Continuous Cloud Testing for Web Applications

SouTwitter Team

Liang Li (ll2992) Jin Liang (jl4598) Yu Wang (yw2783)   Lyujia Zhang (lz2467)

1.Introduction

Cloud Infrastructure and Cloud Computing is not strange to us but has been well adopted and implemented in production with more and more institutions moving their legacy system onto cloud. As a model for enabling ubiquitous on-demand access to computing resources [1], Cloud is changing the way of management and delivery of systems, networks and software, as well as the process of application development. It brings new challenges to some conventional software engineering processes such as testing. Thus, a brand new service model for cloud has been proposed: 'Test as a Service' or Cloud testing. It is a form of software testing for web applications running on cloud environment to simulate real-world user traffic and meanwhile brings great benefits such as cost-effective, time-effective and geography-effective. Thus, in this project we would like to adopt the concept of cloud testing and conduct experiments for cloud web applications to evaluate the performance.

Furthermore, we integrate key concepts of software engineering as continuous integration during application development process. The features of cloud testing are 'perfect for Agile' as well as 'perfect for Continuous Integration' [2]. The flow of Continuous integration is quicker with continuous testing. CI means that every time we have add a piece of code and have a new push to git, we have to redeploy it and test it. With cloud test, it renews the concept and means there can always be a tester nearby and test various scenarios right away. This makes new builds faster than ever before and brings great benefits not only to testing but the whole development process.

For the web application, we developed in this project, we focus on Elasticsearch and its performance vs. traditional database such as MySQL. Elasticsearch has recently become the most popular enterprise search engine since January 2016 [3]. It provides a distributed, multitenant-capable full-text search engine with an HTTP web interface and schema-free JSON documents. [4] Thus has been widely used by many businesses to provide advanced search functionalities and better user experience, due to its strengths in search analytics and scalability. When the Elasticsearch search is deployed to the Cloud, the search engine must deliver the right results to the right person at the right time in every single search among hundreds of thousands of queries. Therefore, we built two version of web applications in this project in order to compare the performance difference of Elasticsearch and traditional relational database(AWS RDS). They share the same search functionality: searching for various valuable keywords as well as related information from Twitter stream. The search features include the basic structured queries and advanced full-text searches. Through the search, the user can have knowledge on the following questions, for example, show me all the tweets in the past month tweeted by the residents in New York City, or show me all the tweets where tweeters mentioned Donald Trump.The search results and performance will be handled to cloud test for further analysis.

In summary, the purpose of project is to explore how to perform cloud testing of a web application and reveal how to integrate cloud test with continuous integration by Jenkins. Through the test evaluation, we also trying to evaluate the performance of AWS Elasticsearch comparing to AWS RDS relational database.

2. System Architecture

We built two versions of Twitter Search applications and deployed them onto Amazon AWS. We further conducted cloud testing to test the performance of two versions of web application, especially the performance of elastic search. We also closely integrate the development process as well as testing process with Continuous Integration. So that every time we release a new version of application, it will automatically be deployed onto AWS and cloud testing cases will trigger to test the new version of web application. The whole process will be running automatically without human interference. (figure 1) We adopt the commercial software SOASTA CloudTest and Jenkins for cloud testing and CI tool.

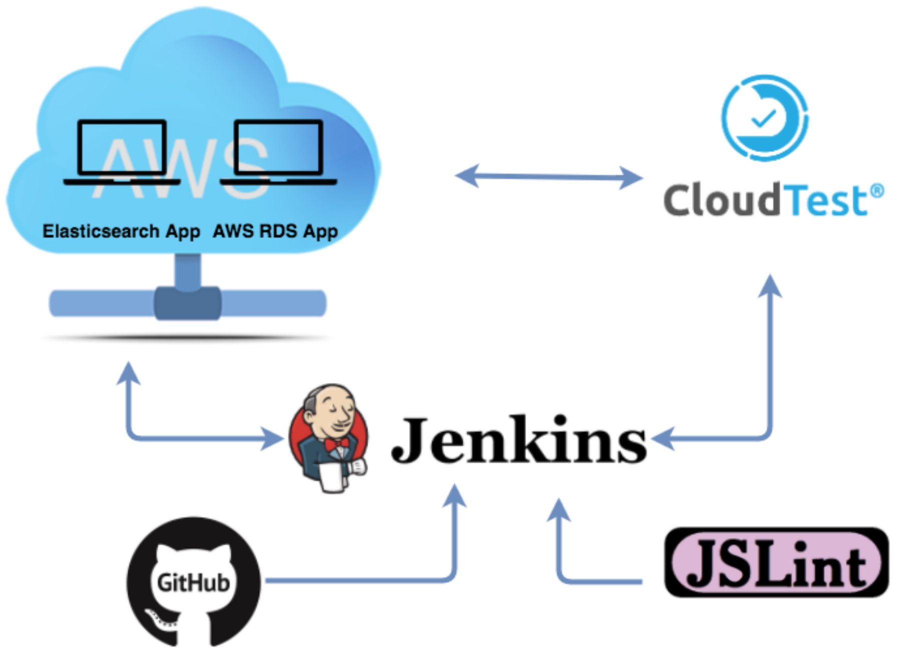


Figure 1. System Architecture

Figure 1 presents the whole system architecture and components. Our application resides on AWS cloud and the source code is managed on GitHub. We have built every connection between those tools so that Jenkins acts the integration role and connects each component as a whole system. Once a code changed has been monitored, several builds operations we defined in Jenkins will be triggered. JSLint as the JavaScript code quality tool will be called to check the code grammar and semantic; Then the application will be automatically deployed onto AWS cloud as a new release version; Corresponding test cases on CloudTest will be performed to test the functionality and performance of web application.

