The implementation of a browser-based source code navigator and its cloud solution

Submitted by

Tong Zhou
dns.t.zhou@gmail.com

Industrial Director:
Jonathan Halliday

Academic Supervisor:
Dr. Paul Ezhilchelvan

Submitted in partial requirements for the degree of Master of Science in Cloud Computing

School of Computing
Newcastle University

August 2019

Abstract

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development[22], but on certain occasions, developers want to browse all java files within an entire jar file.

This dissertation is mainly about the design and implements of a kind of web-based source code navigator for Java, called Online Java Navigator, which provides an online user interface for all developers to browse specific repositories of java source code. Different users can access a general server that uses a unified storage system to look up all relevant information required to resolve the navigation result. With this browser-based source code navigation system, users can view every Java file as an online web page, and click any validated type name within the page, then the system will direct them to the right target java type that is quoted by current scope.

To provide service to users from all over the world, a good cloud structure is pretty crucial during both deployment stage and development stage. There are different kind of solutions to choose from, and all these solutions have their advantages and disadvantages, thus an evaluation of these solutions was conducted during this project.

Declaration

I declare that this dissertation represents my own work except where otherwise explicitly stated.

Acknowledgements

Firstly, I would like to express my thanks to Mr Jonathan Halliday and Dr Paul Ezhilchelvan for their help, advice and patient guidance during this project. I would also like to expand my gratitude to all those who have directly and indirectly guided me in writing this dissertation.

Also, I would like to express my gratitude to my lecturers during the entire MSc programme: Dr Paul Ezhilchelvan, Dr Ellis Solaiman, Dr Stephen McGough, Dr Nigel Thomas, Professor Raj Ranjan, Dr Yu Guan, Dr Paolo Missier. There were so much useful knowledge and information I achieved from those lectures, which benefits me a lot both academically and practically.

This project was so much different from any of my previous project because this was an industrial student project which requires for both academic knowledge and engineering practice, and this project benefited me a lot during the past four months.

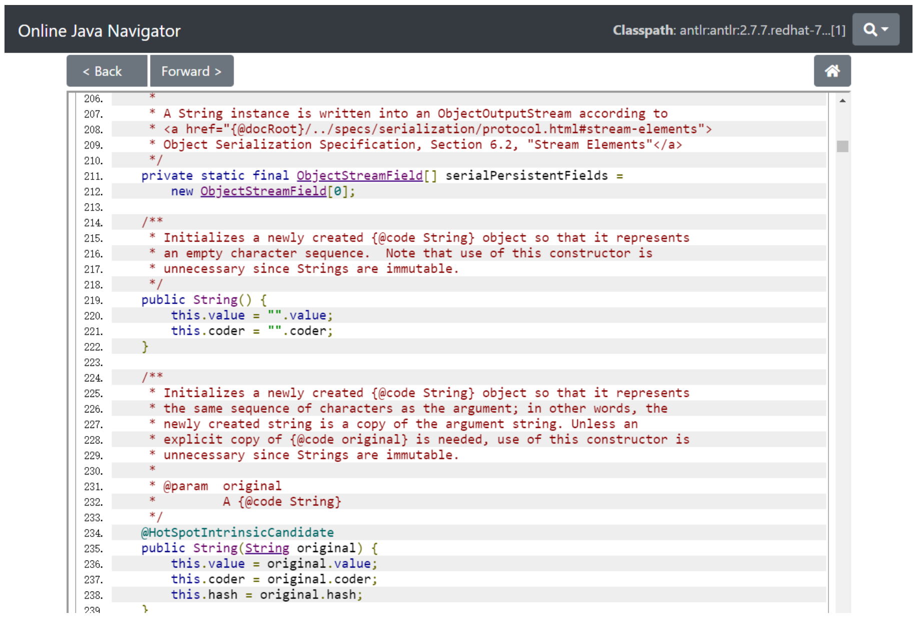
Finally, I would like to thank the Newcastle University and Red Hat Ltd in general, for all kinds of support from academic materials to facility resources during my project.

Table of Contents

Abstract . . . ii
Declaration . . . ii
Acknowledgements . . . iii
1 Introduction . . . 1
2 Background . . . 2
2.1 Current solutions . . . 2
2.2 Compilation and execution of Java program . . . 4
2.3 General requirement . . . 5
3 Comparative Research . . . 5
3.1 Java Parsing . . . 5
3.2 Syntax Highlight . . . 6
3.3 Front-end Framework . . . 7
4 High-level Architecture . . . 8
5 Source code analysis . . . 10
5.1 Source code to AST . . . 11
5.2 AST to Html . . . 12
5.3 AST to JSON file . . . 12
5.4 AST to Index file . . . 13
6 Navigation . . . 13
6.1 Universal Unique Type Name(UUTN) . . . 13
6.2 Resolve Function . . . 14
6.3 Priorities for Resolving . . . 14
6.4 User-customised classpath . . . 16
7 Cloud Solution . . . 17
7.1 EC2 + EBS . . . 17
7.2 EC2 + S3 . . . 19
7.3 Lambda + API Gateway + S3 . . . 20
7.4 EC2 + EBS + S3 . . . 20
8 Evaluation . . . 22
8.1 Unit Testing & Integrate Testing . . . 22
8.2 Automation Testing . . . 23
8.3 Performance Testing . . . 23
8.4 Project Management . . . 26
9 Conclusion . . . 28
9.1 Significance and Contribution . . . 28
9.2 Limitation and defects . . . 28
9.3 Future improvements . . . 29
Appendix 1: Relavant source code . . . 32
Appendix 2: EBS Pricing . . . 40
Appendix 3: S3 Pricing . . . 41

1 Introduction

This dissertation is mainly about the design and implements of a web-based source code navigator for Java, called **Online Java Navigator**[20], which provides an online user interface for all developers to browse specific repositories of java source code.



```
     *
     * A String instance is written into an ObjectOutputStream according to
     * <a href="{@docRoot}/../specs/serialization/protocol.html#stream-elements">
     * Object Serialization Specification, Section 6.2, "Stream Elements"</a>
     */
    private static final ObjectStreamField[] serialPersistentFields =
        new ObjectStreamField[0];

    /**
     * Initializes a newly created {@code String} object so that it represents
     * an empty character sequence.  Note that use of this constructor is
     * unnecessary since Strings are immutable.
     */
    public String() {
        this.value = "".value;
        this.coder = "".coder;
    }

    /**
     * Initializes a newly created {@code String} object so that it represents
     * the same sequence of characters as the argument; in other words, the
     * newly created string is a copy of the argument string. Unless an
     * explicit copy of {@code original} is needed, use of this constructor is
     * unnecessary since Strings are immutable.
     *
     * @param  original
     *         A {@code String}
     */
    @HotSpotIntrinsicCandidate
    public String(String original) {
        this.value = original.value;
        this.coder = original.coder;
        this.hash = original.hash;
    }
```

Fig. 1: Product User Interface

With this system, users can view every java file as an online web page, and click any validated type name within the page, then the system will direct them to the right target java type that is quoted by current scope. The system support user-customised classpath, which means the user can define the boundaries of browsing and navigate only within a specific scope of source code. Also, all java codes are displayed with syntax highlighting, so that users would have a better user experience while reading those codes.

The system uses a non-container solution, which means it is different from the online IDE system. An online IDE uses container technology to create an individual container for each user instance so that they can browse, run and debug code remotely, it does help to build a robust environment for developer to use online but at the cost of a lot of resources consumed to provide service to developers from all over the world. Instead, a non-container solution uses a universal server group for any access from the internet and provide service with low cost and high efficiency. The product is deployed on AWS public cloud with

CDN[15] and BLOB[12] storage system, which means it can provide service to users from all over the world with high availability and low latency.

You can access the demo environment of this project via this link:

http://bit.ly/2T5ODrx

2 Background

An integrated development environment (IDE) is a software application that provides comprehensive facilities to computer programmers for software development[21]. A handy feature provided by IDE is the ability to navigate between different files so that they can conveniently navigate the Types that used in any part of codes. Currently, there are a lot of IDE that provides useful functions like that, but sometimes developers may want to browse code online as a document. For single file cases, this is sometimes possible; for example, GitHub[19] provides an online webpage for every file in the repository, user can easily browse and edit the code.

On certain occasions, developers want to browse all java files within an entire jar file, because now software products are becoming more and more complicated, developers are trended to implement some open-source project into their product to boost their development, and these open-source products are usually packed as jar files. Within these jar files, there are a lot of java files which have internal quotes with each other, just like when you import specific java class and use it in another java class. Because of this domestic relationship between different java files, it is especially useful if developers can browse all related open-source code online with navigation between different java files, just like what a local IDE can provide to users.

2.1 Current solutions

There are some existing solutions and products on the market that provide these features, which can be concluded as two types of solutions:

Container-based solutions: Gitpod[23], SourceGraph[24] These two products similarly provide an individual container to every user instances. Within the container, the system automatically builds a java development environment and integrate a front-end user interface to interact with users. When the user loads any specific project, it passes the code to the container, then compiles, build and debug it in realtime.

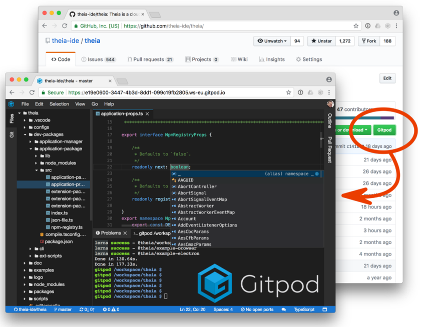
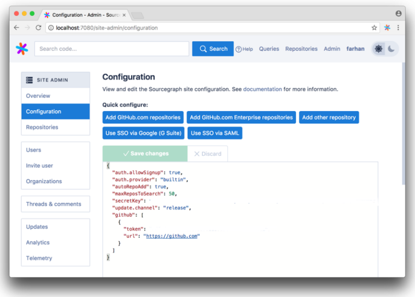


Fig. 2: Gitpod

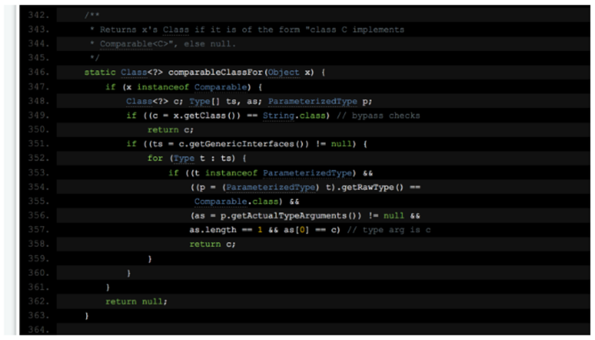

Configuration
View and edit the Sourcegraph site configuration. See documentation for more information.
Quick configure:
Add GitHub.com repositories
Add GitHub.com Enterprise repositories
Add other repository
Use SSO via Google (G Suite)
Use SSO via SAML
Save changes
Discard

```
{
  "auth.allowSignup": true,
  "auth.provider": "builtin",
  "autoRepoAdd": true,
  "maxReposToSearch": 50,
  "secretKey":
  "update.channel": "release",
  "github": [
    {
      "token":
      "url": "https://github.com"
    }
  ]
}
```

Fig. 3: Sourcegraph

This kind of solution can provide the most potent online IDE to users, making it almost the same as an offline development environment. But on the other hand, it cost a lot of resources to support this type of solution, because to use the system, every user will need a complete runtime environment individually. And in terms of open-source code, the user may merely want to browse and look up the source code repository to learn how to implement that product or figure out the root cause of some defect occurred during their implementation. For this kind of user requirement, it would be to luxury to provide an entire IDE environment to users, and that would cost too much to provide service to tremendous of users from all over the world.

Non-container solution: zGrepCode[33] This product is almost the ideal solution for our requirement. It firstly analysis the source code and index them, then provide online access for users to browse and navigate among different java files. It does not provide any customised container to users; instead, it uses a general server to resolve all links by querying the index generated previously.



```
    /**
     * Returns x's Class if it is of the form "class C implements
     * Comparable<C>", else null.
     */
    static Class<?> comparableClassFor(Object x) {
        if (x instanceof Comparable) {
            Class<?> c; Type[] ts, as; ParameterizedType p;
            if ((c = x.getClass()) == String.class) // bypass checks
                return c;
            if ((ts = c.getGenericInterfaces()) != null) {
                for (Type t : ts) {
                    if ((t instanceof ParameterizedType) &&
                        ((p = (ParameterizedType) t).getRawType() ==
                         Comparable.class) &&
                        (as = p.getActualTypeArguments()) != null &&
                        as.length == 1 && as[0] == c) // type arg is c
                        return c;
                }
            }
        }
        return null;
    }
```

Fig. 4: zGrepCode

But zGrepCode does not provide users with the ability to customise their own configurations and preferences. For example, user can not nominate some specific version of JDK or any other open-source repository with specified version id, which makes it difficult for users to build their own scope of the repository. Developers may want a repository with java 8 and JUnit 3, if this scope is not pre-designed by zGrepCode, users will have no way to config it by them self.

