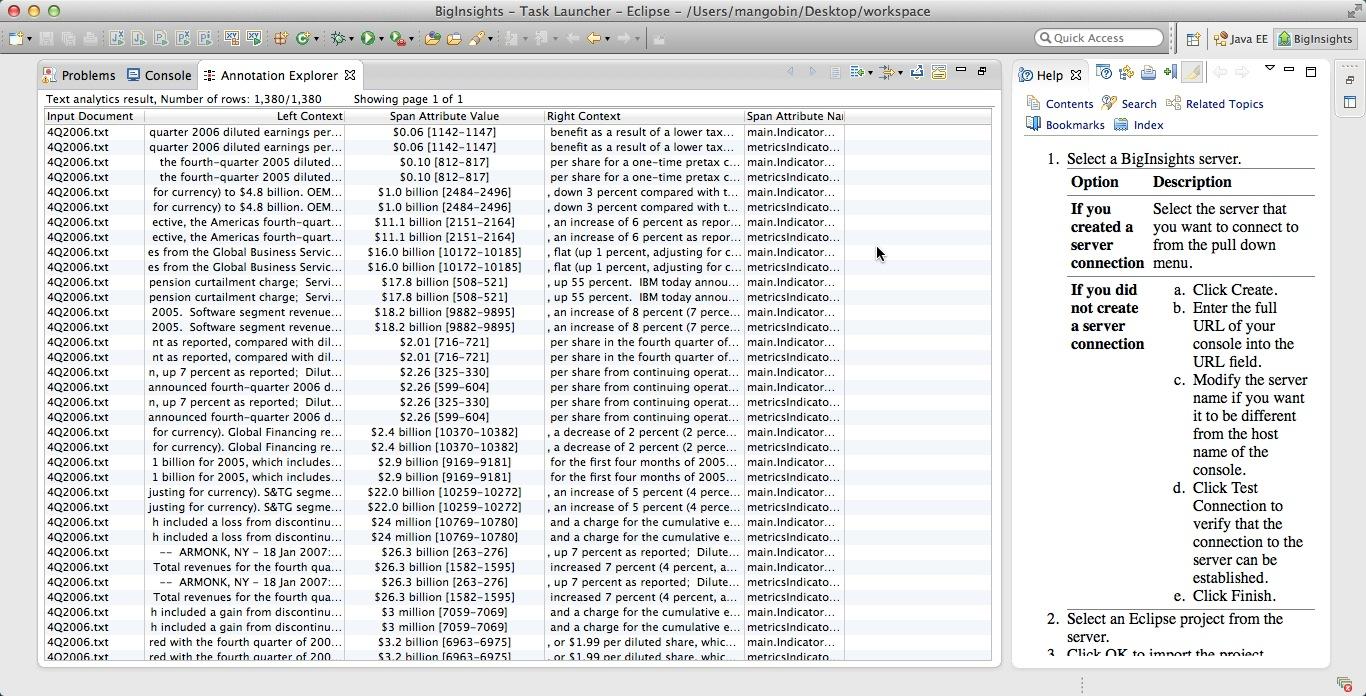


Figure 6.5 : The screenshot of result of text analytics extraction on financial report

### IBM BigInsights Application

Biginsights provides ways to utilize its components to build up customized data processing applications and push it to its server such that automatic distributed computing can be performed on behalf of the application to boost up the performance of the respective big data applications. Also RESTful APIs to access and manage the file system and applications.

In order to process the 10-K and 10-Q file for risk analysis of a company. The Text Analytics application framework was used to build the application for keywords extraction and aggregation. Text Analytics uses AQL (Annotation Query Language) to process and query the file to extract useful information from files.