In this report, we will elaborate application development in Section 2. Cloud testing and experiments will be discussed in Section 3. The builds and jobs in Jenkins will be discussed in detail in Section 4. And we will illustrate challenges and what we learned in this project in Section 5. Deliverables, references as well as software we had used will be in the end of this report.

We split the workload mainly as two parts: application development and cloud testing experiments and evaluation. Web applications are developed by Lyujia Zhang and LiangLi; Testing experiments and results evaluation are done by Yu Wang and Jin Liang. We were all working on Jenkins integration.

3.Application Development

We developed the twitter search web applications that could get tweet data from Tweet API, store them into databases and show them on Google map. (figure 2)

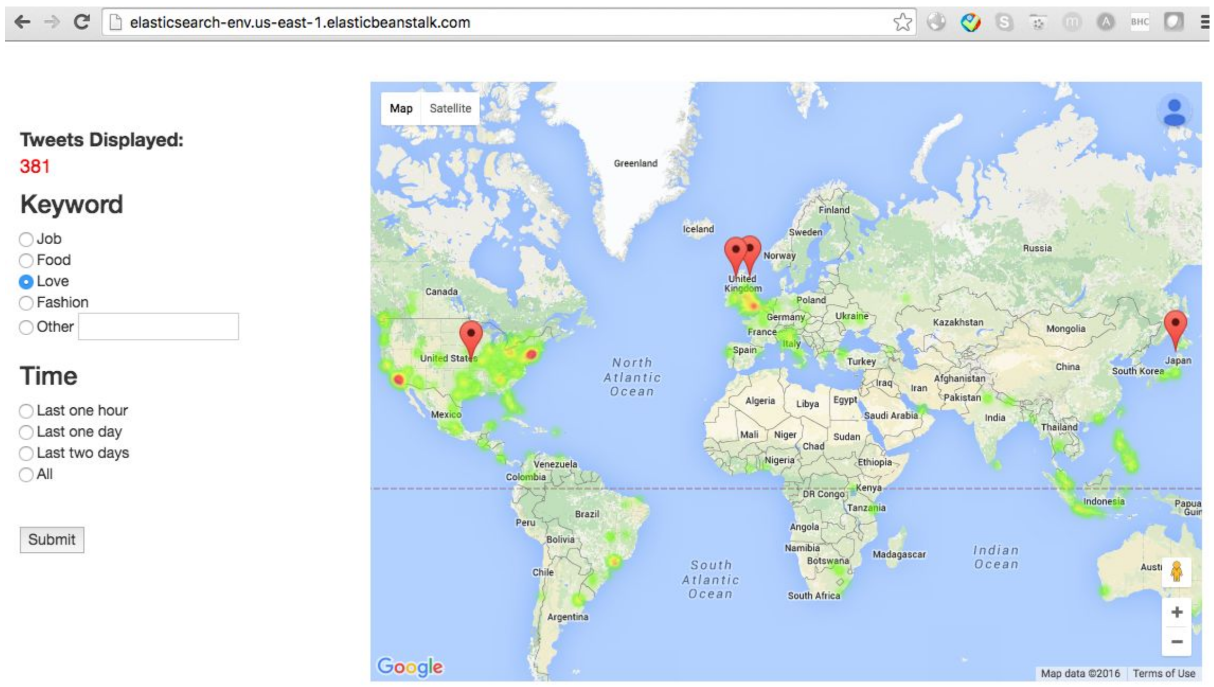


Figure 2. Twitter Search Engine Application

The main functions and features include:

1) Use Twitter Streaming API to fetch tweets from the twitter hose in real-time.
2) Use Amazon RDS (MySQL) or ElasticSearch to store the tweets on the backend.
3) Create a web UI that allows users to search for a few keywords (via a dropdown).
4) Use Google Maps API to render these filtered tweets in the map.
5) Deploy the application on AWS Elastic Beanstalk in an auto-scaling/single instance environment as a preparation for Cloud Testing.

