CSSE477 – Final Project

Software Architecture

By Jonathan Taylor, Austin May and Zach Haloski

The content is stored at .\csse477FinalProject\edu.rosehulman.sws\Reports\Milestone<X>

**Change History**

10/18/2015 Version 1.0: Original Document for Milestone 1

* Original UML Diagram
* Explanation of design patterns used
* Potential Design Improvements
* Test Report for Milestone 1

10/25/2015 Version 2.0: Expanded for Milestone 2

* Creation of Change History
* Updated UML Diagram and UML Description
* Updated Architecture Diagram
* Potential Improvements
* Test Report for Milestone 2
* Created Milestone 2 Feature Listing

11/01/2015 Version 3.0: Expanded for Milestone 3

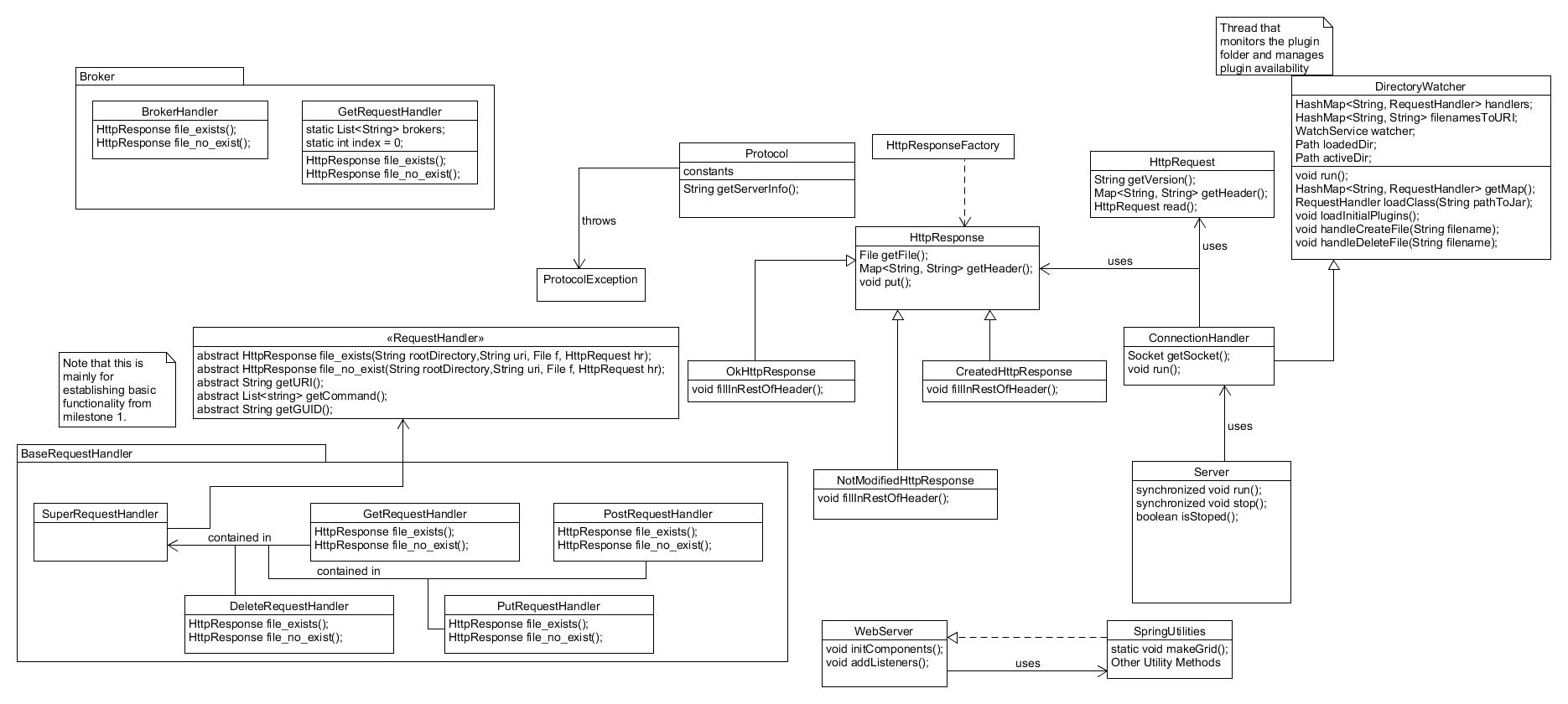
* Updates to Change History
* Minor tweaks to UML Diagram
* Re-formatted to meet specifications
* Tactics and Feature Listing updates
* Architectural Evaluation and Improvements