Even worse, this product has been out of service for a few months, and there is no indication whether it will be available in the future.

2.2 Compilation and execution of Java program

Other languages like Python and JavaScript have a lot of existing tools to use. How ever, there are not many tools providing a similar function for Java, because it is more challenging to implement a navigation system for java code, because of the complexity of its compilation and execution process.

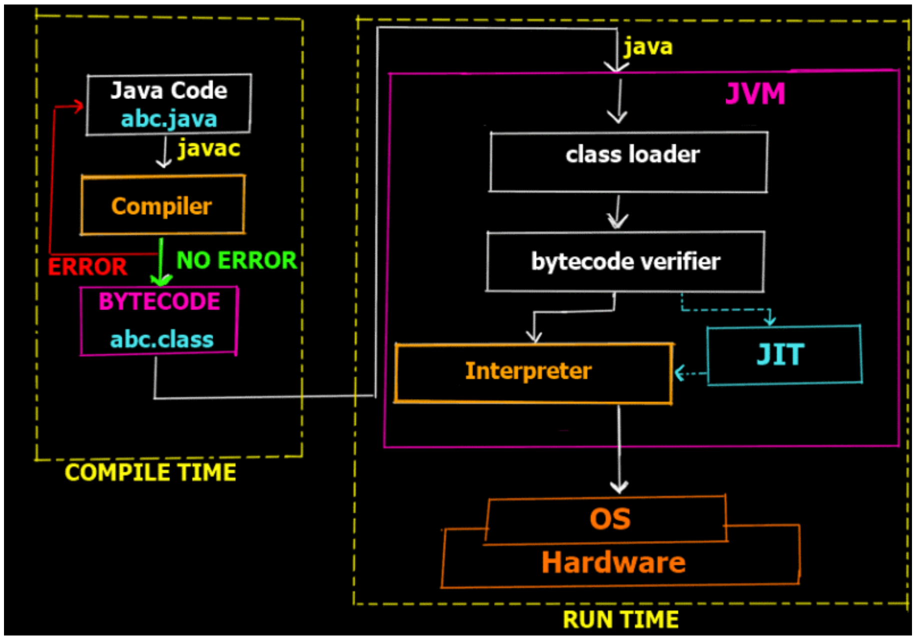


Fig. 5: Execution process of Java program in detail[18]

There are two processes working individually, which is the compiled time and run time. In each process, the original java code is processed separately with different rules, and the behaviour of type navigation that presented to the user is a combination of the result from both processes. If we want to implement a source code navigation system without compile time and run time, a general rule for navigation is required to build this system, but currently, there is no relevant research that purposed such kind of general rules.

2.3 General requirement

Base on the research mentioned above, this project was proposed by Red Hat, which aimed to develop a browser-based source code navigation system for open-source developers and their customers. This navigation system should meet the following requirements:

1. User should be available to browse all Java codes from the repositories online.
2. The system should allow users to config customised classpath which defines some specific versions of dependencies to involve in the current scope of code.
3. The code displayed online should be syntax highlighted so that the user can easily browse the code on the webpage.
4. This product should be available to provide service to multiple users at the same time and at the same time, cost as less computing resource as possible.
5. The code should be efficient enough to:
   (a) Provide cost-efficient operation of the bulk rendering component, with a time budget of 10ms per input source java file on a single CPU core.
   (b) Provide responsive user interaction for the link resolution, with a time budget of 50ms for returning a response, excluding network RTT, given a single concurrent user on a single CPU core with 2GB memory.

3 Comparative Research

There are a lot of different open-sourced projects that can help develop a browser-based source code navigation system, so before designing this project, several products were evaluated, and a general comparison can be found as follow.

3.1 Java Parsing

Without the process of a specific parser, the raw java file is just a combination of characters, so firstly it is essential to find an appropriate solution to parse the source code.

A parser is usually composed of two parts: a lexer, also known as scanner or tokenizer, and the proper parser. Not all parsers adopt this two-steps schema: some parsers do not depend on a lexer. They are called scannerless parsers.[27]

- **Lexer + Parser solution:** AnnoFlex[4] + Antlr[5]
  AnnoFlex Bases on annotation (own JavaDoc labels) to produces the matching tokens, and then use Antlr to generate an AST.
  This solution is a little bit complex, and the improper JavaDoc labels will also cause syntax warning from syntax highlight tool and IDE.

- **CFG(Context-free grammars) solution:** Antlr[5]

  This kind of solution is widely used to generate parsers. You can define your own grammar and use it to create a parser that works for it. It is a basic low-level tool, but for our project, the language is predefined for Java only, so there is no need to spend a lot of effort on creating our own parser.

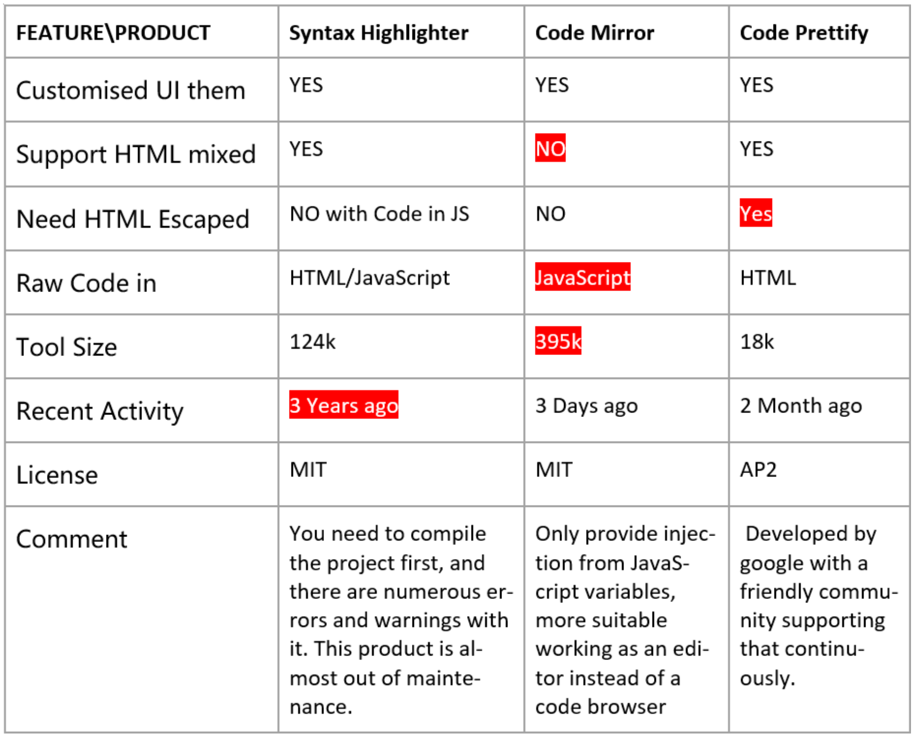
- **PEG(Parsing Expression Grammar) solution:** JavaParser[25]

  Java Parser is A well-organised parser derived from JavaCC. It supports all versions of java from 1 to 12, and it also supports lexical preservation and pretty printing. It is also Eligible to use with JavaSymbolSolver, which can boost up our project to save a lot of effort to implement such kind of functions.

Obviously, for this project, the JavaParser would be the best solution to implement our own parsing functions.

3.2 Syntax Highlight

In order to prettify the code with syntax highlight and make it more similar to a local IDE, this project requires a tool to provide syntax highlight. There are three products been evaluated during this project, which is Syntax Highlighter[31], Code Mirror[14], Code Prettify[13].



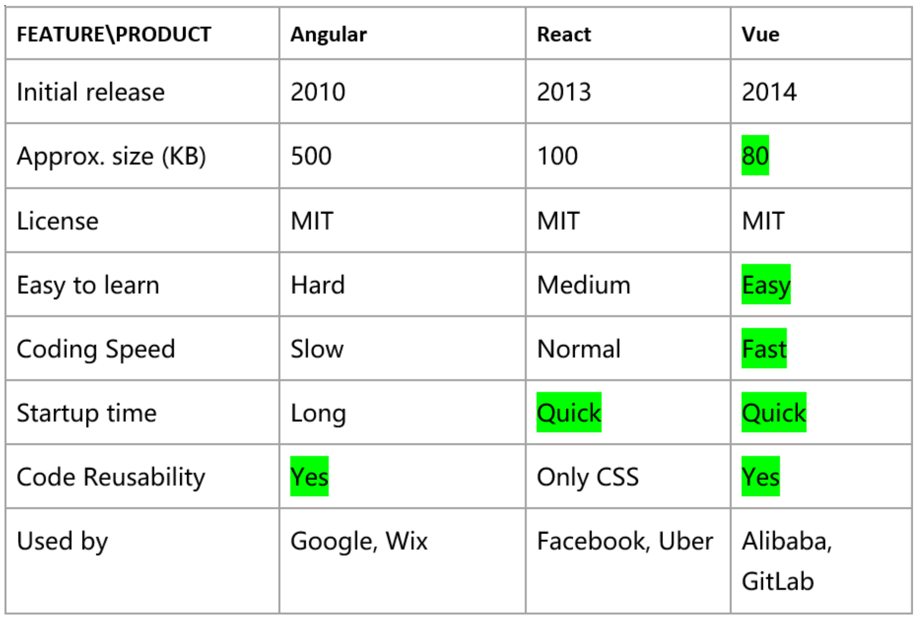
FEATURE\PRODUCT	Syntax Highlighter	Code Mirror	Code Prettify
Customised UI them	YES	YES	YES
Support HTML mixed	YES	NO	YES
Need HTML Escaped	NO with Code in JS	NO	Yes
Raw Code in	HTML/JavaScript	JavaScript	HTML
Tool Size	124k	395k	18k
Recent Activity	3 Years ago	3 Days ago	2 Month ago
License	MIT	MIT	AP2
Comment	You need to compile the project first, and there are numerous errors and warnings with it. This product is almost out of maintenance.	Only provide injection from JavaScript variables, more suitable working as an editor instead of a code browser	Developed by google with a friendly community supporting that continuously.

Fig. 6: Metrix of different syntax highlighting tools

After the comparison between all three products, it is more appropriate to integrate Code Prettify into our project to provide a syntax highlight feature.

3.3 Front-end Framework

This navigation system provides a browser-based user interface for the developer to access, at the same time send, receive and process information during the entire resolving stage. Thus, a front-end framework is especially important.



FEATURE\PRODUCT	Angular	React	Vue
Initial release	2010	2013	2014
Approx. size (KB)	500	100	80
License	MIT	MIT	MIT
Easy to learn	Hard	Medium	Easy
Coding Speed	Slow	Normal	Fast
Startup time	Long	Quick	Quick
Code Reusability	Yes	Only CSS	Yes
Used by	Google, Wix	Facebook, Uber	Alibaba, GitLab

Fig. 7: Metrix of front-end frameworks

In terms of this project, the front-end logic is not complicated at all, so a light front-end framework like Vue.js would be the best choice.

4 High-level Architecture

According to what required in paragraph 2.3, this system should adopt a non-container solution to provide service to as many users as possible.

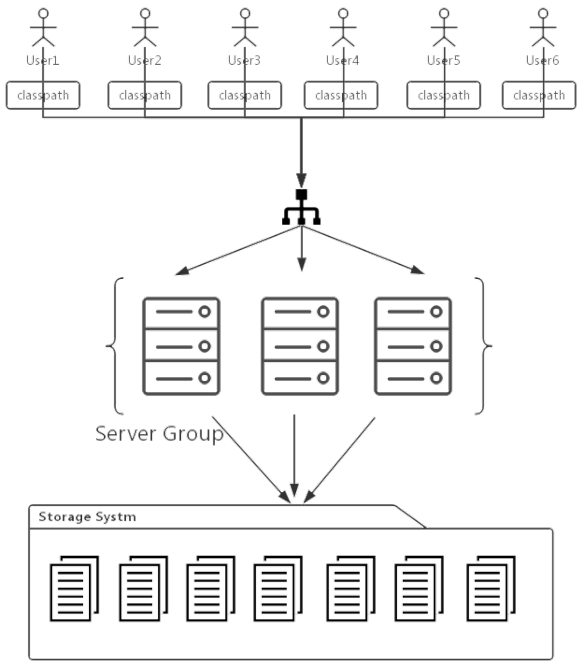


Fig. 8: High level user access

But with a non-container solution, it is impossible to use JVM to resolve the symbol in realtime. Instead, different users should be able to access a general server(or server group) that uses a unified storage system to look up all relevant information required to resolve the navigation result(see Fig. 8). Thus, the navigator is designed as a two-stage system:

- **Stage1: Analysis Process**
  In this stage, the system scans all java codes in the .jar files within the repository, converts them into Html document. Every Java types are wrapped with hyperlink tags so that all types(classes, interfaces, annotations) within this Html file can be easily recognised by the user, and the user can click the hyperlink to request for navigation to redirect to the definition of this specific type.
  To make it possible for the server to navigate between different Java files, several supplementary documents are also generated in this stage, including explanatory JSON files and index files.
- **Stage2: Resolving Service**
  The resolving service is designed to process the resolving request passed from the front-end user interface. When the user clicks any link on the webpage,

the front-end JavaScript will send a request to resolving server with some parameters that describe where the user is navigating from and the name of the type that they want to access.