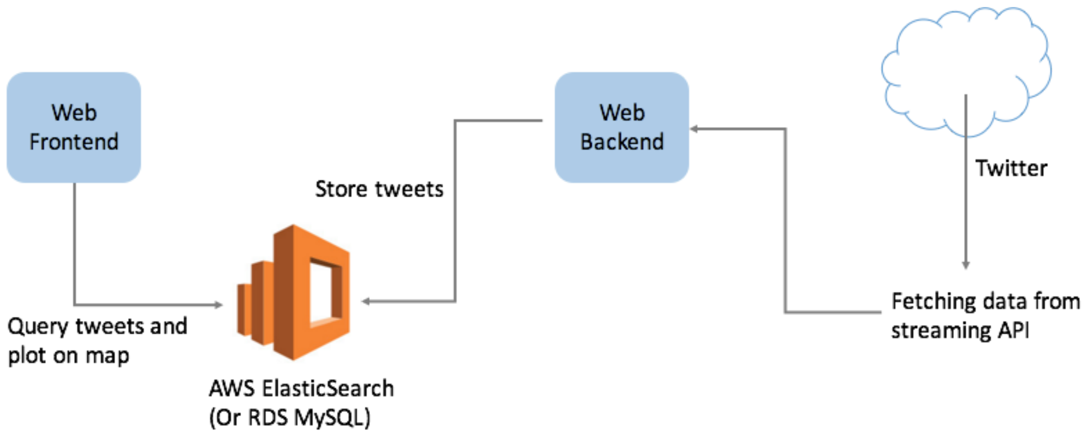

Web Frontend
Web Backend
Twitter
Store tweets
Query tweets and plot on map
Fetching data from streaming API
AWS ElasticSearch (Or RDS MySQL)

Figure 3: Web application work flow

- Tweet Capturing

We first get sample tweet using tweet.4j library, which is an unofficial Java library for Twitter API and is capable of getting sample real-time tweets in JSON format. For this project, we only received tweets with geolocation information attached, which makes it possible to be visualized on Google map.

- Databases

After we get sample Tweets, we extract key information out including user id, tweet content, tweet sent time, latitude, longitude and put the tweet into database. For testing purpose, we used two different databases, mysql and elasticsearch, and kept other settings the same.

- Client and Server

For client side, client could select keyword and server will search from database and return all tweets containing the keyword. Then these tweets will be plotted on Google map using heatmap and the last five tweets will be plotted as markers. If we keep receiving new tweet then on Google Map we can see that the markers are changing their positions.

For server side, we used node.js and express.js framework because it could largely simplify server side configurations.

For server-client communication, we used socket.io to provide a full-duplex communication. Besides, we also used AJAX for partial-page update.

As mentioned, in order to explicitly show that the load testing results towards elasticsearch attribute in the web application, we will compare the metrics with RDS application which has the same attributes and functions, but the only difference is how data is stored and retrieved. In our testing version, the twitter database is about 20,000 record. The web front design keeps the same in two application versions.

4. Cloud Test Evaluation

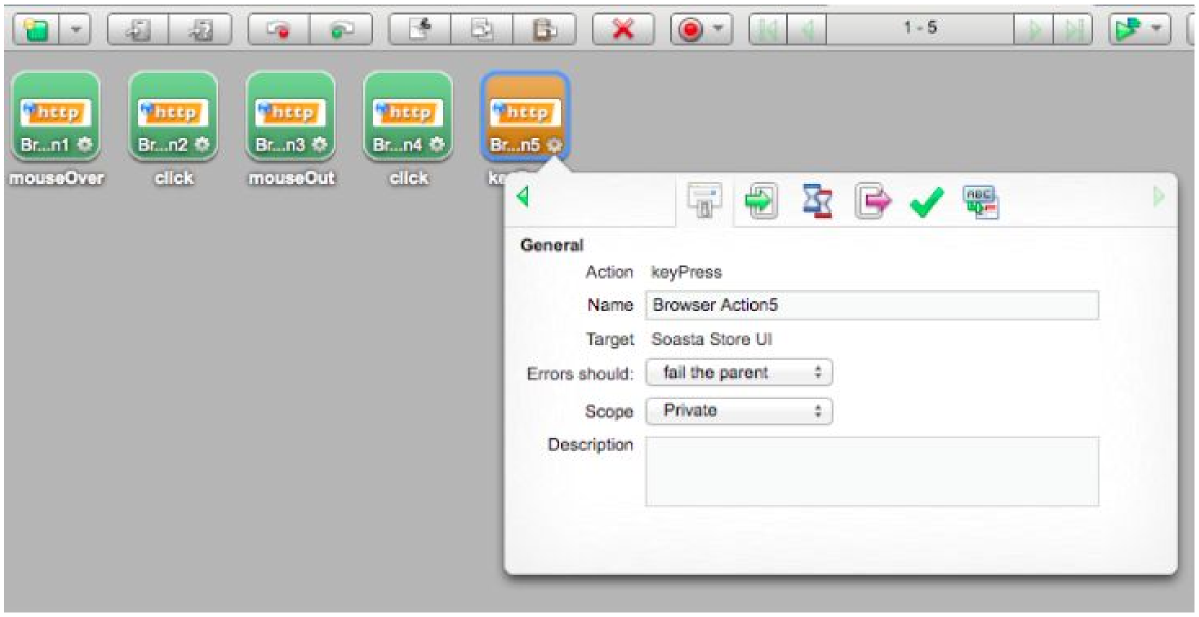
4.1 Test Scope and Environment

There are several commercial tools available for cloud testing and the key players are SOASTA CloudTest, Cloud Testing and iTKO LISA. In this project, we adopt CloudTest as cloud testing software and test the performance of our web applications.