11/01/2015 Version 4.0: Expanded for Milestone 4

* Updates to Change History
* Minor tweaks to old sections
* Added Sample Application Section
* Added Experimentation with Scaling section
* Updates to Future Improvements

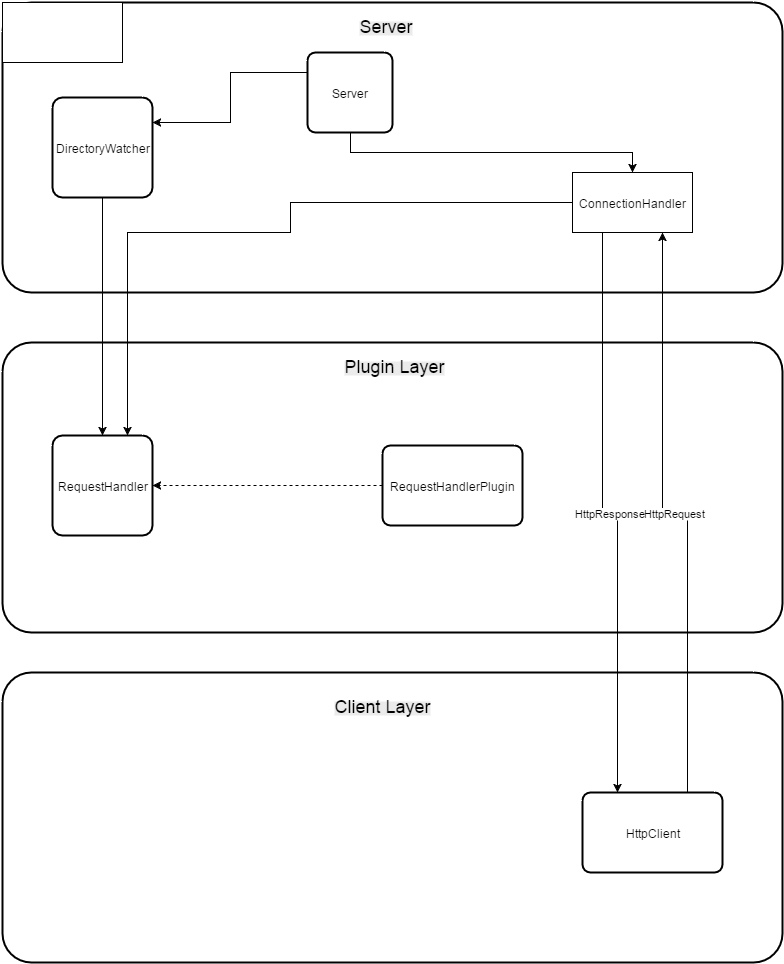
**Design**

1. UML Diagram



For this milestone we used plug-in architecture to attach the RequestHandler(s) into a SuperRequestHandler. We used the DirectoryWatcher class to dynamically track changes to the plug-in folder.