When the server receives the request form user browser, it will find all targets within the index file and find out the right target according to the rule that applied by Java compiler and JVM. Also, the server will filter out those targets not included in the user-customised classpath and return a result to the front-end whether there is no result within the repository or the result is blocked by classpath restriction.

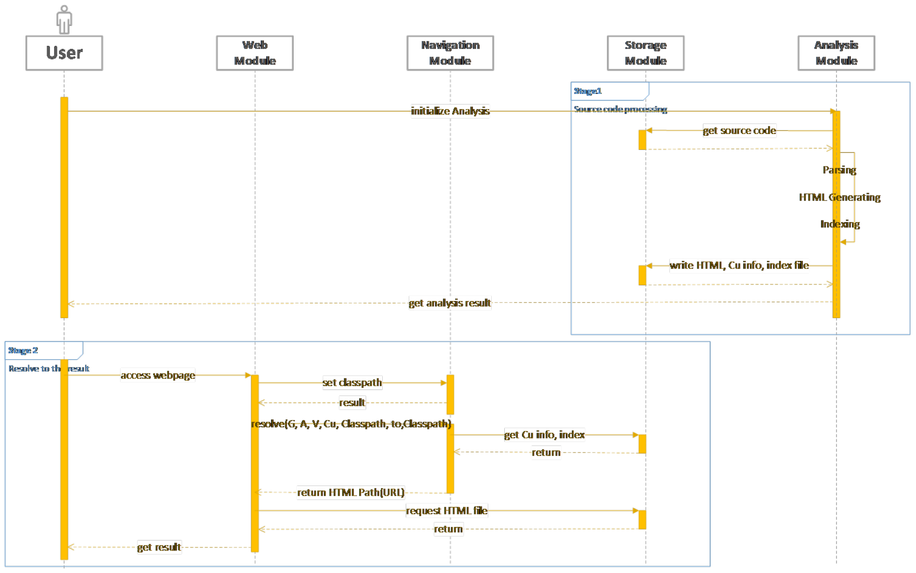


Fig. 9: UML of the system

With this architecture(Fig. 9), all source codes in the jar files are proceeded at the same time during the analysis stage and saved to the storage system, making it possible to provide a general resolving service in high efficiency and low expenditure.

5 Source code analysis

There are several steps within the analysis stage, which can be briefly introduced by Fig. 10.

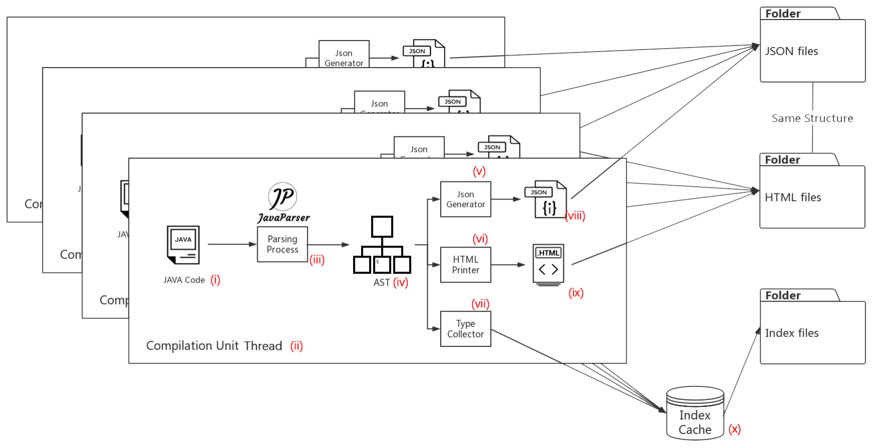


Fig. 10: UML of the system

5.1 Source code to AST

The source code for a Java class is organized into compilation units, and a simple compilation unit contains a single class definition and is named for that class[35], so firstly, the system will read every jar file within the target repository, and process every java file(i) individually in the compilation unit thread(ii). The java code will be parsed by the parsing process(iii), then generate an AST(iv) that stores all relevant information of this java file.

The AST (Abstract Syntax Tree) is a tree representation of the abstract syntactic structure of source code written in a programming language.[1] For example, "437+734" can be processed by the following steps:

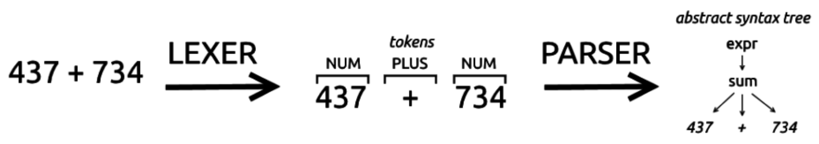


Fig. 11: Example of how lexer and parser works[27]

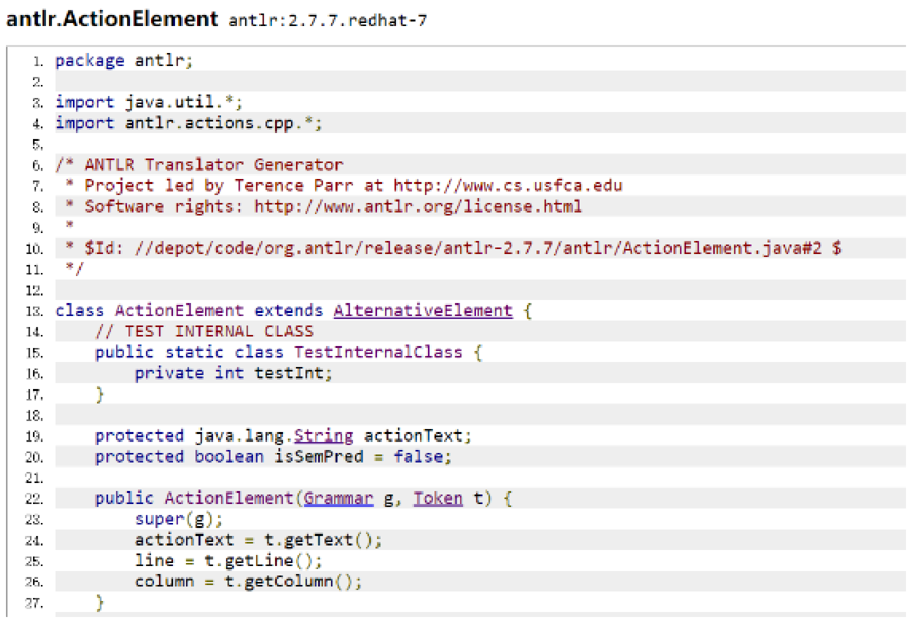
Firstly, the lexer will recognise different tokens and marks them with the meaning of these tokens, and then the parser will generate an abstract syntax tree which stores this information into every individual node. With the tree structure, we can easily trace between different nodes and find out the relation between different nodes.

This project uses Java Parser to analysis the code and generates the AST structure of the code. The JavaParser library allows you to interact with Java source code as a Java object representation in a Java environment. More formally, it refers to this object representation as an Abstract Syntax Tree (AST).[38]

5.2 AST to Html

When we got an entire AST(iv) that describe everything within specific compilation unit, the Html Printer(vi) will transform the AST(iv) into an Html file(ix), because basically the AST[1](iv) stores everything that comes from the raw Java file. The only difference is that the Html Printer[6] need to find out all the occurrence of every class and interface type that is used in within the code and then wrap it with "¡a¿" tag, so that these types will be displayed as hyperlinks for the user to click. See Appendix 1 for the relevant code. Also, Code Prettify[13] is integrated with the Html file(ix) generated by Html Printer(vi).

After this process, we can generate a single Html file(ix) that display all the java code from the raw java file(ii), like what is shown in the Fig. 12.



antlr.ActionElement antlr:2.7.7.redhat-7

```
package antlr;

import java.util.*;
import antlr.actions.cpp.*;

/* ANTLR Translator Generator
 * Project led by Terence Parr at http://www.cs.usfca.edu
 * Software rights: http://www.antlr.org/license.html
 *
 * $Id: //depot/code/org.antlr/release/antlr-2.7.7/antlr/ActionElement.java#2 $
 */

class ActionElement extends AlternativeElement {
    // TEST INTERNAL CLASS
    public static class TestInternalClass {
        private int testInt;
    }

    protected java.lang.String actionText;
    protected boolean isSemPred = false;

    public ActionElement(Grammar g, Token t) {
        super(g);
        actionText = t.getText();
        line = t.getLine();
        column = t.getColumn();
    }
```

Fig. 12: Webpage generated by AST

5.3 AST to JSON file

A single Html file(ix) can help display the source code that stores in the java file(ii), but if the user wants to resolve the links to get the right target result, some necessary information should be generated and saved into storage system together with the Html file(ix).

The JSON Generator(v) is a function that retrieves type declarations, import declarations and package declarations from the AST(iv), then save them as a JSON file(viii) in the storage system. For every Html file(ix), there is a JSON file(viii) with the same filename and same storage path with the Html file(ix). Here is an example of how JSON file looks like.

5.4 AST to Index file

If we want to resolve specific link that contains the target type name, we need to generate an index system that works as a dictionary to help the resolver know how much possible targets are there in the entire repository. So within the analysis stage of every compilation unit thread(i), the type collector(vii) will retrieve full qualified type name from the type declaration in the AST(iv) and save it to a hash map in the index cache(x).

After all jar files have been analysed, the system will iterate every item in the cache and save them into different index file named with the type name. See Appendix 1 for the relevant code.

6 Navigation

After we analysis all Java files within the repository, the system should be available to provide a navigation service for front-end browser to redirect between different pages.

6.1 Universal Unique Type Name(UUTN)

It is possible that two different java files from different jar files share the same fully qualified type name. For example, "java.lang.String" may come from source code of either Java12 or Java8. In order to navigate between different java files, we need to design a file storage system that copes with the co-existence of the Java files from different versions of source code.

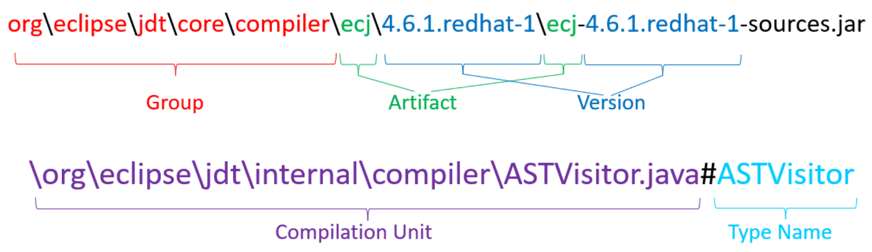

org\eclipse\jdt\core\compiler\ecj\4.6.1.redhat-1\ecj-4.6.1.redhat-1-sources.jar
Group
Artifact
Version
\org\eclipse\jdt\internal\compiler\ASTVisitor.java#ASTVisitor
Compilation Unit
Type Name

Fig. 13: Full path of a specific Java type

The Fig. 13 shows an example of a universal unique file name of a specific Java type. It consists of five factors:

- The group id of the project;
- artefact id of this project;
- The version of this product;
- The fully qualified name of this compilation unit;
- The type name of this type.

Since both Html files and JSON files are saved in this path and file name, you can locate any Java type with the entire repository with these five factors.

6.2 Resolve Function

When user trying to navigate from specific Java file to another Java file that via the hyperlink, it should pass the UUTN of current Java file (which is regarded as "navigate from") and the target type name(which is regarded as "navigate to") to the backend server. A typical resolve request may look like this:

```
Resolver.resolve(groupId, artifactId, version,
    compilationUnit, typeName, navTo){/*...*/}
```

Listing 1: Resolver example

The groupId, artifactId, version, compilationUnit and typeName parameter are exactly the UUTN of where you navigate from, and the navTo represent the String of the type name of your target.

6.3 Priorities for Resolving

As what was mentioned in paragraph 2.2, there is no existing research defines a general rule of how to resolve the type name and navigate to the right target Java type that is quoted by current scope. Thus a cross-validation test was conducted to all different kinds of references.