We designed our test plan according to the feature of web application and focus on 3 main scopes of test: functional test, performance test and load test.

- Functional Test

In functional test, we tested the function of web application based on specifications of our application component by feeding input and examining the output. For example, we simulate a single user interaction by clicking the single select buttons on web page and input keywords and click ‘submit’ button to view twitter map plot on Google heat map. The entire process of actions and interaction will be recorded and system response data will be recorded accordingly. Test cases are performed by UI test function in CloudTest referring to its official tutorial: CloudTest® Web UI Testing Tutorial [5]. Figure 4 shows the CloudTest Web UI test interface.



General
Action keyPress
Name Browser Action5
Target Soasta Store UI
Errors should: fail the parent
Scope Private
Description

Figure 4. CloudTest Web UI Testing Interface

- Performance Test

In performance test, we mainly focus on the performance of web application server and database performance. CloudTest provides feature to build 'monitor server groups' so that we can monitor the real-time system performance data of various servers simultaneously on a single graph. In order to observe database behavior more straightforwardly, we also explore the 'direct to database testing' feature of CloudTest to perform tests on database directly. The product provides several metrics such as disk sorts, index scan, full table scan, per catalog connections, etc. However, these metric are targeted for traditional relation database and the current version only supports PostgreSQL, MySQL and oracle. So we are unable to assess Elasticsearch by this feature and would focus mainly on AWS server instance system performance.

- Load Test

Load testing is the key process to test a web application and is also the main purpose of CloudTest. We will perform load test from two perspectives: automation load testing and user defined(pre-parameter) loaded testing. For automation load testing, test cases are generated automatically by CloudTest without specific user interaction with the web application. For user-defined load testing, we use pre-parameterized data or 'seed' data to pre-define how each user of the load will behave and interact with our web application. To be specific, we pre-defined 10 keywords in our test and each simulated user would ' typed in' these keywords into our twitter application.

We adopt the free trial version of SOASTA CloudTest. However, the trial version has a limitation of most 100 virtual users so we deployed 3 CloudTest servers in total in order to simulate more users for load testing. SOASTA also provide direct installation on physical server ESXi on cloud. Due to the hardware limitation, we use general Mac laptops serve as CloudTest server:

Operating System	Mac OS X
Memory	8G/16G
Disk	At least 30G
Dependence Software	VMware Fusion

Table 1. CloudTest Server Configuration

4.2 Test Cases and Metrics

Approach	Type of Testing	Description	Test Case
Functional Test	Record Clips	Record User interaction with application	1. Manual Recording 2. Edit Recording to Clips 3. Analyze
Performance Test	Server Group Test	Test application host server performance	1. connect to physical server 2. build dedicated monitor to track system resources like CPU/memory/IO USER 100 : Random 4 keywords from pre-parameterized seed data
Load Test	Visual User Simulation	Simulate different user loads range from 5 to 300.	1.Record user interaction with web application 2.Prepare seed data 3.Run test case under different visual user load: 5; 100;200;300 users

Table 2. Test Cases

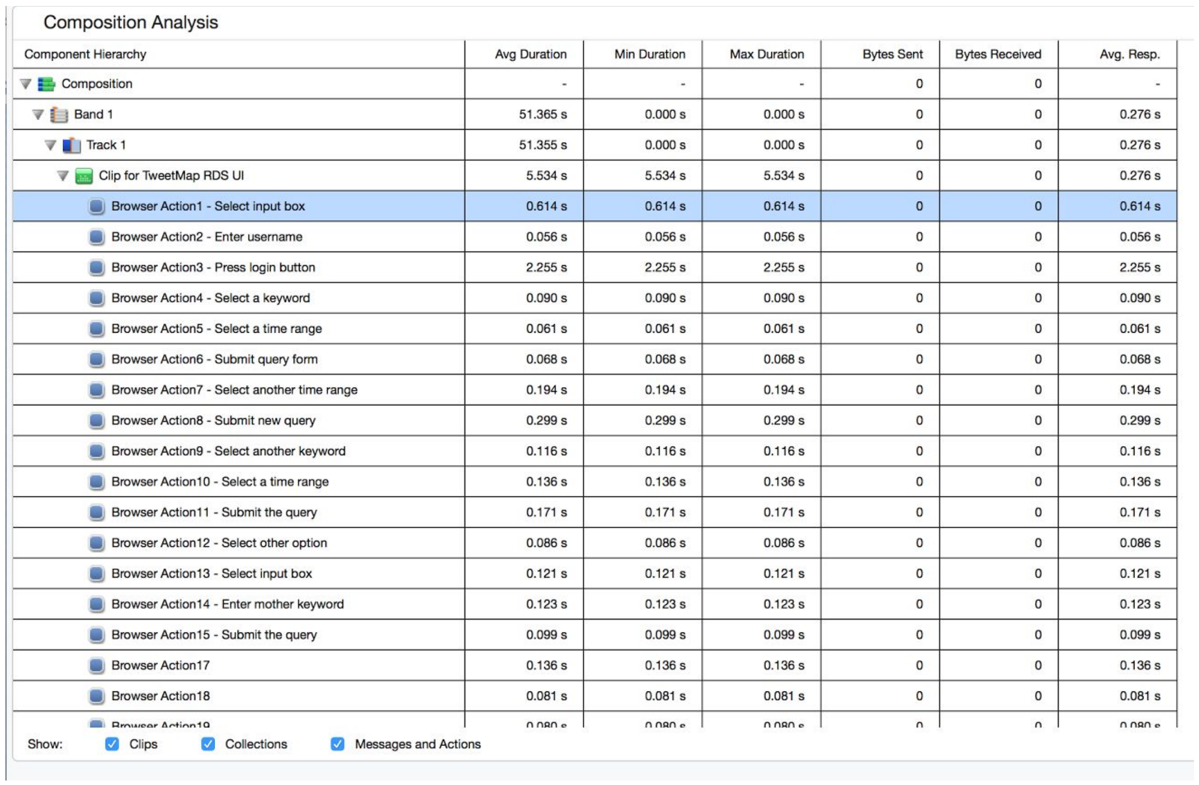
We following the general metrics for web application, which is also implemented in CloudTest panel to indicate the results the testing.