1. Architecture Diagram



**Tactics/Feature Listing**

1. Original Design/MS1 Functionality Implementation: Team
2. DirectoryWatcher Implementation: Austin May
3. Plug-in Architecture Implementation: Jonathan Taylor/Zach Haloski
4. Architectural Evaluations Section Implementation: Team
5. Performance Scenarios Implementation: Team
6. Code Updates based on Scenarios Implementation: Jonathan Taylor/Austin May
7. Server Javascript Implementation: Jonathan Taylor
8. Put/Get Handlers Implementation: Austin May
9. Delete/Post Handlers Implementation: Zach Haloski
10. LoadBalancer Implementation: Team
11. Reformatting of Report Implementation: Zach Haloski

**Architectural Evaluations and Improvements**

Availability Scenario 1

1. Source: File System

Stimulus: Exception

Environment: Normal operation

Artifact: Malformed jar

Response: System handles the fault without failing

Response Measurement: Rate that the system handles the fault without failing

1. Test Plan

Load 10 malformed jars into the server one after the other. Measure the number of jars that cause the server to crash.

1. Baseline

0%. All faults turn into failures

1. Improvement tactics

We will put a try-catch around our server code. We will handle the malformed jar exceptions by deleting the jar from the file system.

1. Results

Our server improved from an availability of 0% after getting a malformed jar exception to 100% after applying our handling tactic.

Availability Scenario 2

1. Source: People/Connections

Stimulus: Large amount of connections

Environment: Overloaded operation

Artifact: Server

Response: System remains available when it gets more requests than it can handle

Response Measurement: Rate that the system handles the fault without failing

1. Test Plan

Do a DDoS attack on the server and measure the rate at which it doesn’t crash.

1. Baseline

0%. All faults turn into failures

1. Improvement tactics

We will throttle the number of incoming connections we receive, and stop receiving connections when we get close to a number that will crash our server. We will test to find this number by doing DDoS attacks with various amounts of requests.

1. Results

Our server improved from an availability of 0% after handling a large amount of clients, to 100% availability up to 100 connections.

Performance Scenario 1

1. Source: People/Connections

Stimulus: Large amount of connections

Environment: Overloaded operation

Artifact: Server

Response: System retains a latency of under 100 ms.

Response Measurement: Measuring of the latency for the responses in the overloaded state

1. Test Plan

Do a DDoS attack on the server and measure the latency of our responses.

1. Baseline

Average Latency of x milliseconds

1. Improvement tactics

We will throttle the number of incoming connections we receive, and queue receiving connections when we get close to a number that will crash our server. We will test to find this number by doing DDoS attacks with various amounts of requests.

1. Results

Our server improved from an average latency of x ms to an average latency of 1.6 ms.

Performance Scenario 2

1. Source: People/Connections

Stimulus: Normal Get Request

Environment: Normal operation

Artifact: Server

Response: System retains a latency of under 100 ms.

Response Measurement: Measuring of the latency for the responses in the overloaded state

1. Test Plan

Do a normal request and measure the latency.

1. Baseline

Average Latency of x milliseconds

1. Improvement tactics

We will add additional threading to our handling code in order to reduce the latency that the requests have.

1. Results

Our server improved from an average latency of x ms to an average latency of 1.6 ms.

Security Scenario 1

1. Source: People/Connections

Stimulus: Large amount of connections

Environment: Overloaded operation

Artifact: Server

Response: System is able to handle DDoS attacks

Response Measurement: System doesn’t fail as a result of a DDoS attack

1. Test Plan

Do a DDos attack on the server and measure the latency of our responses.

1. Baseline

Server crashed due to a DDoS attack

1. Improvement tactics

We will throttle the number of incoming connections we receive, and queue receiving connections when we get close to a number that will crash our server. We will test to find this number by doing DDoS attacks with various amounts of requests.

1. Results

Our server didn’t crash after doing a DDoS attack.

Security Scenario 2

1. Source: File System

Stimulus: Plugin Jar

Environment: Normal operation

Artifact: Server

Response: System will not load jars that have a guid that it doesn’t know about.

Response Measurement: Percentage of jars that the server doesn’t allow that have improper guids.