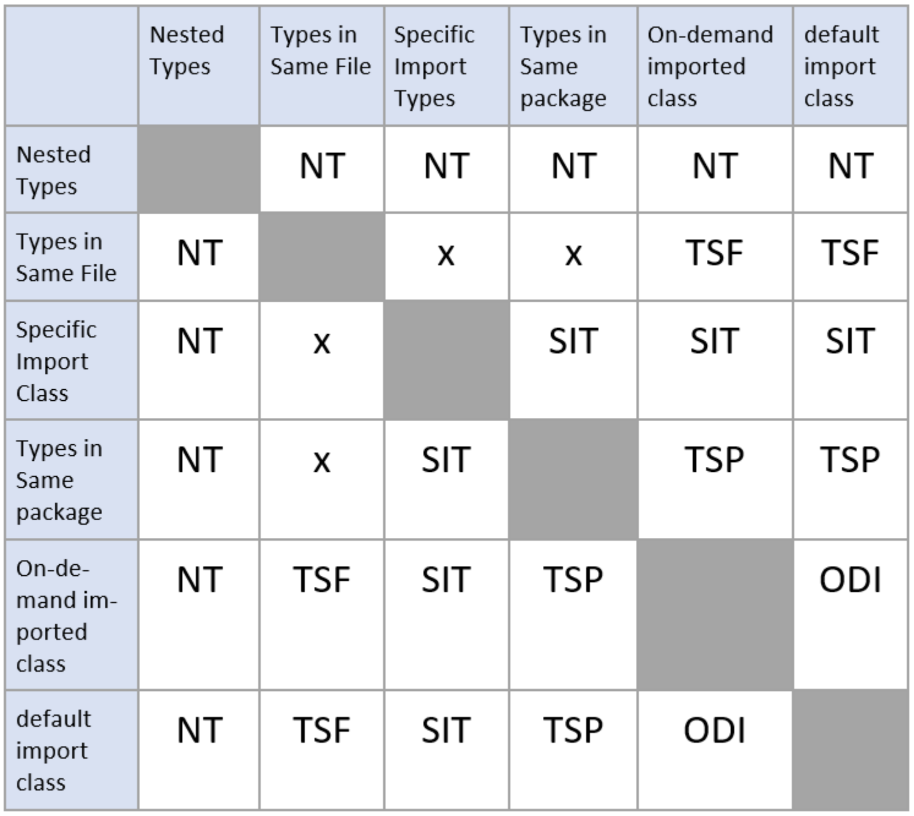
This test is designed some cases with different type with same type name but imported differently. For example, in Listing 1.2, we explicitly imported class Type from package org.myproj, and at the same time declared a nested type named Test as well, then we compile and run the code to figure out which class is actually used by current type.

```
package Test1;

//This is a specific imported class Test;
import org.myproj.Test;

public class ImportTest {
  Test test = new Test();
```

```
    class Test {
      // This is a Nested Class Test;
    }
  }
```

Listing 2: Sample code to show the complexity of Java

The result of the cross-validation is shown as below:



	Nested Types	Types in Same File	Specific Import Types	Types in Same package	On-demand imported class	default import class
Nested Types		NT	NT	NT	NT	NT
Types in Same File	NT		x	x	TSF	TSF
Specific Import Class	NT	x		SIT	SIT	SIT
Types in Same package	NT	x	SIT		TSP	TSP
On-demand imported class	NT	TSF	SIT	TSP		ODI
default import class	NT	TSF	SIT	TSP	ODI	

*NT: Nested Types, TSF: Types defined in the Same File, SIT: Specific Imported Types, TSP: Types in the Same Package, ODI: On-demand Imported types

**Result marked with x means this kind of combination is impossible to occur.

Fig. 14: Cross-validation result

According to the result of the cross-validation, there are seven steps to resolve any target of Java types:

1. Find if it has the same name with current Type (Class / Interface) itself;

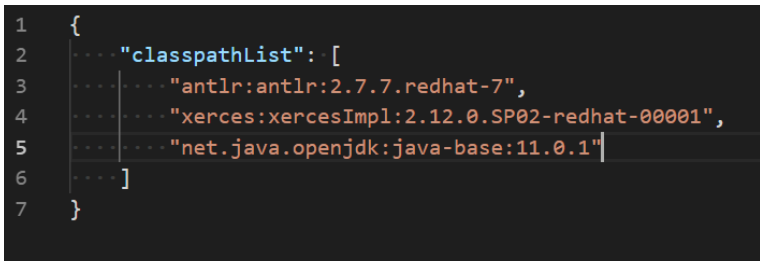
2. Find all nested Types and Types that defined in same java file;
3. Find specific-imported type;
   e.g., import uk.ac.ncl.cs.tongzhou.navigator.learnning.testpackage.Date;
4. Find Types defined in the same package;
5. Find on-demand imported types;
   e.g., import uk.ac.ncl.cs.tongzhou.navigator.learnning.testpackage.*;
6. Find in all default imported packages , e.g., String from java.lang
7. Find directly nominated type, e.g., java.util.Date date=null.

These seven rules can help the resolver to find out the right target without Java compiler or JVM, which gives an unobstructed waterflow to resolve some specific type name into the correct target that been referred by current Type.

6.4 User-customised classpath

In paragraph 6.1, we introduced UUTN(Universal Unique Type Name), which is a method that uses group id, artefact id, version, compilation unit name and type name to find any type within the system. But if you want to locate a specific jar file, you only need G(Group), A(artefact), V(version) to specify a particular version of some open-source project.

Classpath information is just a set of GAVs that defines which jar files are included in the customized scope the user wants to use during the resolving process. Since the classpath list might be pretty long, users need to save their own classpath to the server so that they can use it every time they access the system.

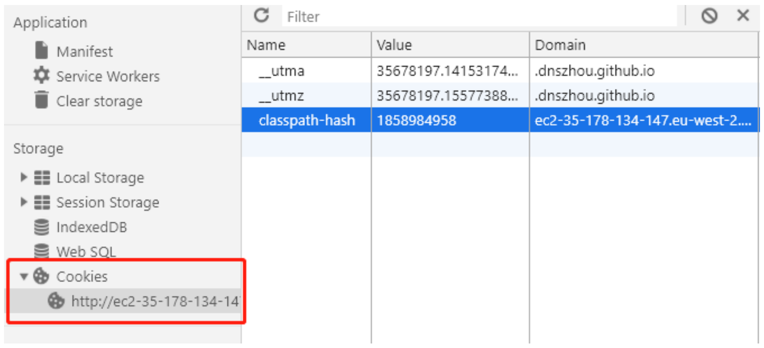


```
{
    "classpathList": [
        "antlr:antlr:2.7.7.redhat-7",
        "xerces:xercesImpl:2.12.0.SP02-redhat-00001",
        "net.java.openjdk:java-base:11.0.1"
    ]
}
```

Fig. 15: Sample of a classpath Json file

However, as was shown in Fig. 15, different users share a single server (or server group), if the server uses a general classpath file, the operation from different users may have interference with each other.

To solve this problem, when the user uploads their customised classpath, the system will actually generate the hash code from the classpath, name the classpath file with this hash code and save the file into the storage system, then finally send the response back to the browser with this hash code. The front-end script will create a new item in the cookie named “classpath-hash” and save the hash code to the value.



Name	Value	Domain
__utma	35678197.14153174...	.dnszhou.github.io
__utmz	35678197.15577388...	.dnszhou.github.io
classpath-hash	1858984958	ec2-35-178-134-147.eu-west-2....

Fig. 16: Hash code saved in cookie storage

Then every time the front-end trying to resolve any link, it will bring the cookie information to the server. The server can find the right classpath file according to the cookie information with the request, then resolve the link correctly according to the classpath information.

7 Cloud Solution

To provide service to users from all over the world, a good cloud structure is pretty crucial during both deployment stage and development stage. There are different kind of solutions to choose from. All these solutions have their advantages and disadvantages. Thus an evaluation of these solutions was conducted during this project.

7.1 EC2 + EBS

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) cloud.[9]

Amazon Elastic Block Store (EBS) is an easy to use, high-performance block storage service designed for use with Amazon Elastic Compute Cloud (EC2) for both throughput and transaction-intensive workloads at any scale.[7]

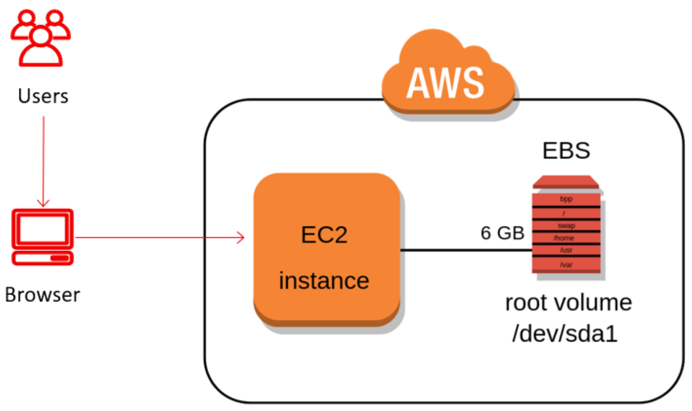
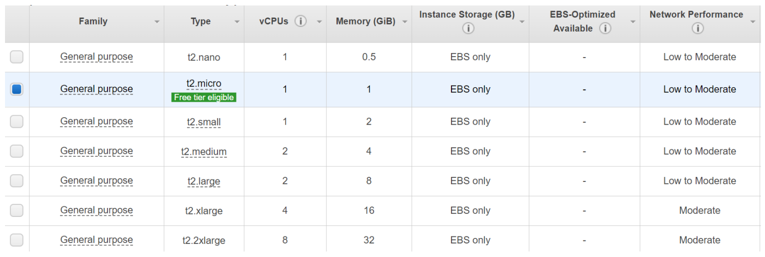


Fig. 17: EC2 + EBS solution

This is the easiest cloud solution to implement because using EC2 and EBS is just like using a virtual machine with a high-speed hard disk volume automatically mounted to the system.

To examine the solution we chose a free-tier server type with one core CPU and 1GiB memory, according to the requirement defined in paragraph 2.3-5b.



	Family	Type	vCPUs	Memory (GiB)	Instance Storage (GB)	EBS-Optimized Available	Network Performance
☐	General purpose	t2.nano	1	0.5	EBS only	-	Low to Moderate
☑	General purpose	t2.micro Free tier eligible	1	1	EBS only	-	Low to Moderate
☐	General purpose	t2.small	1	2	EBS only	-	Low to Moderate
☐	General purpose	t2.medium	2	4	EBS only	-	Low to Moderate
☐	General purpose	t2.large	2	8	EBS only	-	Low to Moderate
☐	General purpose	t2.xlarge	4	16	EBS only	-	Moderate
☐	General purpose	t2.2xlarge	8	32	EBS only	-	Moderate

Fig. 18: Diiferent configurations of AWS EC2 instances

This solution provided a nice performance(see paragraph 8.3-1), which is the best one among all the solutions under the performance test because EBS is a high-performance block storage service that works better than traditional hard disk.

But with this solution the cost of storage may become really high, because Amazon EBS pricing is based on its use for high-performance workloads, it can be expensive for data that doesn't require access via highly performant disks.[17] It is also tricky for system administrator to increase the storage if there are new source code to be analysed in the repository.

Another disadvantage of this solution is that we can not integrate EBS volume with CDN(Content Delivery Network)[15] system, which may cause a latency when global users are trying to access our system.

7.2 EC2 + S3

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance.[3]

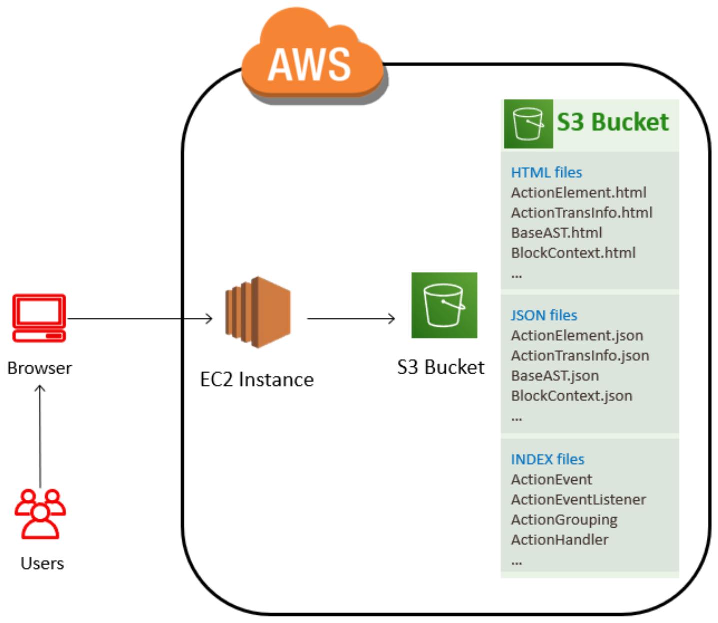


Fig. 19: EC2 + S3 solution

With this solution, all data were stored in S3 bucket, and the program within EC2 instances access S3 directly to retrieve relevant data.

Since S3 can be easily integrated with CDN service, this solution can ensure the accessibility are equally from all over the world.

But actually, the cost of this solution may increase exponential, because of the pricing rules of S3 product. (see Appendix 3) It not only charges for storage but also charges for requests that exchanged with S3 bucket, no matter the request comes from EC2(internal AWS network) or from user browser.