- **Running Time:** elapsed time or running time for test case
- **Message/ Action Errors**: error count for message or action error
- **Msg Sent**: number of sent messages
- **Average Response Time**: average response time from server, including minimum and maximum response time
- **Total Bytes**: the total bytes of sent/received by cloud test server
- **Effective Throughput:** number of messages per second

4.3 Evaluation

- Function Test

The results of function testing are shown in the table below. According to the table, our web apps successfully pass our test cases and function well. It is expected that the app users will have fluent and friendly user experience. The test case is built after recording and capturing all user actions and user flow when users are visiting our web apps. The user actions and user flow include various kinds of Click, Type, Drag and Drop, as shown in the first column of the table. For example, the first browser action activated by the user is selecting the input box for the username on the login page.



Composition Analysis

Component Hierarchy	Avg Duration	Min Duration	Max Duration	Bytes Sent	Bytes Received	Avg. Resp.
Composition	-	-	-	0	0	-
Band 1	51.365 s	0.000 s	0.000 s	0	0	0.276 s
Track 1	51.355 s	0.000 s	0.000 s	0	0	0.276 s
Clip for TweetMap RDS UI	5.534 s	5.534 s	5.534 s	0	0	0.276 s
Browser Action1 - Select input box	0.614 s	0.614 s	0.614 s	0	0	0.614 s
Browser Action2 - Enter username	0.056 s	0.056 s	0.056 s	0	0	0.056 s
Browser Action3 - Press login button	2.255 s	2.255 s	2.255 s	0	0	2.255 s
Browser Action4 - Select a keyword	0.090 s	0.090 s	0.090 s	0	0	0.090 s
Browser Action5 - Select a time range	0.061 s	0.061 s	0.061 s	0	0	0.061 s
Browser Action6 - Submit query form	0.068 s	0.068 s	0.068 s	0	0	0.068 s
Browser Action7 - Select another time range	0.194 s	0.194 s	0.194 s	0	0	0.194 s
Browser Action8 - Submit new query	0.299 s	0.299 s	0.299 s	0	0	0.299 s
Browser Action9 - Select another keyword	0.116 s	0.116 s	0.116 s	0	0	0.116 s
Browser Action10 - Select a time range	0.136 s	0.136 s	0.136 s	0	0	0.136 s
Browser Action11 - Submit the query	0.171 s	0.171 s	0.171 s	0	0	0.171 s
Browser Action12 - Select other option	0.086 s	0.086 s	0.086 s	0	0	0.086 s
Browser Action13 - Select input box	0.121 s	0.121 s	0.121 s	0	0	0.121 s
Browser Action14 - Enter mother keyword	0.123 s	0.123 s	0.123 s	0	0	0.123 s
Browser Action15 - Submit the query	0.099 s	0.099 s	0.099 s	0	0	0.099 s
Browser Action17	0.136 s	0.136 s	0.136 s	0	0	0.136 s
Browser Action18	0.081 s	0.081 s	0.081 s	0	0	0.081 s
Browser Action19	0.080 s	0.080 s	0.080 s	0	0	0.080 s

Show: ✓ Clips ✓ Collections ✓ Messages and Actions

Figure 5 Function Test Result

The average durations and response times for each user actions are also recorded in the table below. It can be seen that some user actions take longer time than others. For instance, it takes 2.255 s when users are pressing login button. Obviously, in order to provide smoother user experience, we need to examine our login button part of our code, and try to reduce the login button's response time.

- Performance Test

The performance test part is targeted for AWS server and we built server monitors to monitor the server performance in real time when user traffic is loaded to web application. The system performance indicators include CPU percentage, Memory Usage, IO reads/writes Network Mbits sent/received, etc. Figure 6 shows the CPU usage trend when 300 visual user is simulated on elastic search application.