1. Test Plan

Load some “malicious” jars that our server doesn’t know about and test how many of them the server loads.

1. Baseline

Our server adds every jar that adheres to our plugin interface to the server’s list of plugins.

1. Improvement tactics

We will add a guid to our plugin interface. If a plugin tries to join our server and doesn’t have a guid in our list we will delete the plugin from our list.

1. Results

Our server didn’t add the x number of plugins to our server.

**Sample Application API**

We created a web based way to view, add, remove, and modify the server’s files.

**F1 – Logging in to a repo**

**Method**: GET

**URI**: /manage/

**Request Body**:

<none>

**Response Body**:

{

“code”: 200,

“message”: “Ok”,

}

**Development Status**: DONE

**F2 – View a file**

**Method**: GET

**URI**: /’file id’

**Request Body**:

{

}

**Response Body**:

{

“code”: 200,

“message”: “Ok”,

“body”:

[

“text”: “filetext”

]

}

**Development Status**: DONE

**F3 – Post a file**

**Method**: POST

**URI**: /

**Request Body**:

{

“file”: “filename”

}

**Response Body**:

{

“code”: 200,

“message”: “Ok”,

“filename”:“filename.txt”

}

**Development Status**: DONE

**F4 – Delete a file**

**Method**: ‘file id’

**URI**: /’fileid’

**Request Body**:

<empty>

**Response Body**:

{

“code”: 200,

“message”: “Ok”,

}

**Development Status**: DOING

**F5 – Put a file**

**Method**: PUT

**URI**: /’fileid’

**Request Body**:

{

}

**Response Body**:

{

“code”: 200,

“message”: “Ok”,

[

“text”: “new filetext”

]

}

**Development Status**: Done

**Experimentation with Scaling**

Availability Scenario 1

1. Source: File System

Stimulus: Exception

Environment: Normal operation

Artifact: File System

Response: System handles the fault without failing

Response Measurement: System doesn’t crash when an exception occurs during a file operation

1. Test Plan

Delete a file. This will cause an exception as the connection is using the file at the time it tries to be deleted. Test to see if the server is still available after the fault

1. Baseline

100%. Despite the faults, the server remains available.

1. Scalability

Scaling from one server to two, using a load balancer.

1. Results

Our server remained at 100% availability.

Performance Scenario 1

1. Source: People/Connections

Stimulus: Large amount of connections

Environment: Overloaded operation

Artifact: Server

Response: Server latency remains under 100 ms.