What the application does is to extract keywords and count them to produce a table like structure for the use of the user. The table like structure is in JSON format. The application was developed in eclipse offline using Biginsights library and published on Biginsights server to be used as a web service. What the service does is to take in an input file directory on biginsights server and process the files inside the directory, outputing the results inside a directory on biginsights server. In order for the process to be fully automatic in user’s view, the RESTful APIs were used. Using the APIs, the risk analysis JAVA application firstly uploads a 10-K or 10-Q file to the biginsights file system. Then the application was passed some configurations to run, outputing the file inside a certain directory on the same biginsights server. Finally, after the run has finished, the output file was read and the content inside the file was passed to the user as a string in JAVA. After all these, all files created inside the file system on biginsights are deleted in order to make space for the next run.

### **Auto-Scaling**

For Auto-Scaling, the interface is obvious, and the functionality is straightforward. However, we were worried about multiple instances. Since we have a database, MongoDB, attached to the instance, we were wondering that if the Auto-Scaling increased one instance, will there be one more database?

So we designed an experiment, the steps are as follow:

1. Activate one instance A of risk analysis web app;
2. Crawl IBM’s information from A;
3. Verify the information accessible from A;
4. Activate another instance B via Auto-Scaling;
5. Verify IBM’s information from A;
6. Verify IBM’s information from B;
7. Crawl Apple’s information from B;
8. Verify Apple’s information from A;
9. Verify Apple’s information from B;
10. Deactivate B via Auto-Scaling;
11. Verify IBM’s information from A;
12. Verify Apple’s information from A;

From the result, we found out that all the verifications were valid, which means that those two instances share one same database. And it makes sense, since we only set the existed database connection address in the code. Auto-Scaling is fully functional.

### Monitoring **and Analytics**

We tried to create an experimental app within a Node.js server. And it turned out that the Performance Monitoring and Log Analysis were not working correctly.

We found no documentation in the IBM Bluemix sites, so we went out searching for help. And finally we found the solution.

1. For the Performance Monitoring, we need to add one line

"scripts":{ "start": "node app.js"}

in the package.json file. And then restart the server.

1. For the Log Analysis, we need to add one line

require("loganalysis")

in the app.js file. And use npm install to add that package, then push the whole thing to the IBM Bluemix.

## **Conclusions and future work**

### Conclusion

In this project, we have successfully fulfilled the planned functionalities using services provided by IBM BlueMix and have our application deployed on the platform.

This application can serve as a sufficient proof for leveraging the concepts of next generation SOA, where developers can go to quickly develop, create, deploy and manage applications in the cloud.

By creating the application on basis of IBM BlueMix, this project helps to push the traditional SOA to next-level, cloud-based SOA, and to advocate the BlueMix platform for more developers to use, even for starters.

### **Future work**

In the next step, we will go on to improve the risk analysis algorithm to include more risk factor information, and do more precise analytics over the information attracted, like determining the dominant factors, mining the correlation of the risk factors.

Also, we should enhance the UI design and make the application a killer app.

## Contribution of each team member

|  |  |
| --- | --- |
| Task | Member |
| Integrate with the previous project | Sammy, Ding, Tan |
| IBM Analytics for Hadoop | Bin, Xuan, Tan |
| Watson Relationship Extraction | Sammy, Ding |
| Watson Concept Expansion | Sammy, Ding |
| IBM BigInsights | Tan, Sammy, Ding, Bin, Xuan |
| Auto-Scaling | Ding, Sammy |
| Monitoring and Analytics | Ding, Sammy |
| 10Q/10K | Sammy, Ding, Xuan |

**Appendix:**

-Check in everything onto GitHub under the predefined directory including the following items

-Readme file: Describe briefly the purpose of the project, how to download and install the software, how to use the software

-API (sub-directory): instruct APIs as well as descriptions and examples

-src (sub-directory): include all source code categorized by packages

-lib (sub-directory): include all related library packages needed to support the project

-contact: please provide every team member’s contact information (cell number, personal email)

-Documents (sub-directory): in different WORD files

-access information: URL, user name/password

-presentations (ppt file)

-tutorial: step-by-step usage file with screen shots included

-technical report

-Transit the knowledge to either Advisor or a signed student (schedule time to sit down for transition)