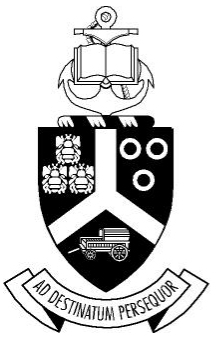

UNIVERSITEIT VAN PRETORIA
UNIVERSITY OF PRETORIA
YUNIBESITHI YA PRETORIA

Department of Computer Science

COS 301 - Software Engineering

Amazon
Network Visualizer
Testing Document

Not-Like-This

Authors:	*Student number:*
Jedd Shneier	u13133064
Daniel King	u13307607
Muller Potgieter	u12003672

September 28, 2016

Contents

1 Introduction

Our project is all about visualizing Amazon Web Services (AWS) virtual networks. The user can scan their entire network and in real time get a depiction of their cloud networks. It is thus a network architecture tool as well as a teaching tool for the average user to help them understand AWS network architecture and infrastructure. This project mainly focuses with performance and scalability. Scalability as very large networks connected in very complex ways need to be scanned, and performance as this needs to be done quickly and rendered in real time. These requirments reflect in our testing cases. This document details our Functional Features to be Tested, PAss/Fail Criteria, Test Deliverables,Detailed test results and finnally Conlusions and Reccomendations.

1.1 Purpose

The purpose of this document is to contain the complete suite of artifacts that describe test planning, test design, test execution, test results and conclusions drawn from the testing activity. This document holds our guidlines for testing , or results from these tests and a blueprint for future tests.

1.2 Scope

The scope of this document is structured as follows. The features that are considered for testing are listed and those that have been identified from the requirements are discussed in detail in the Unit Test Plan. Furthermore, this document outlines the test environment and the risks involved in the testing approaches that will be followed. Assumptions and dependencies of this test plan will also be mentioned.The Unit Test Report discusses and concludes on the results of the testsFinally reccomendations are made for further development,issues encountered and future tests to be carried out.

1.3 Test Environment

- Hardware: All tests were performed on midrange laptops.4GB RAM, Intel(R)Core(TM) i5 CPU 750 @2.67 GHz
- Software: Amazon network scanner (Back-end), Amazon network vizualizer(Front-End), NetworkRESTAPI(REST Server)
- Network: 4MB uncapped MWEB ADSL
- Programming Languages:Java EE (SDK-7u2),Jeresey-RX,JavaScript, Viz.js
- Testing Frameworks: JUnit
- Coding Environment: InteliJ IDEA Developers edition 2016.2.1
- Operating System: Windows 7 64 bit Proffesional edition
- Internet Browsers: Firefox, Google Chrome, IE. All latest editions.

- Server:GlassFish Server Open Source Edition 4.1.1
- API: Amazon EC2 API

1.4 Assumptions and Dependencies

Assumptions:

- We assumed a fast, stable uncapped internet conenction.
- We assumed a valid user account for AWS.
- We assumed a midrange network on the account.
- We assumed Amazon web services would be running in all regions.

Dependencies:

- Java EE installed.
- GlassFish installed.
- Amazon SDK as a Maven dependency.
- Jersey 2.16 as a Maven dependency.
- Junit a Maven dependency.
- Viz.js library.
- AngularJs.
- Bootstrap.

2 Unit Test Plan

2.1 Test Items

The application under test is the Amazon Web Services Network Visualiser. It is divded into three seperate and interconnected applications namely the Front-end(User interface and network visualizer), the Back-end(Network Scanner and Smart buffer) and the applocation server(NetworkRESTServer and NetworkRESTAPI). All three are tested in their own unit tests and integration tests, and together are tested in the use case tests.

2.2 Functional Features to be Tested

For this level of testing each of the three applications will be tested. Their individual subprograms will be tested according to their functionality and each applciation will be integration tested and finnaly perform their own unit test. The functional features to be tested in each application are as follows:

Front-end

- Send data to server tested through AJAX scripts in UI HTML.
- Recieve data from server tested through AJAX scripts in UI HTML.
- Visualize network jsons tested through the visualiser.js script.
- Iterative adding to network tested through the visualiser.js script.
- Dynamic menu and information box tested through the UI HTML.
- Button expected functionality tested through the UI HTML.

For this unit tests will be performed with mock results from the server. The functionality of the interface will be tested through user testing.

Back-end

- Connecting to AWS account tested through the Credential validation function.
- Scan regions tested through the AWSScanner.
- Threading on different levels tested through the AWSScanner.
- Scanning VPCS tested through the vpcScannerThread.
- Scanning Subnetworks tested through the subnetworkScannerThread.
- Scanning instnaces tested through the isntanceScannerThread.
- Adding scanned results to buffer tested through Sharedbuffer.
- Constructing tree tested through Sharedbuffer.
- Iteratively returning the nodes as JSON objects tested through Sharedbuffer.