Another problem of S3 solution is about the performance, the access from EC2 to S3 is actually worse than the access to EBS(see paragraph 8.3), because EBS is directly attached to EC2, and for S3 there are a lot of validation rules which cost more time than EBS.

7.3 Lambda + API Gateway + S3

Another popular solution is the serverless architecture that using API Gateway[2] to accept Http requests and redirect to AWS Lambda[10] service to process and interact with the S3 storage system.

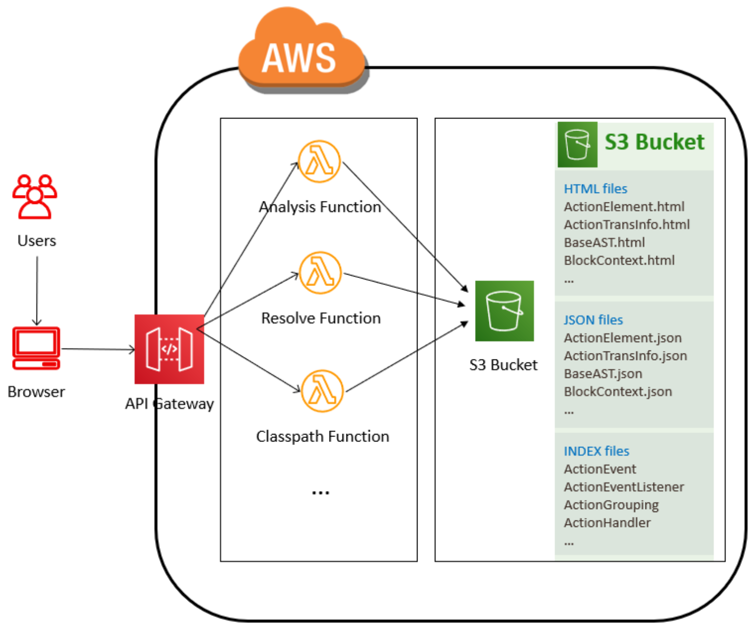


Fig. 20: Lambda + API Gateway + S3 solution

AWS Lambda lets you run code without provisioning or managing servers. You pay only for the compute time you consume - there is no charge when your code is not running.

Although Lambda supports a lot of different programming language, it does not work perfectly with Java code. You can not debug Java code online like Node.js or other supported languages, and the platform support only Java version 8, while our project was developed with Java12.

Also, it is challenging to apply CI(Continuous Integration)[16] tools to Lambda services because all the codes are separately in different Lambda functions.

7.4 EC2 + EBS + S3

After investigation EC2, EBS, S3, Lambda and other relevant cloud solutions, we decided to build a hybrid architecture that takes advantages from both S3 solution and EBS solution, which is shown in following diagram.

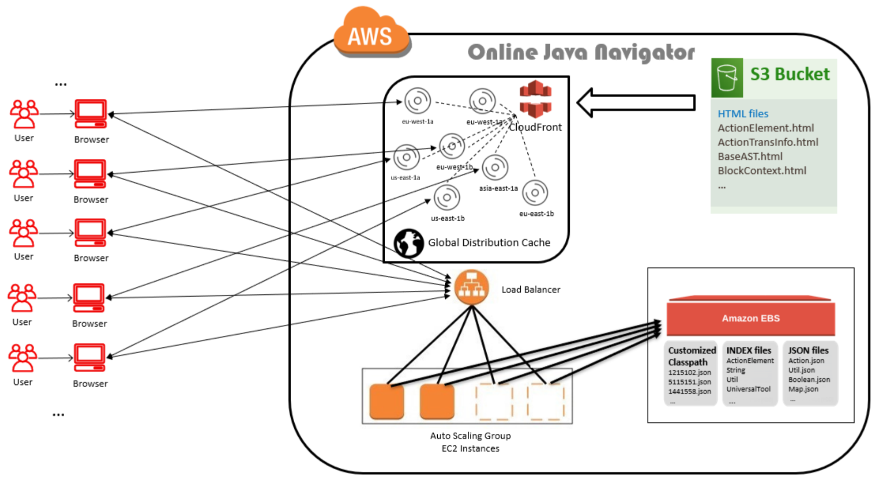


Fig. 21: Topology of Online Java Navigator

As is presented in Fig. 21, there are different steps to finish the entire process:

1. Different users can use their own browser to access our website;
2. When user click the link, the browser send a request to the load balancer;
3. The load balancer will find a server with the lowest capacity to process the request
4. The server within the EC2 instance retrieves the right classpath file from EBS
5. The server looks up the index file and JSON file in EBS to resolve the link
6. The server returns the right path of the type back to the front-end browser
7. The Browser retrieves the target Html file from the global distribution cache that generated from the S3 bucket.

If all EC2 instances are under high capacity, the auto-scaling group will initialise another EC2 instance and connect it to the load balancer, which makes sure that every user can access the system with highest accessibility.

To enable this kind of auto-scaling structure, the system is required to be designed as a stateless program. Every time a user resolve any type in our system, the request send to backend actually have everything needed for resolver to get the result, which means every request is independent with each other.

Also, with this structure, the number of access request from EC2 to S3 are minimised. With the EC2 + S3 solution introduced in paragraph 7.2, for every request there will have at most three get request directed to S3 bucket, which is retrieving the JSON file, the index file and the Html file. With Hybrid solution, the only request to S3 bucket is the get request for Html file from user browser, which means the amount of the request has been reduced by two-thirds. As we discussed in paragraph 7.2, the access to S3 is actually slower than the access to EBS, so this hybrid solution obviously improves the performance of this system(see paragraph 8.3 for the testing result).

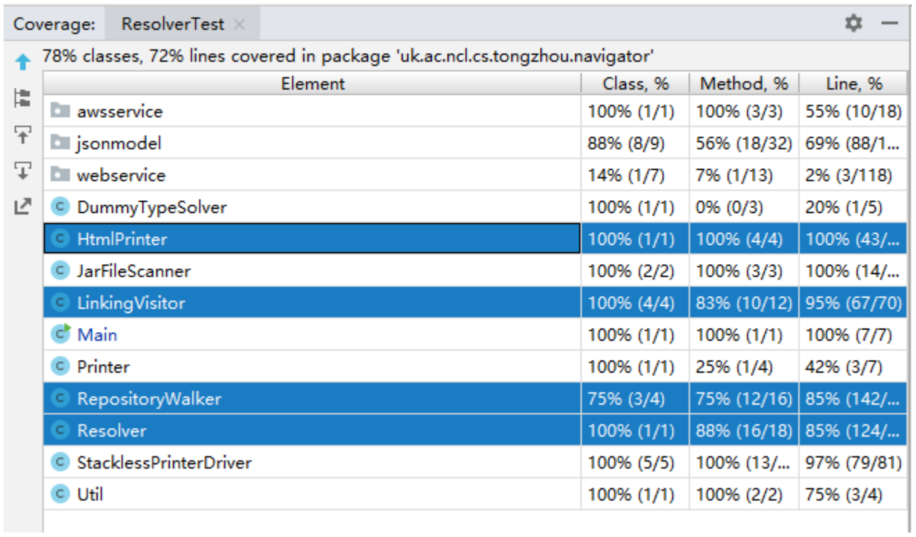
On the other hand, since AWS charge the S3 service by the amount of the request that sends to S3, this hybrid solution saves many budgets and at the same time achieves even better performance.

8 Evaluation

8.1 Unit Testing & Integrate Testing

Unit testing is a software testing method by which individual units of source code, sets of one or more computer program modules together with associated control data, usage procedures, and operating procedures, are tested to determine whether they are fit for use.[37] Junit tool was integrated into this project to perform unit testing and integrate testing during the development stage.

Different type of test cases was designed to cover as much code as possible. In Fig. 22 it shows the coverage of the code that was involved by Junit test cases, and most of those core classes that marked in blue were covered in a high coverage.


Coverage: ResolverTest

78% classes, 72% lines covered in package 'uk.ac.ncl.cs.tongzhou.navigator'

Element	Class, %	Method, %	Line, %
awsservice	100% (1/1)	100% (3/3)	55% (10/18)
jsonmodel	88% (8/9)	56% (18/32)	69% (88/1...
webservice	14% (1/7)	7% (1/13)	2% (3/118)
DummyTypeSolver	100% (1/1)	0% (0/3)	20% (1/5)
HtmlPrinter	100% (1/1)	100% (4/4)	100% (43/...
JarFileScanner	100% (2/2)	100% (3/3)	100% (14/...
LinkingVisitor	100% (4/4)	83% (10/12)	95% (67/70)
Main	100% (1/1)	100% (1/1)	100% (7/7)
Printer	100% (1/1)	25% (1/4)	42% (3/7)
RepositoryWalker	75% (3/4)	75% (12/16)	85% (142/...
Resolver	100% (1/1)	88% (16/18)	85% (124/...
StacklessPrinterDriver	100% (5/5)	100% (13/...	97% (79/81)
Util	100% (1/1)	100% (2/2)	75% (3/4)

Fig. 22: Coverage analysis of the test cases

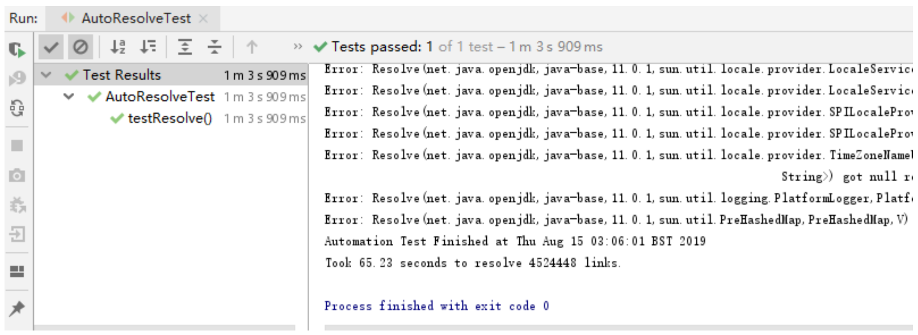
During the development stage of the project, these test cases also worked as a kind of regression test. Regression testing is re-running functional and non-functional tests to ensure that previously developed, and tested software still performs after a change.[34] Since different functions are developed separately, and different functions may have internal relationship with each other, it is especially important to make sure that newly developed functions do not ruin any existing function in the system.

8.2 Automation Testing

Since this navigator generates links for every Type that occurred in the java code, it is important to know how much links can be resolved appropriately, which can reflect the accuracy of the resolving system.

Thus an automation testing program was developed for this product. As what is shown in the Fig. 10 in paragraph 5, the AST was generated by parsing process and contained all relevant information of the source code, so the server also produced a list of test cases that describe every available link as G,A,V,Cu,TypeName and NavigateTo, and save them into CSV files.

In the Junit test folder, there is a testing program that loads all the test cases and passes it to resolving function, so that it can cover every link in the system and test them automatically.


Run:
AutoResolveTest
Tests passed: 1 of 1 test – 1 m 3 s 909 ms
Test Results 1 m 3 s 909 ms
AutoResolveTest 1 m 3 s 909 ms
testResolve() 1 m 3 s 909 ms

```
Error: Resolve(net.java.openjdk, java-base, 11.0.1, sun.util.locale.provider.LocaleServic
Error: Resolve(net.java.openjdk, java-base, 11.0.1, sun.util.locale.provider.LocaleServic
Error: Resolve(net.java.openjdk, java-base, 11.0.1, sun.util.locale.provider.SPILocalePro
Error: Resolve(net.java.openjdk, java-base, 11.0.1, sun.util.locale.provider.SPILocalePro
Error: Resolve(net.java.openjdk, java-base, 11.0.1, sun.util.locale.provider.TimeZoneNamel
                                                                     String>) got null r
Error: Resolve(net.java.openjdk, java-base, 11.0.1, sun.util.logging.PlatformLogger,Platf
Error: Resolve(net.java.openjdk, java-base, 11.0.1, sun.util.PreHashedMap,PreHashedMap,V)
Automation Test Finished at Thu Aug 15 03:06:01 BST 2019
Took 65.23 seconds to resolve 4524448 links.

Process finished with exit code 0
```

Fig. 23: The result of the automation test

8.3 Performance Testing

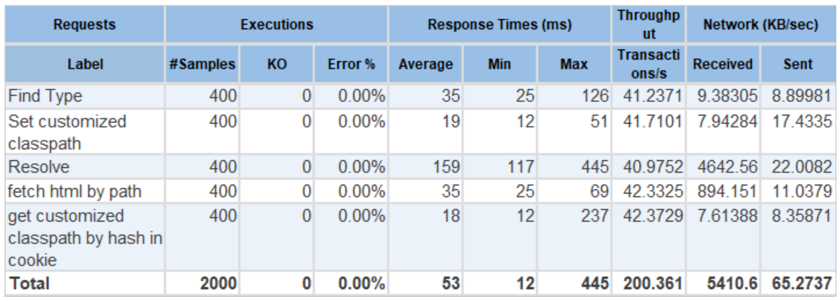
Performance testing is, in general, a testing practise performed to determine how a system performs in terms of responsiveness and stability under a particular workload.[30] Since this project is designed to provide service to different users at same time, so it is especially important to examine how the system works when multiple user access the system.