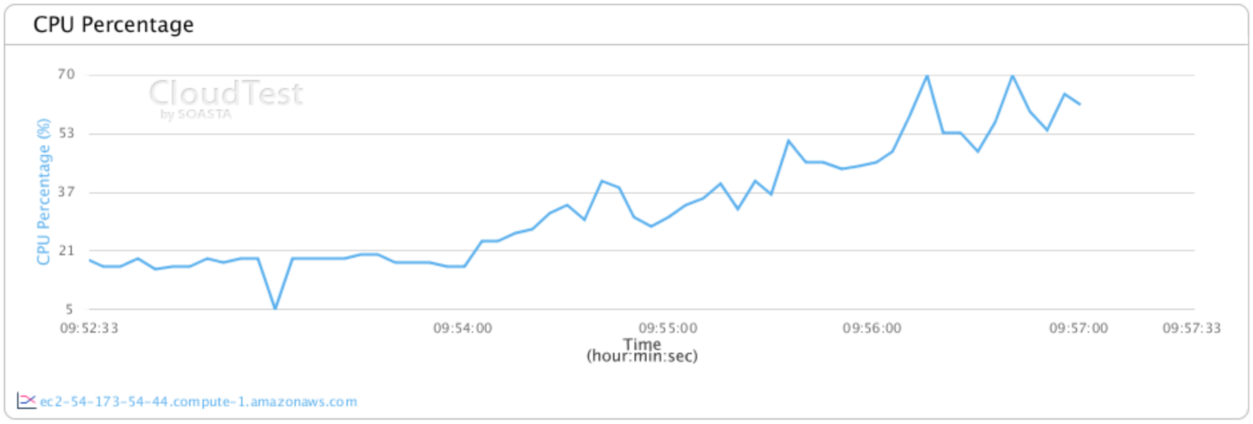


Figure 6. CPU percentage trend for elastic application server

- Load Test

The results of all the load testing are shown in the table below(average response time). Four kinds of testing scenarios are tested with SOASTA CloudTest software package. There are five, one hundred, two hundred, and three hundred virtual users in each testing scenarios. It can be seen that the web app with ElasticSearch has less response times and better performance in some scenarios, but the web app with RDS better in other scenarios. We recorded the detail data in TestPlan document.

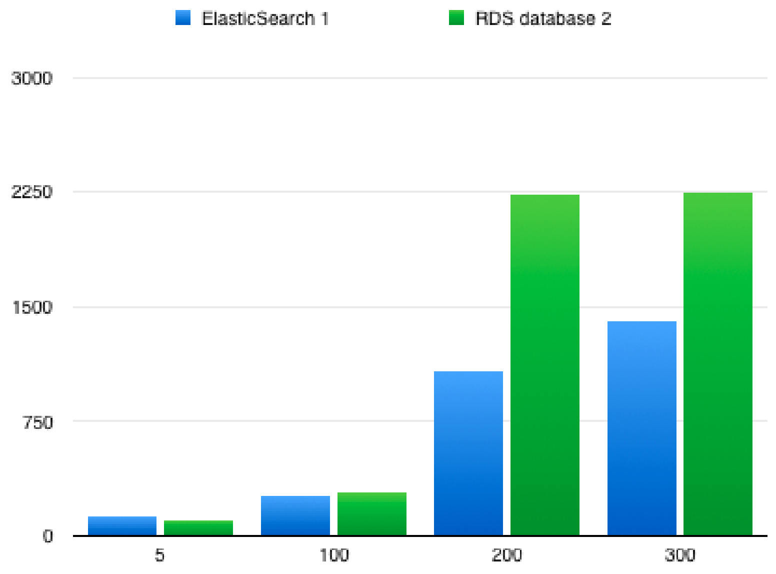


Figure 7 Average Response Time Comparison

The difference of load test results may lay in various dimensions. First, search performance can greatly rely on the data. Though we have the same amount of data record in both databases, they pull tweets separately and the actual data records are not the same. Thus it is highly possible that a specific keyword search is faster in one than the other. Second, the performance is highly related to database performance in general. As elasticsearch enables full-text search, it is much easier to search for a specific keyword in the whole database than traditional relational databases. According to our test case results, Elasticsearch does have better performance in most ‘heavy load’ scenarios. Third, AWS cloud environment may also influence the test results. As the cloud infrastructure is transparent to users and we do not have control on the server instance. It might be shifting around when we conduct the test cases or experiencing other changes within the cloud which are not visible to us.

5. Jenkins Integration

Jenkins is integrated to our project for the purpose of connecting the activities of application development and testing. To stimulate the situation where a stable website needs to be maintained while adding more features to it and doing the testing, continuous delivery plays an important role in our project.