Unit tests are performed for each file and if they perform exepcted functionality they pass. The entire Back-end wil lintergration test and the final artifact will be tested as a whole in its own unit test.

REST server

- Deploy on servert tested through GlassFish.
- Accessible from client tested through FireFox and FireBug.

- Able to launch scanner with provided credentials tested through ServerAPI.
- Able to return on AJAX request tested through ServerAPI,FireFox and FireBug .
- Scanning Subnetworks tested through the subnetworkScannerThread.

The unit tests will be done as intergration tests wit hthe first two applciations. Once the unit tests of the two applciations as a whole have been verified correct they will be used in the test of the REST server.

Feature ID	RDS Source	Summary	Approach
Network Visualization	Main Requirement	If network visualised within 3 seconds of input and correctly rendered then success	Mock JSON unit test
Interface interaction	Usability Requirement	If user deems the interface usable and pleasurable then success	Usability test
Read from server	Integration requirement	If visualizer can construct tree from server JSON then success	Mock JSON unit test. Integration test with server
Request services from server	Integration requirement	If interface can request services on the server then success	Mock response unit test. Integration test with server
Connect to AWS	Integration requirement	If able to successfully dry-run connect to account then success	Unit test with user keys. Integration Test with AWS
Launch Threads for each level	Performance requirement	If able to start thread for each level and each thread able to connect then success	Unit test with each levels thread
Scan network	Main Requirement	If network is accurately scanned and buffer updated within 3 seconds then success	Unit test
Scan sub-parts of the network	Performance Requirement	If parts of network are accurately scanned and buffer updated within 3 seconds then success	Unit test
Add to buffer	Performance requirement	If each thread able to add to buffer their correct scans then success	Unit test with mock results from threads
Construct tree	Performance Requirement	If buffer able to construct accurate tree from threads then success	Unit test with mock results from threads
Connect interface to server	Integration requirement	If interface can access server and receive response then success	Integration Test
Connect back-end to server	Integration requirement	If server methods can call different parts of the scanner then success	Integration Test

3 Test Cases

3.1 Test Case 1: Login

3.1.1 Condition : Valid Credentials allow connection to AWS

3.1.2 Objective:

The purpose of this test is to test if valid users are able to connect to AWS with valid credentials.

3.1.3 Input:

The following inputs will be used to test this functionality:

- Valid Access Key and Secret key.
- Valid Access Key and invalid Secret key.
- Invalid Access Key and valid Secret key.
- Invalid Access Key and invalid Secret key.

3.2 Test Case 2: Scan Network

3.2.1 Condition : Network is scanned and visualized

3.2.2 Objective:

The purpose of this test is to test if a full network scan and visualization is successful for a valid user.

3.2.3 Input:

The following inputs will be used to test this functionality:

- Click start scan.

3.2.4 Outcome:

The following are the expected outcomes for a pass result for the functionality:

- Complete network scanned and visualised on the UI.

3.3 Test Case 3: Scan Region

3.3.1 Condition :Specified region is scanned and visualized

3.3.2 Objective:

The purpose of this test is to test if a region scan and visualization is successful for a valid user.

3.3.3 Input:

The following inputs will be used to test this functionality:

- Select region to be scanned.

3.3.4 Outcome:

The following are the expected outcomes for a pass result for the functionality:

- Subtree scanned and visaulised on UI.

3.4 Test Case 4: Scan from UUID

3.4.1 Condition :Scan starts from specified UUID and perfroms sscan of surrounding space

3.4.2 Objective:

The purpose of this test is to test if a subscan is successfully performed with a valid UUID for valid user.

3.4.3 Input:

The following inputs will be used to test this functionality:

- Valid vpc uuid.
- Valid subentwork uuid.
- Valid instance uuid.
- Invalid uuid.

3.4.4 Outcome:

The following are the expected outcomes for a pass result for the functionality:

- Subtree scanned from vpc and visaulised on UI.
- Subtree scanned from subentwork and visaulised on UI.
- Subtree scanned from isntance and visaulised on UI.
- Error message dispalyed.

3.5 Test Case 5: Scan up

3.5.1 Condition :Scan resumes from uuid scan and scans the above region

3.5.2 Objective:

The purpose of this test is to test if a subscan scans up from previous scan with a valid UUID for valid user.

3.5.3 Input:

The following inputs will be used to test this functionality:

- Call scan on region with parents to be scanned.

- Call scan on region with no higher parents to be scanned.