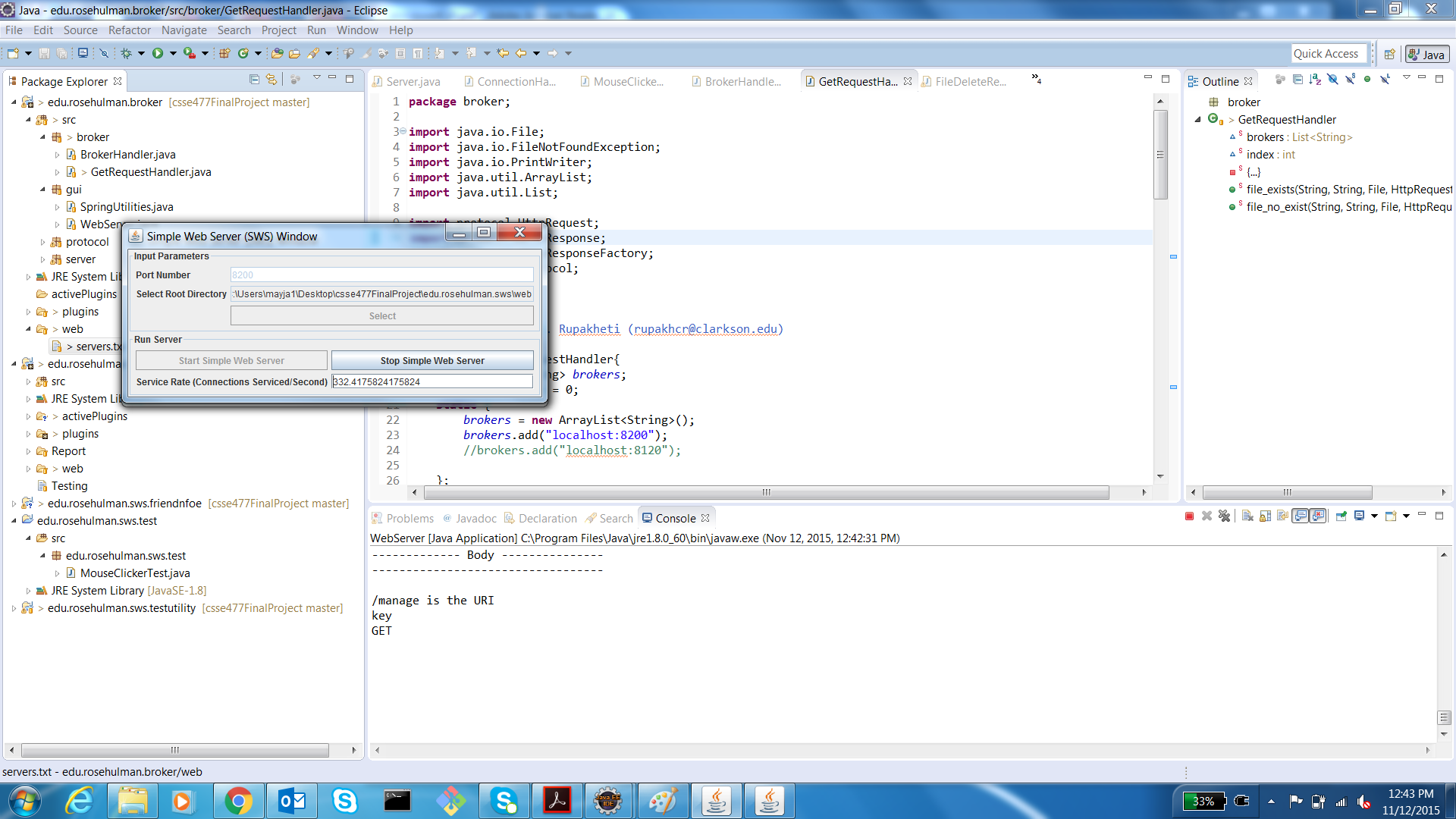
Response Measurement: Measuring of the latency for the responses in the overloaded state

1. Test Plan

Use a mouse clicker to press the get request button 1000 times on our web page and measure the response rate.

1. Baseline

With our initial setup below is the connections per second that we serviced:



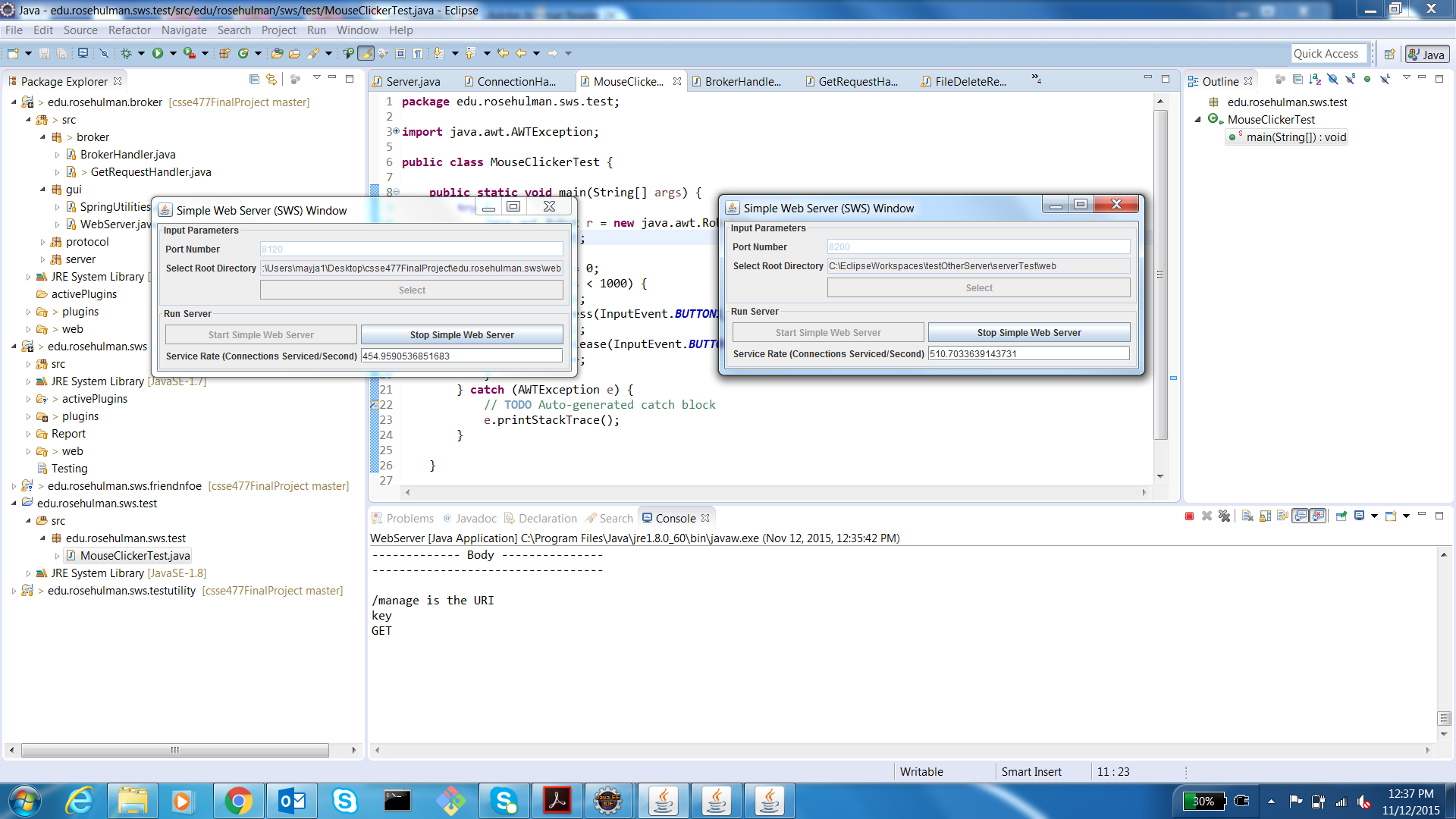
This is 332 connections per second.

1. Scaling

We scaled from one server to two servers using a load balancer

1. Results

Using two servers, below is our measured latency:



This is 454 connections per second for one of our servers and 510 connections per second for our other server, which is an improvement over the baseline.

Security Scenario 1

1. Source: File System

Stimulus: Delete request

Environment: normal operation

Artifact: files.txt file

Response: Though files.txt is deleted, server responds by recreating the file

Response Measurement: System doesn’t fail as a result of the files.txt file being deleted

1. Test Plan

Our delete mechanism allows the users to delete the files from the server. If the user deletes the files.txt file, this removes functionality from our server. Our server should respond by recreating the file.

1. Baseline

Server recreated the files.txt when it was deleted.

1. Scaling

We will scale from one server to two.

1. Results

Both servers were able to recreate the files.txt even when it was deleted.

**Potential Future Improvements**

Handle having two plug-ins with the same URI if they implement different methods.

Maintain connections better for people who are trying to connect.

Ensure that we don’t continue stale connections that do not have activity for long periods of time.