To carry out this test, JMeter[36] tool is used during the performance testing stage. Apache JMeter is an Apache project that can be used as a load testing tool for analysing and measuring the performance of a variety of services, with a focus on web applications.[6] We set the testing parameter as follow:

– Threads: 20;

– Ramp-Up Time: 5 seconds;
– Loops: 20

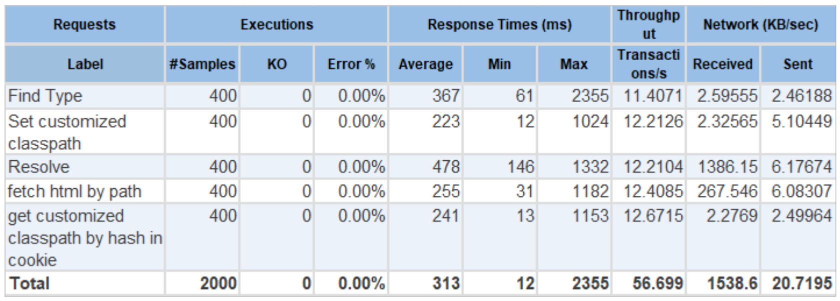
Which means the JMeter will initial 20 user threads within 5 seconds, and it will test every API for 20 times. During the testing stage, three different solutions were tested as a comparison.



Requests	Executions			Response Times (ms)			Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Transactions/s	Received	Sent
Find Type	400	0	0.00%	35	25	126	41.2371	9.38305	8.89981
Set customized classpath	400	0	0.00%	19	12	51	41.7101	7.94284	17.4335
Resolve	400	0	0.00%	159	117	445	40.9752	4642.56	22.0082
fetch html by path	400	0	0.00%	35	25	69	42.3325	894.151	11.0379
get customized classpath by hash in cookie	400	0	0.00%	18	12	237	42.3729	7.61388	8.35871
Total	**2000**	**0**	**0.00%**	**53**	**12**	**445**	**200.361**	**5410.6**	**65.2737**

Fig. 24: Result of 20 thread performance testing with EC2 + EBS solution

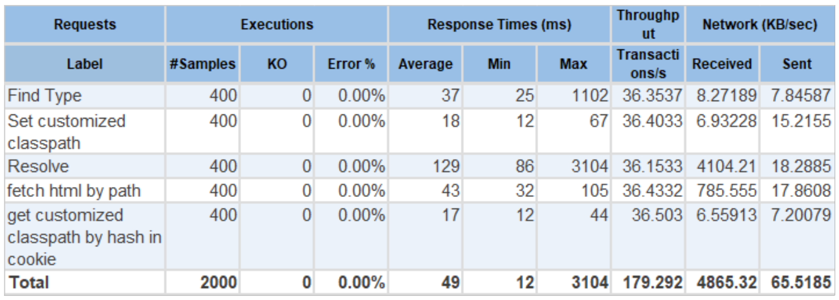
EC2 + EBS solution This solution got the best result in terms of the performance, the average response time is only 53ms, and the total throughput is 200.361 transactions per second.



Requests	Executions			Response Times (ms)			Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Transactions/s	Received	Sent
Find Type	400	0	0.00%	367	61	2355	11.4071	2.59555	2.46188
Set customized classpath	400	0	0.00%	223	12	1024	12.2126	2.32565	5.10449
Resolve	400	0	0.00%	478	146	1332	12.2104	1386.15	6.17674
fetch html by path	400	0	0.00%	255	31	1182	12.4085	267.546	6.08307
get customized classpath by hash in cookie	400	0	0.00%	241	13	1153	12.6715	2.2769	2.49964
Total	**2000**	**0**	**0.00%**	**313**	**12**	**2355**	**56.699**	**1538.6**	**20.7195**

Fig. 25: Result of 20 thread performance testing with EC2 + S3 solution

EC2 + S3 solution When the entire storage system was moved to S3, the performance became much worse than that on EBS. On average, it took 313ms to process every request, and the average throughput decreased to 56.699 transactions per second, only a quarter as much as that with EBS solution.



Requests	Executions			Response Times (ms)			Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Transactions/s	Received	Sent
Find Type	400	0	0.00%	37	25	1102	36.3537	8.27189	7.84587
Set customized classpath	400	0	0.00%	18	12	67	36.4033	6.93228	15.2155
Resolve	400	0	0.00%	129	86	3104	36.1533	4104.21	18.2885
fetch html by path	400	0	0.00%	43	32	105	36.4332	785.555	17.8608
get customized classpath by hash in cookie	400	0	0.00%	17	12	44	36.503	6.55913	7.20079
Total	**2000**	**0**	**0.00%**	**49**	**12**	**3104**	**179.292**	**4865.32**	**65.5185**

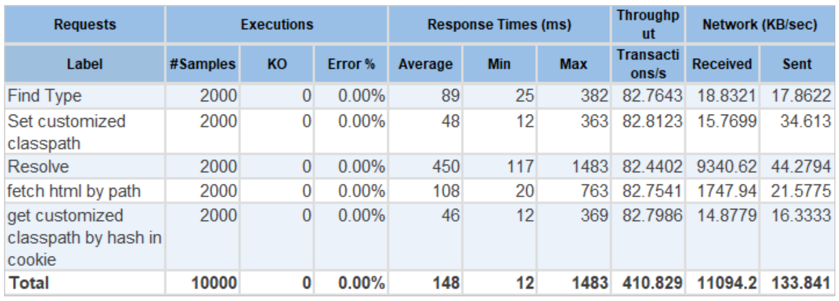
Fig. 26: Result of 20 thread performance testing with Hybrid solution

EC2 + EBS + S3 hybrid solution With a hybrid solution, the average response time was 49 ms, similar to the EBS solution. The throughput was 179.292 transactions per second, a little bit less than EBS solution, but much better than the S3 solution.

High capacity testing To examine the performance of the product under a higher capacity, another test plan was executed to both EBS solution and EBS + S3 hybrid solution.

– Threads: 100;
– Ramp-Up Time: 10 seconds;
– Loops: 20

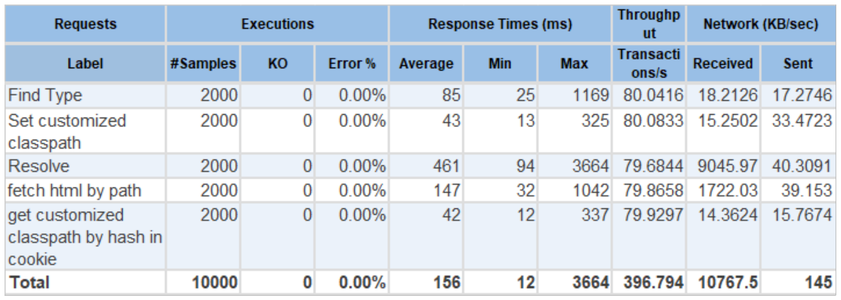
EBS solution:



Requests	Executions			Response Times (ms)			Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Transactions/s	Received	Sent
Find Type	2000	0	0.00%	89	25	382	82.7643	18.8321	17.8622
Set customized classpath	2000	0	0.00%	48	12	363	82.8123	15.7699	34.613
Resolve	2000	0	0.00%	450	117	1483	82.4402	9340.62	44.2794
fetch html by path	2000	0	0.00%	108	20	763	82.7541	1747.94	21.5775
get customized classpath by hash in cookie	2000	0	0.00%	46	12	369	82.7986	14.8779	16.3333
Total	**10000**	**0**	**0.00%**	**148**	**12**	**1483**	**410.829**	**11094.2**	**133.841**

Fig. 27: Result of 100 thread performance testing with EC2 + EBS solution

Hybrid solution:



Requests	Executions			Response Times (ms)			Throughput	Network (KB/sec)	
Label	#Samples	KO	Error %	Average	Min	Max	Transactions/s	Received	Sent
Find Type	2000	0	0.00%	85	25	1169	80.0416	18.2126	17.2746
Set customized classpath	2000	0	0.00%	43	13	325	80.0833	15.2502	33.4723
Resolve	2000	0	0.00%	461	94	3664	79.6844	9045.97	40.3091
fetch html by path	2000	0	0.00%	147	32	1042	79.8658	1722.03	39.153
get customized classpath by hash in cookie	2000	0	0.00%	42	12	337	79.9297	14.3624	15.7674
Total	**10000**	**0**	**0.00%**	**156**	**12**	**3664**	**396.794**	**10767.5**	**145**

Fig. 28: Result of 100 thread performance testing with Hybrid solution

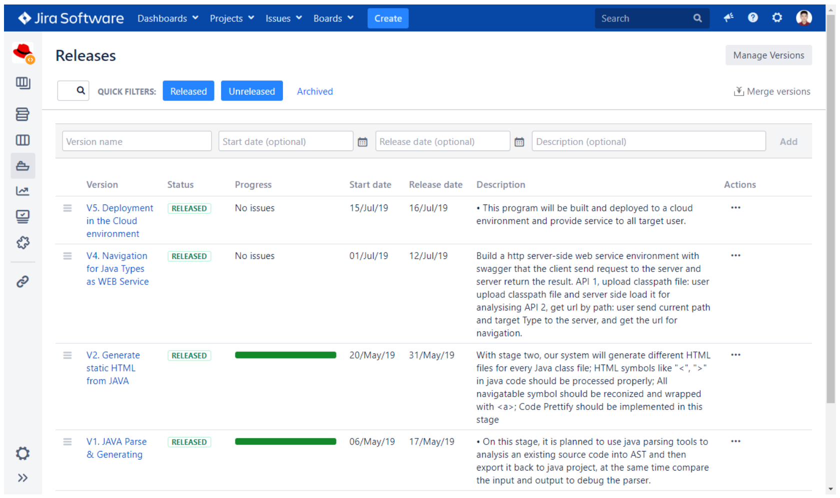
With higher capacity, both these two solutions work similarly. The Hybrid solution seems to be a little bit worse than EBS solution in terms of average response time and throughput.

8.4 Project Management

Although this project was conducted by one person only, it is still important to adopt an appropriate methodology that helps to manage the entire development process.

Traditional develop team take waterfall[32] model to manage the development life-cycle so that they can clearly research, design, develop, test and deploy the product in separate stages. This model makes the develop progress tracked explicitly, but with the development of software engineering, it takes less time to develop software products. On the other hand, the requirement of the software product is frequently changing, which is almost a disaster to traditional waterfall model.

While Scrum[29] model is such a kind of methodology that was designed to follow up the changing market. In this project, we use a professional tool, Jira[26], to help track and manage the entire development plan.



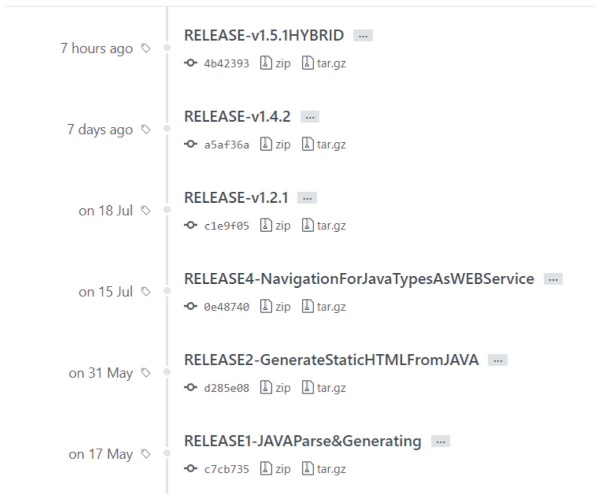
Releases

QUICK FILTERS: Released Unreleased Archived

	Version	Status	Progress	Start date	Release date	Description	Actions
☰	V5. Deployment in the Cloud environment	RELEASED	No issues	15/Jul/19	16/Jul/19	• This program will be built and deployed to a cloud environment and provide service to all target user.	•••
☰	V4. Navigation for Java Types as WEB Service	RELEASED	No issues	01/Jul/19	12/Jul/19	Build a http server-side web service environment with swagger that the client send request to the server and server return the result. API 1, upload classpath file: user upload classpath file and server side load it for analysising API 2, get url by path: user send current path and target Type to the server, and get the url for navigation.	•••
☰	V2. Generate static HTML from JAVA	RELEASED	🟩	20/May/19	31/May/19	With stage two, our system will generate different HTML files for every Java class file; HTML symbols like "<", ">" in java code should be processed properly; All navigatable symbol should be reconized and wrapped with <a>; Code Prettify should be implemented in this stage	•••
☰	V1. JAVA Parse & Generating	RELEASED	🟩	06/May/19	17/May/19	• On this stage, it is planned to use java parsing tools to analysis an existing source code into AST and then export it back to java project, at the same time compare the input and output to debug the parser.	•••

Fig. 29: Jira tool release management page for this project

In this project, there was four separate sprint which achieved different goals and published several successful releases.