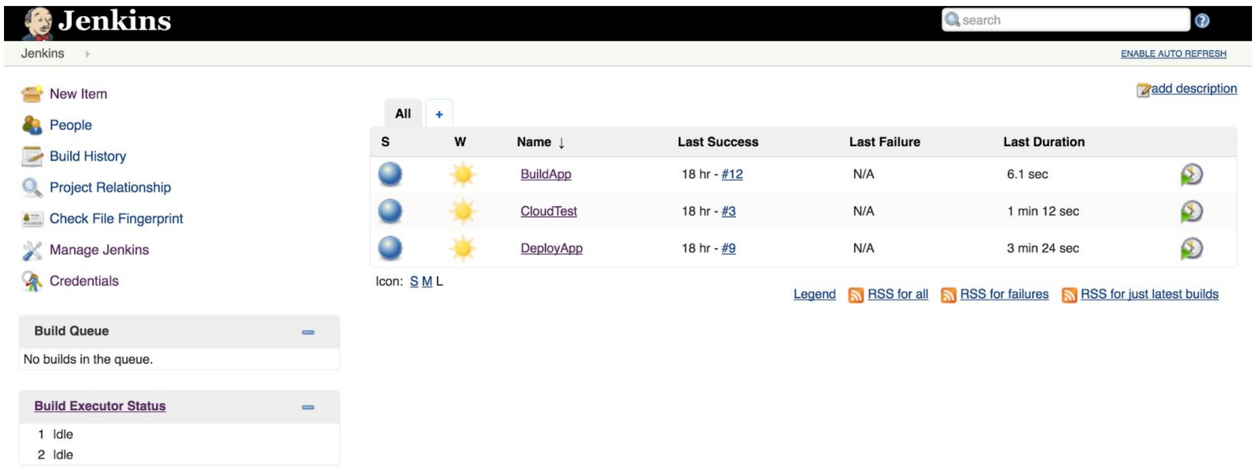
5.1 Jenkins Plugins List

There are several plugins Jenkins provides to be installed before we integrate our project with Jenkins.

- Github Plugin: To enable downloading updated repository from Github.
- Copy Artifact Plugin: To enable passing artifacts from build job to deploy job.
- AWS Elastic Beanstalk Deployment Plugin: To enable deploying the web application to AWS Elastic Beanstalk server.
- SOASTA CloudTest Plugin: To enable automatic cloud testing.

5.2 Jenkins Job workflow

The workflow of the Jenkins is captured in the system architecture section. The workflow can be divided into three modules or jobs, which are BuildApp, DeployApp and CloudTest, shown as below.



S	W	Name ↓	Last Success	Last Failure	Last Duration
●	☀	BuildApp	18 hr - #12	N/A	6.1 sec
●	☀	CloudTest	18 hr - #3	N/A	1 min 12 sec
●	☀	DeployApp	18 hr - #9	N/A	3 min 24 sec

Build Queue

No builds in the queue.

Build Executor Status

1 Idle
2 Idle

Figure 8 Jenkins Workflow

The BuildApp job will be triggered first by every Github push. It will download the latest version of the web application and pass it through JSLint plugin to checkstyle of the JavaScript codes. After above steps are done, it will save the artifacts, which is our web application file folders, and trigger the next job.

The DeployApp job is then triggered. First it will copy the artifacts from the BuildApp job. Then it will deploy the application to AWS Elastic Beanstalk, where we already have an application environment set up for it deploys to. After this job is done, we will have the latest version of the web application ready to be test.

Lastly, the CloudTest job is triggered after the deployment is finished. It will automatically start CloudTest server to run load testing, and we will receive email notifications after the result of testing is out.

Ideally those three jobs can be deployed to three different servers, and Jenkins server can also be deployed to one server. That is to say, the completed process could be done on the cloud automatically, triggered by Github push and thus no human involvement is needed. However, because our web application is simple and we don’t have such complicated build and deploy steps, unlike some C language based projects, all three jobs mentioned above are simply done locally.

6. Challenges and What We Learned

6.1 Challenges

Our project consists of two parts, web app development and cloud testing.

Web app development:

Before this project, we already had another version of our tweet map application, which was the project from COMS 6998 - Cloud Computing and Big Data. However, we also made several changes to it. We changed server side completely to Node.js and simplified the front end functionality.

The first challenge we met is how to use Node.js and socket.io work together. For testing purpose, we must use form in the client side for data submission. However, in the beginning we found that whenever server side receives a request and respond via socket.io, it will always be in a waiting status, and the page will reload after server seconds. After intensive Googling we found that we didn't define HTTP response. Then we added it and it turns out the the page will be re-directed and reloaded, which is not desired since we want our data to be updated on the Map. Finally we found that AJAX is the right way for partial-page update and we integrated it into our application and made everthing works.

We also met some problem in dynamic routing, as we want the user information to be part of url, and this helps for debugging and testing purpose. However, we struggled for a long time and could make it work since the tutorial online were not exactly what we want. Finally we decided to use just static routing instead and print the user information on the web page to help debugging.

Cloud testing:
The concept of Cloud Testing as well as the SOASTA software are relatively new and we have no previous experience of the concept nor the cloudtest software. We have run into several challenges during the project from deployment and executing test.