3.5.4 Outcome:

The following are the expected outcomes for a pass result for the functionality:

- Subtree parents visualised.
- Notification that nothing further to be scanned.

3.6 Test Case 6: Scan down

3.6.1 Condition :Scan resumes from uuid scan and scans the below region

3.6.2 Objective:

The purpose of this test is to test if a subscan scans down from previous scan with a valid UUID for valid user.

3.6.3 Input:

The following inputs will be used to test this functionality:

- Call scan on region with sibling children to be scanned.
- Call scan on region with no sibling children to be scanned.

3.6.4 Outcome:

The following are the expected outcomes for a pass result for the functionality:

- Subtree sibling children visualised.
- Notification that nothing further to be scanned.

3.7 Test Case 7: Request information

3.7.1 Condition : Information retrieved for particualr node

3.7.2 Objective:

The purpose of this test is to test if inforamtion can be retrieved for selected nodes.

3.7.3 Input:

The following inputs will be used to test this functionality:

- Click on node.

3.7.4 Outcome:

The following are the expected outcomes for a pass result for the functionality:

- Information dispalyed in inforamtion box.

3.8 Item Pass/Fail Criteria

- Valid keys must result ina connection to AWS.
- Scan must produce a visualised scanned network that matches the visualised mock JSON file.
- Scan of region must produce the region subtree that matches the subtree visualized in full scan.
- Scan from uuid must produce a subtree that matches the subtree visualized in full scan.
- Scan up must produce a subtree that matches the subtree visualized in full scan.
- Scan down must produce a subtree that matches the subtree visualized in full scan.
- the information needs to match the information for the node got through comamnd line interface.

4 Unit Test Report

4.1 Detailed Test Results

4.1.1 Overview of Test Results

The individual unit tests of each part of the applciations were all successful. the applciations themselves passed their unit tests. The intergration tests did not have any failures. The first 4 test cases as well as the last test case succeded. The scan up and scan down test case failed. This is due to a bug in the construction of the network tree in the buffrer.Junit was used in unit tests with the neccesary mock objects and the actual objects were used in the integration tests. The ability to run automated repeatable tests vastly sped up our testing.

4.1.2 Test Case 1

Valid keys were used to conenct to AWS. Invalid keys threw an exception that was caught as intended.

4.1.3 Pass

4.1.4 Test Case 2

Network scanned and visualised matched the JSON file scanned network.

4.1.5 Pass

4.1.6 Test Case 3

Region scanned and visualised matched the subtree of the region found in the full scan.

4.1.7 Pass

4.1.8 Test Case 4

Subnetwork scanned from uuid matched the subtree of the region found in the full scan.

4.1.9 Pass

4.1.10 Test Case 5

Tree did not expand upwards in visualization .

4.1.11 Fail

4.1.12 Test Case 6

Tree expanded downwards but with wrong siblings childrens .

4.1.13 Fail

4.1.14 Test Case 7

Infromation retrieved from node click matched node output from command line.

4.1.15 Pass

4.2 Other

4.2.1 Maven

```
<project xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd"
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.notlikethis</groupId>
  <artifactId>NotLikeThisRESTServer</artifactId>
  <version>1.0-SNAPSHOT</version>
- <dependencies>
  - <dependency>
      <groupId>org.json</groupId>
      <artifactId>json</artifactId>
      <version>20160810</version>
    </dependency>
  - <dependency>
      <groupId>com.amazonaws</groupId>
      <artifactId>aws-java-sdk</artifactId>
      <version>1.11.22</version>
    </dependency>
  - <dependency>
      <groupId>javax</groupId>
      <artifactId>javaee-api</artifactId>
      <version>7.0</version>
    </dependency>
  - <dependency>
      <groupId>org.java-websocket</groupId>
      <artifactId>Java-WebSocket</artifactId>
      <version>1.3.0</version>
    </dependency>
  </dependencies>
</project>
```

4.2.2 Contracts and Mock objects

Our contracts for Scanner, Buffer,NetworkTree and Subscanner allowed great flexibility in development and in testing. Plugging in mock objects that gave expected results allowed us to confrim unit tests of the intergrated concrete parts.

4.2.3 Application Server

Our application is currently deployed to a GlassFish Server however our exploded WAR file can easily be deployed to a Tomcat server(Or any other application server). However the neccesary maven dependencies will be required to be downlaoded and the server configuration set up to deploy the artifact.

4.3 Conclusions and Recommendations

Our system is mostly working correctly however the failed test cases need to be addressed. Once they are addressed all functionality requirements will have been met and we can turn to the performance tests for larger networks and usability tests. The unit tests do not indicate how well system will perfrom with larger networks. The tests are thus only minimum requirments that have been met. Focus from here is on perfromance and scalibility tests.