7 hours ago — RELEASE-v1.5.1HYBRID — 4b42393 zip tar.gz

7 days ago — RELEASE-v1.4.2 — a5af36a zip tar.gz

on 18 Jul — RELEASE-v1.2.1 — c1e9f05 zip tar.gz

on 15 Jul — RELEASE4-NavigationForJavaTypesAsWEBService — 0e48740 zip tar.gz

on 31 May — RELEASE2-GenerateStaticHTMLFromJAVA — d285e08 zip tar.gz

on 17 May — RELEASE1-JAVAParse&Generating — c7cb735 zip tar.gz

Fig. 30: All releases delivered for this project[28]

The most crucial benefit of apply Scrum methodology comes from the flexibility of Scrum. During the project, every two weeks were regarded as an individual sprint, and after each sprint you can think over the entire project and modify the goal and tasks of next sprint.

For example, before the start of the project, we designed another sprint which aimed to deliver some edge feature like function navigation, but after sprint 2 we found that the type navigation function spent more effort than expected. In order to deliver the entire product on time, the scope and the sprint target was modified, and some edge features were discarded. If we adopted waterfall model, there would have no room to alter the scope after we started the project.

9 Conclusion

This project finished with the Online Java Navigator been developed and deployed to AWS successfully, and 97.12% of the links can be resolved, according to the automation test. There are a lot of significance and contributions achieved from this project, and also there are rooms for future improvements.

9.1 Significance and Contribution

This project delivered precisely what Red Hat customer want, an exclusive web-based source code navigator with a configurable classpath for the user to use. Currently, it is the only open-source product that providing service to all developers.

The optimized cloud solution with both EBS and S3 make it possible to provide service to as many users as possible and at the same time, reduce as much cost of cloud services as possible.

The two-stage structure of the system makes it easy to integrate the system with any other code repository. You can easily change the result of the resolving from S3 bucket to any repository link like GitHub, and then it can leverage any existing Java code browsing system with the ability to navigate from different types.

9.2 Limitation and defects

The time limit was the most significant problem because it takes much time to understand the requirement, research on different technology, develop the software product and debug it. Thus a lot of useful features, such as method navigation and integration with the third-party website, was discarded due to the limitation of the time.

Another limitation was the lack of knowledge and experience on Computational linguistics and the mechanism of Java compiling. It took much time to understand those basic concepts and theory and apply them into practice.

Also, due to some defect from the open-source product we use like Java Parser, there may have some defects that difficult to fix.

9.3 Future improvements

There are still some defects within the system, so firstly we will try to fix them in the following versions.

Also, a lot of useful functions will be developed, such as directly upload local ".classpath" file from Eclipse and load it to our system.

Last but not least, the parsing and resolving algorithms will be improved to provide better performance to the user.

References

1. Abstract syntax tree. https://en.wikipedia.org/wiki/Abstract_syntax_tree.
2. Amazon api gateway. https://aws.amazon.com/api-gateway/.
3. Amazon s3. https://aws.amazon.com/s3/.
4. Annoflex. https://tomassetti.me/parsing-in-java/.
5. Antlr. https://tomassetti.me/parsing-in-java/.
6. Apache jmeter. https://en.wikipedia.org/wiki/Apache_JMeter.
7. Aws ebs. https://aws.amazon.com/ebs/.
8. Aws ebs pricing. https://n2ws.com/blog/ebs-snapshot/aws-ebs-pricing-hdd-and-ssd.
9. Aws ec2. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/concepts.html.
10. Aws lambda. https://aws.amazon.com/lambda/.
11. Aws s3 costs to save. https://blog.cloudability.com/aws-s3-understanding-cloud-storage-costs-to-save/.
12. Binary large object(blob). https://en.wikipedia.org/wiki/Binary_large_object.
13. Code prettify. https://github.com/google/code-prettify.
14. Codemirror. http://codemirror.net/.
15. Content delivery network. https://en.wikipedia.org/wiki/Content_delivery_network.
16. Continuous integration. https://en.wikipedia.org/wiki/Continuous_integration.
17. Ebs, efs, s3: Best cloud storage system. https://cloud.netapp.com/blog/ebs-efs-amazons3-best-cloud-storage-system.
18. Execution process of java program in detail. http://bit.ly/2TWxqkF.
19. Git hub. https://github.com.
20. Github repository of online java navigator. https://git.io/fjHUy.
21. Integrated development environment(ide). https://en.wikipedia.org/wiki/Integrated_development_environment.
22. Integrated evelopment environment. https://en.wikipedia.org/wiki/Integrated_development_environment.
23. Introduction of gitpod. https://www.gitpod.io/about/.
24. Introduction of sourcegraph. https://about.sourcegraph.com.
25. Javaparser. https://javaparser.org/.
26. Jira software. https://www.atlassian.com/software/jira.
27. Parsing in java. https://tomassetti.me/parsing-in-java/.
28. Releases of online java navigator. https://github.com/DnsZhou/online-Java-navigator/releases.
29. Scrum (software development). https://en.wikipedia.org/wiki/Scrum_(software_development).
30. Software performance testing. https://en.wikipedia.org/wiki/Software_performance_testing.
31. Syntax highlighter. https://github.com/syntaxhighlighter/syntaxhighlighter.
32. Waterfall model. https://en.wikipedia.org/wiki/Waterfall_model.
33. zgrepcode. https://zgrepcode.com/.
34. Anirban Basu. *Software quality assurance, testing and metrics*. PHI Learning Pvt. Ltd., 2015.

35. Patrick Niemeyer Daniel Leuck. *Learning Java, 4th Edition*. O'Reilly Media, Inc., 2013.
36. Emily H Halili. *Apache JMeter: A practical beginner's guide to automated testing and performance measurement for your websites*. Packt Publishing Ltd, 2008.
37. Dorota Huizinga and Adam Kolawa. *Automated defect prevention: best practices in software management*. John Wiley & Sons, 2007.
38. Nicholas Smith, Danny van Bruggen, and Federico Tomassetti. *JavaParser: Visited*. Leanpub, November 2016.

Appendix 1: Relavant source code

```java
//uk.ac.ncl.cs.tongzhou.navigator.LinkinVisitor.java

  @Override
    public void visit(ClassOrInterfaceType node, JavaSymbolSolver javaSymbolSolver) {

        SimpleName simpleName = node.getName();
        String fullName = node.getTokenRange().get().toString();
        if (fullName.contains("<")) {
            fullName = fullName.replaceAll("<.*>", "");
        }
        String navFrom = findRelativePath(node);
        LinkObject linkObject = new LinkObject(navFrom, fullName);
        simpleName.setData(LINK_TO, linkObject);
        simpleName.setData(LINK_STYLE, "ClassOrInterfaceType");

    }
```

Listing 3: Retrieve all occurrence of class and interface type during visiting

```java
//uk.ac.ncl.cs.tongzhou.navigator.LinkinVisitor.java

public class LinkingVisitor extends VoidVisitorAdapter<JavaSymbolSolver> {

    public static final DataKey<String> LINK_STYLE = new DataKey<>() {
    };
    public static final DataKey<String> LINK_ID = new DataKey<>() {
    }; // for origins
    public static final DataKey<LinkObject> LINK_TO = new DataKey<>() {
    }; // for targets

    private final List<TypeDeclaration> typeDeclarations = new ArrayList<>();
    private final List<ImportDeclaration> importDeclarations = new ArrayList<>();

    private final List<String> declaredTypes = new ArrayList<>();
    private final List<String> declaredImports = new ArrayList<>();

    public List<ImportDeclaration> getImportDeclarations() {
        return importDeclarations;
    }
    public List<String> getDeclaredTypes() {
        return declaredTypes;
    }


    @Override
    public void visit(ClassOrInterfaceDeclaration node,
   JavaSymbolSolver javaSymbolSolver) {

        typeDeclarations.add(node);

        ResolvedReferenceTypeDeclaration
   resolvedReferenceTypeDeclaration = javaSymbolSolver
                .resolveDeclaration(node,
   ResolvedReferenceTypeDeclaration.class);
        String name = resolvedReferenceTypeDeclaration.
   getQualifiedName();

        node.setData(LINK_ID, name);
        node.setData(LINK_STYLE, "ClassOrInterfaceDeclaration
   ");

        this.declaredTypes.add(name);

        super.visit(node, javaSymbolSolver);
  }

  /*...*/
}
```

Listing 4: Retrieve type declaration during visiting

```java
//uk.ac.ncl.cs.tongzhou.navigator.RepositoryWalker.java

private void indexRepository() throws IOException {
  //scan all the package.json file in the output/JsonDocs
    folder and generate index into index/ folder
  /*
  Index format:
  <group>:<artifact>:<version>:<declaration name>
  eg  antlr:antlr:2.7.7.redhat-7:antlr.actions.cpp.
    ActionLexer
   */
  Files.createDirectories(outputIndexRootDir.toPath());
  Files.walkFileTree(outputJsonRootDir.toPath(), new SimpleFileVisitor<Path>() {

    @Override
    public FileVisitResult visitFile(Path file, BasicFileAttributes attributes) throws IOException {

      //Todo: think about whether change it to a special name
      if (file.toFile().isFile() && file.toFile().getName().equals("package.json")) {
        String relativePath = file.toString()
            .replace(outputJsonRootDir.getPath(), "")
            .replace(SLASH + "package.json", "");

        //To solve the separator problem with regular expression
        String[] pathTokens = relativePath.substring(1).split(SLASH.equals("\\") ? "\\\\" : SLASH);

        String artifact = pathTokens[pathTokens.length - 3];
        String version = pathTokens[pathTokens.length - 2];
        String group = relativePath.substring(1)
            .replace(SLASH + artifact + SLASH + version + SLASH + artifact + "-" + version, "");
        group = group.replace(SLASH, ".");
        PackageInfo currentPackageInfo = objectMapper.readValue(file.toFile(), PackageInfo.class);
        for (CompilationUnitDecl cuItem : currentPackageInfo.compilationUnitDecls) {
          for (TypeDecl typeDecl : cuItem.typeDecls) {
            String typeName = typeDecl.name.substring(typeDecl.name.lastIndexOf(".") + 1, typeDecl.name.length());
            String gavCuString = group + ":" + artifact + ":" + version + ":" + typeDecl.name;
            IndexType.addIndexItem(typeName, gavCuString);
          }
        }
      }
      return FileVisitResult.CONTINUE;
    }
  });
  IndexType.generateIndex();
}
```

Listing 5: Generate index file

```java
//uk.ac.ncl.cs.tongzhou.navigator.Resolver.java