- SOASTA cloudtest deployment

The cloud testing tool from SOASTA is not actually a traditional software but a packed virtual machine in cloud environment. To be specific, SOASTA provides two ways of installation: running as an individual virtual machine or directly deployed on hypervisor physical host (ESXi). Thus, in our test environment, we neither have cloud infrastructure or hypervisor host server. So the final solution is we firstly install virtualization software and build virtual machine on the virtualization environment. We built 4 cloudtest servers in total.

- Visual User Simulation

SOASTA provides two modes of service/produce: on-demand test and product. The minimum package of on-demand test is $2,500 and product license would be $25,000. So we adopted free trial version but it only allows 100 virtual users for one server. Thus, to maximum the possible virtual users, we deployed 4 servers and able to simulate 400 virtual users simultaneously in total.

- Cloud Server Deployment

A key feature for cloud test is to deploy test cases on cloud servers which might locate in different geographic location. The specific feature on CloudTest is build in 'grid management' which enable to deploy test cases on various server instances on cloud located in different geographic locations like London, San Francisco. However, this feature is not available in free trial version. Thus we are unable to test simultaneous deployment of test cases on different servers.

6.2 What We Learned and Acknowledgement

Jin Liang

I learned how to do cloud testing with SOASTA CloudTest software package, which is used by big companies such as Wal-Mart, Microsoft, and NordStrom. The cloud testing process provides me with a whole new perspective of web application developments. I obtain basic understanding of function testing and load testing on web apps. In the near future, I hope I can do the cloud testing with mobile apps, which seems to be the trend. Moreover, I think it is interesting to see the differences in documentation styles between the software products from SOASTA and Elastic. Elasticsearch from Elastic is my mid term research topic. Many thanks to Professor Gail, through the course I am able to be exposed to many interesting areas in software engineering. Recent one is the React.js. Also many thanks to my teammates.

Liang Li

I was responsible for the development of web application. During the development, I learned about details of how to use Node.js and Express.js framework, and realized that they are really easy to configure and simple to use. I also had a better understanding of the difference between Post and Get Http Request, how server actually established connection with client via pre-defined end point, and how static routing and dynamic routing works in the communication. Besides, I learned why there is a need to use AJAX for asynchronous communication and really benefit from it in this project.

I appreciate all the effort that my teammates and I had made to make our project finished on time, and especially thanks for their help when I met problems in the development. It’s been great memory to work with you guys.

Yu Wang

My primary endeavour for this project is Cloud Testing, including environment configuration, test plan design, test execution as well as the integration with Jenkins. The whole concept of cloud testing and the software is brand new subject to me so I learned a lot through every single step in the project. First, I had a better understanding of cloud test and how it actually works in a cloud environment. Second, by plan and execution of test cases in SOASTA cloudtest, I learned how to use and operates the software to perform function test, load test and how to monitor the server performance as well as database performance. Third, a large effort is to find out if CloudTest can work with Jenkins as the primary material and tutorial is about mobile application plugin. We did not make it work until several trial on Jenkins. Thus in this build step, I managed to learn how Jenkins work in general and how it can actually connect to cloudtest server and play targeted test cases. In summary, through the project I learned how to put the ‘test as a service' into practice and how to manage the test automation workflow in Jenkins to achieve continuous integration.

Lyujia Zhang

My major responsibility for this project is to set up an Elasticsearch version application based on Liang's RDS version, also to design and build the Jenkins workflow. I learned how Elasticsearch works, and also how to save the information to the database, how to query the database, etc. By building the Elasticsearch version application, I understood the structure difference between Elasticsearch and RDS MySQL database. Because I've used Jenkins in my past experience, designing the workflow is easy to do. However, using the plugins of CloudTest and AWS Elastic Beanstalk was new to me. By trying out new plugins of Jenkins, I learned that Jenkins is very powerful in many aspects. I think it's a great experience working with my teammates, from brainstorming to implementation, and it's amazing how we blended everyone's idea into one project and made it a worthwhile research.

We sincerely extend our gratitude to professor Kaiser and TAs for your help and suggestions in our project.

7.Deliverables

Github Source Code
https://github.com/zljchloe/SE-CloudTesting
Test Plan and Results
Test Plan.xlsx
Final Report

References

[1] Wikipedia "Cloud Testing" Retrieved from https://en.wikipedia.org/wiki/Cloud_testing
[2] 'Top 10 Benefits for Mobile Cloud Testing' Retrieved from :
https://experitest.com/top-10-benefits-of-cloud-testing/
[3] Andlinger, Paul. "Elasticsearch Replaced Solr as the Most Popular Search Engine." Elasticsearch Replaced Solr as the Most Popular Search Engine. N.p., 12 Jan. 2016. Web. 07 Apr. 2016.
[4] Wikipedia 'Elastic Search ' Retrieved from https://en.wikipedia.org/wiki/Elasticsearch
[5] CloudTest ® WebUI Testing Tutorial. (n.d.).

Softwares

1.SOASTA CloudTest
https://www.soasta.com

2.Jenkins
https://jenkins.io/
3.Github
https://github.com/
4.JSLint
http://www.jslint.com/