```

```java
public String resolve(String groupId, String artifactId, String version,
                          String compilationUnit, String navigateFrom, String navigateTo, List<String> classpathGAVs) throws IOException {
        this.currentClasspathGavs = classpathGAVs;
        GavCu navigateFromGavCu = new GavCu(groupId, artifactId, version, compilationUnit);

        /*Get information of current Type from GavCu*/
        CompilationUnitDecl cuDecl = getCompilationUnitDecl(navigateFromGavCu);
        String navFromPkg = getPackage(cuDecl);
        List<ImportDecl> navFromImports = getImports(cuDecl);
        List<TypeDecl> navFromTypeDecls = getTypeDecls(cuDecl);
        String fullNavFromString = compilationUnit.substring(0, compilationUnit.lastIndexOf(".") + 1) + navigateFrom;

        /*====Rule 1: Current Type itself*/
        if (navigateTo.equals(subStringLastDot(navigateFromGavCu.cuName.replace(navFromPkg + ".", "")))) {
            return validateAndMakePath(Arrays.asList(navigateFromGavCu).stream(), null);
        }

        /*====Rule2: target type is an nested type of current type*/
        if (navFromTypeDecls != null && !navFromTypeDecls.isEmpty()) {
            for (TypeDecl typeDecl : navFromTypeDecls) {
                /*case 1: quote internal class without the name of its father class, eg: InternalClass class; */
                if (typeDecl.name.replace(fullNavFromString + ".", "").equals(navigateTo)) {
                    String targetId = fullNavFromString.substring(fullNavFromString.lastIndexOf(".") + 1, fullNavFromString.length());
                    return validateAndMakePath(Arrays.asList(navigateFromGavCu).stream(), targetId + "." + navigateTo);
                }
                /*case 2: quote internal class with the name of its father class, eg: ThisClass.InternalClass class; */
                if (typeDecl.name.equals(fullNavFromString.substring(0, fullNavFromString.lastIndexOf(".")) + "." + navigateTo)) {
                    return validateAndMakePath(Arrays.asList(navigateFromGavCu).stream(), navigateTo);
                }
            }
        }

        /* Find the index file with the type name;*/
        String navigateToTypeName = navigateTo.substring(navigateTo.lastIndexOf(".") + 1, navigateTo.length());
        List<String> gavsContainingMatchingCUs = findGAVsContaining(navigateToTypeName);
        if (gavsContainingMatchingCUs != null && !gavsContainingMatchingCUs.isEmpty()) {
            Set<String> candidateSet = new HashSet<>(gavsContainingMatchingCUs);

            /*====Rule 3: The specific single type imported Type*/
            if (navFromImports != null && !navFromImports.isEmpty()) {
                List<String> importStringList = navFromImports.stream().map(importDecl -> importDecl.name).collect(Collectors.toList());
                Set<String> importFilteredCandidates = filterCandidatesByImports(candidateSet, importStringList);
                if (importFilteredCandidates.isEmpty()) {
                    /*====Rule 3.1 nested type for specific imported case*/
                    if (navigateTo.contains(".")) {
                        String parentTypeString = navigateTo.substring(0, navigateTo.indexOf("."));
                        List<String> filteredImportForCandidate = importStringList.stream()
                                .filter(imp -> imp.substring(imp.lastIndexOf(".") + 1, imp.length()).equals(parentTypeString))
                                .collect(Collectors.toList());
                        List<String> candidateImportStrings = filteredImportForCandidate.stream()
                                .map(str -> str.substring(0, str.lastIndexOf(".")) + "." + navigateTo)
                                .collect(Collectors.toList());
                        importFilteredCandidates = filterCandidatesByImports(candidateSet, candidateImportStrings);
                    }
                }
                if (importFilteredCandidates != null &&
    importFilteredCandidates.size() != 0) {
                    return validateAndMakePath(
    importFilteredCandidates.stream().map(candidate -> new
    GavCu(candidate)), navigateTo);
                }
            }

            /*====Rule 4: same package Types, get current
    package and use it to filter the index candidates*/
            Set<String> result = new HashSet<>(candidateSet);
//            navigateTo
            result.removeIf(candidate -> substringPkgName(
    candidate, navigateTo) == null
                    || !substringPkgName(candidate,
    navigateTo).equals(navFromPkg));
            if (!result.isEmpty()) {
                return validateAndMakePath(result.stream().
    map(candidate -> new GavCu(candidate)), navigateTo);
            }

            /*Todo: fix X.X type for specific single type
    import*/
            /*====Rule 5: on demand import, aka wildcard *
    import*/
            if (navFromImports != null && !navFromImports.
    isEmpty()) {
                List<String> tryImportDecls = navFromImports.
    stream().filter(importDecl -> importDecl.name.contains("*
    ")).map(importDecl
                        -> importDecl.name.replace("*",
    navigateTo)).collect(Collectors.toList());
                Set<String> ondemandImportFilteredCandidates
    = filterCandidatesByImports(candidateSet, tryImportDecls)
    ;
                if (ondemandImportFilteredCandidates != null
    && ondemandImportFilteredCandidates.size() > 0) {
                    return validateAndMakePath(
    ondemandImportFilteredCandidates.stream().map(candidate
    -> new GavCu(candidate)), null);
                }
            }

            /*====Rule 6: Default imported Type: java.lang*/
            List<String> langResult = candidateSet.stream().
    filter(candidate -> substringPkgName(candidate,
    navigateTo) != null
                    && substringPkgName(candidate, navigateTo
    ).equals("java.lang")).collect(Collectors.toList());
```

```
      if (langResult != null && langResult.size() > 0) {
        return validateAndMakePath(langResult.stream().map(candidate -> new GavCu(candidate)), null);
      }
    }
    /*Todo: fix X.X type for specific single type import*/
    /*Rule 7: Directly referred Type eg: com.google.testClass test = new com.google.testClass()*/
    String directRefTypeName = navigateTo.substring(navigateTo.lastIndexOf(".") + 1, navigateTo.length());
    List<String> gavsContainingMatchingCUsForDR = findGAVsContaining(directRefTypeName);
    if (gavsContainingMatchingCUsForDR != null && !gavsContainingMatchingCUsForDR.isEmpty()) {
      Set<String> candidateSet = new HashSet<>(gavsContainingMatchingCUsForDR);
      candidateSet.removeIf(candidate -> !substringTypeName(candidate).equals(navigateTo));
      if (candidateSet.size() > 0) {
        return validateAndMakePath(candidateSet.stream().map(candidate -> new GavCu(candidate)), null);
      }
    }

    return null;
  }
```

Listing 6: Main navigation algorithm

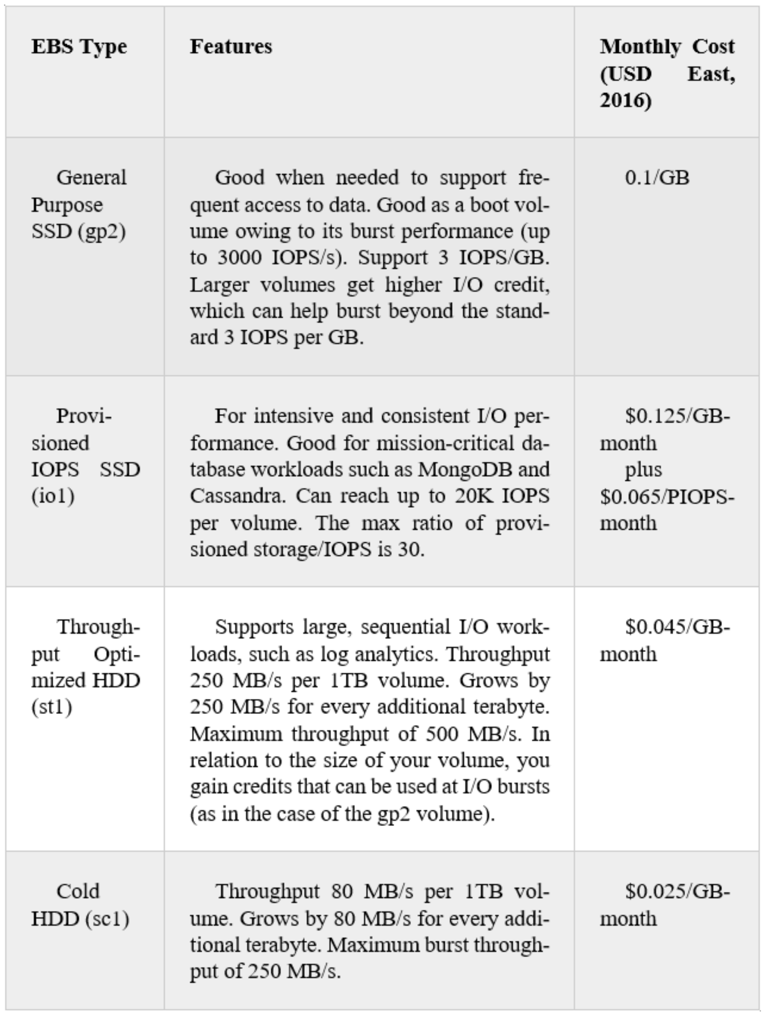
```
//uk.ac.ncl.cs.tongzhou.navigator.Resolver.java

private String validateAndMakePath(Stream<GavCu> candidateStream, String navTo) {
 /*iterate the classpath in order, when find the candidate in any classpath, make the path with this candidate*/
 /*which means, if there is multiple candidate, the one who is listed upper in classpath will be used.*/
 if (this.currentClasspathGavs != null && this.currentClasspathGavs.size() != 0
   && !isEmptyStringList(this.currentClasspathGavs)) {
  List<GavCu> candidates = candidateStream.collect(Collectors.toList());
  for (String classpath : this.currentClasspathGavs) {
   GavCu result = candidates.stream().filter(candidate -> validateGavCuInSingleClasspath(candidate, classpath)).findFirst().orElse(null);
   if (result != null) {
        return makePath(result, navTo);
      }
    }
    return CLASSPATH_NOT_INCLUDED_RESULT;
  }
  return makePath(candidateStream.findAny().orElse(null),
    navTo);
}
```

Listing 7: Classpath validation

```
//uk.ac.ncl.cs.tongzhou.navigator.RepositoryWalker.java

if (GENERATE_TEST_CASES && linkObjectListInCurrentCu != null
    && !linkObjectListInCurrentCu.isEmpty()) {
  List<String> gavCuToList = linkObjectListInCurrentCu.stream
    ().map(linkObject -> gavCuString.replace(
      ":", ",") + "," + linkObject.navFrom + "," + linkObject
    .navTo)
      .distinct().collect(Collectors.toList());
  File testCaseFile = new File(outputTestCaseFileRootDir,
    gavCuString
      .replace(":", "_") + ".csv");
  testCaseFile.getParentFile().mkdirs();
  Files.write(testCaseFile.toPath(), gavCuToList,
    StandardCharsets.UTF_8);
}
```

Listing 8: Generating test cases for automation test

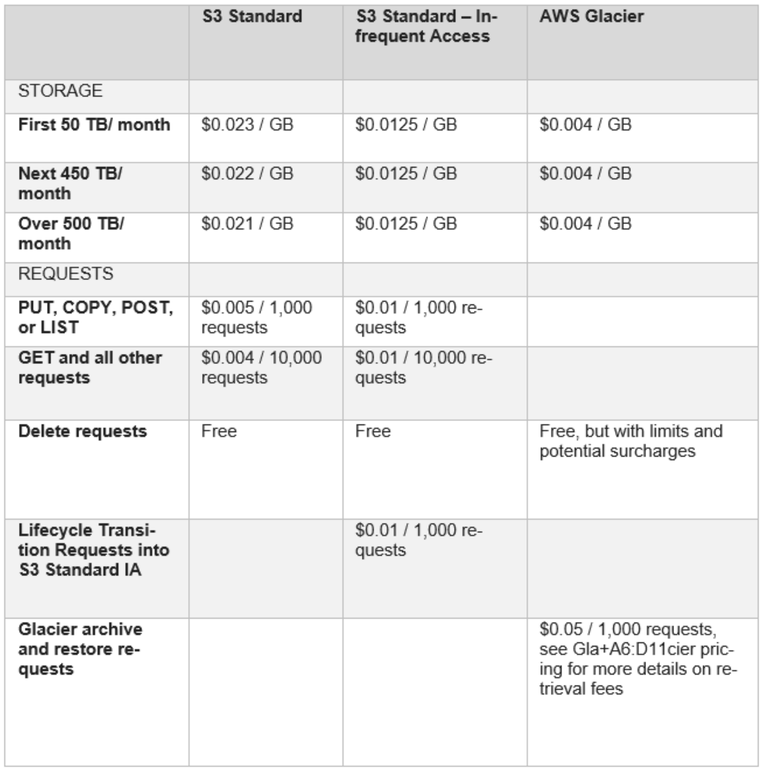
Appendix 2: EBS Pricing



EBS Type	Features	Monthly Cost (USD East, 2016)
General Purpose SSD (gp2)	Good when needed to support frequent access to data. Good as a boot volume owing to its burst performance (up to 3000 IOPS/s). Support 3 IOPS/GB. Larger volumes get higher I/O credit, which can help burst beyond the standard 3 IOPS per GB.	0.1/GB
Provisioned IOPS SSD (io1)	For intensive and consistent I/O performance. Good for mission-critical database workloads such as MongoDB and Cassandra. Can reach up to 20K IOPS per volume. The max ratio of provisioned storage/IOPS is 30.	$0.125/GB-month plus $0.065/PIOPS-month
Throughput Optimized HDD (st1)	Supports large, sequential I/O workloads, such as log analytics. Throughput 250 MB/s per 1TB volume. Grows by 250 MB/s for every additional terabyte. Maximum throughput of 500 MB/s. In relation to the size of your volume, you gain credits that can be used at I/O bursts (as in the case of the gp2 volume).	$0.045/GB-month
Cold HDD (sc1)	Throughput 80 MB/s per 1TB volume. Grows by 80 MB/s for every additional terabyte. Maximum burst throughput of 250 MB/s.	$0.025/GB-month

Fig. 31: EBS Pricing[8]

Appendix 3: S3 Pricing



	S3 Standard	S3 Standard – In-frequent Access	AWS Glacier
STORAGE			
First 50 TB/ month	$0.023 / GB	$0.0125 / GB	$0.004 / GB
Next 450 TB/ month	$0.022 / GB	$0.0125 / GB	$0.004 / GB
Over 500 TB/ month	$0.021 / GB	$0.0125 / GB	$0.004 / GB
REQUESTS			
PUT, COPY, POST, or LIST	$0.005 / 1,000 requests	$0.01 / 1,000 re-quests	
GET and all other requests	$0.004 / 10,000 requests	$0.01 / 10,000 re-quests	
Delete requests	Free	Free	Free, but with limits and potential surcharges
Lifecycle Transi-tion Requests into S3 Standard IA		$0.01 / 1,000 re-quests	
Glacier archive and restore re-quests			$0.05 / 1,000 requests, see Gla+A6:D11cier pric-ing for more details on re-trieval fees

Fig. 32: S3 Pricing[11]