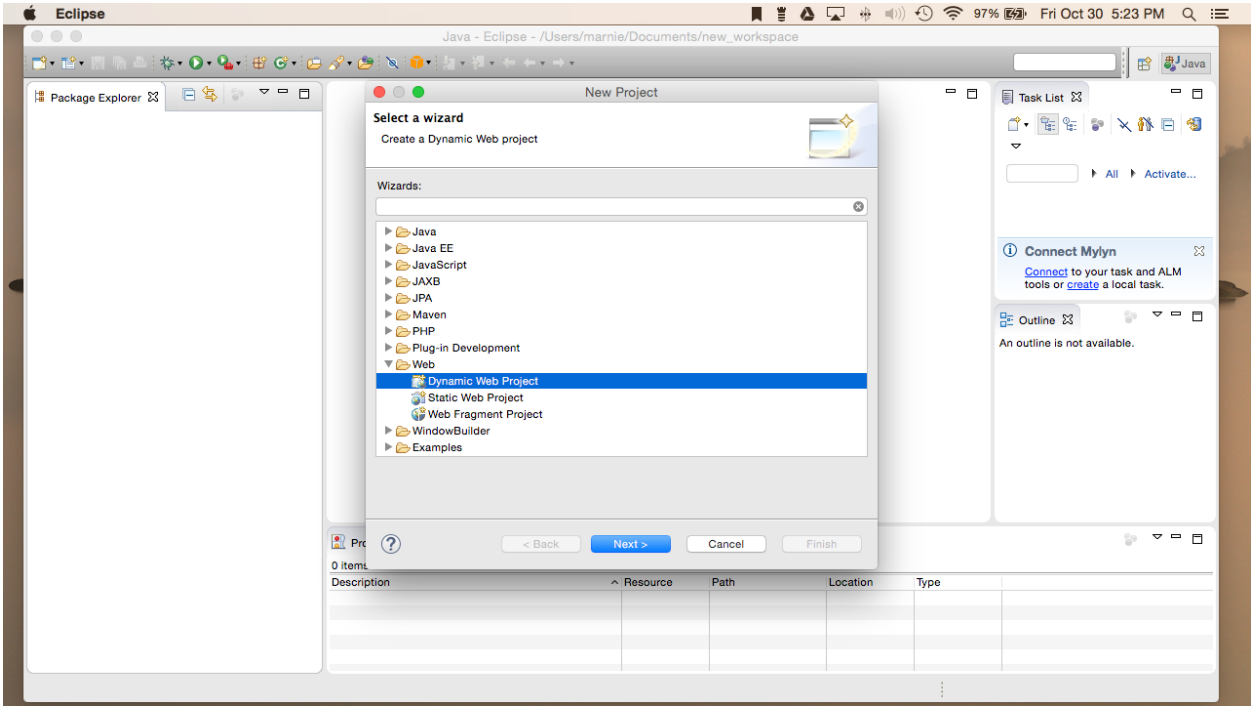
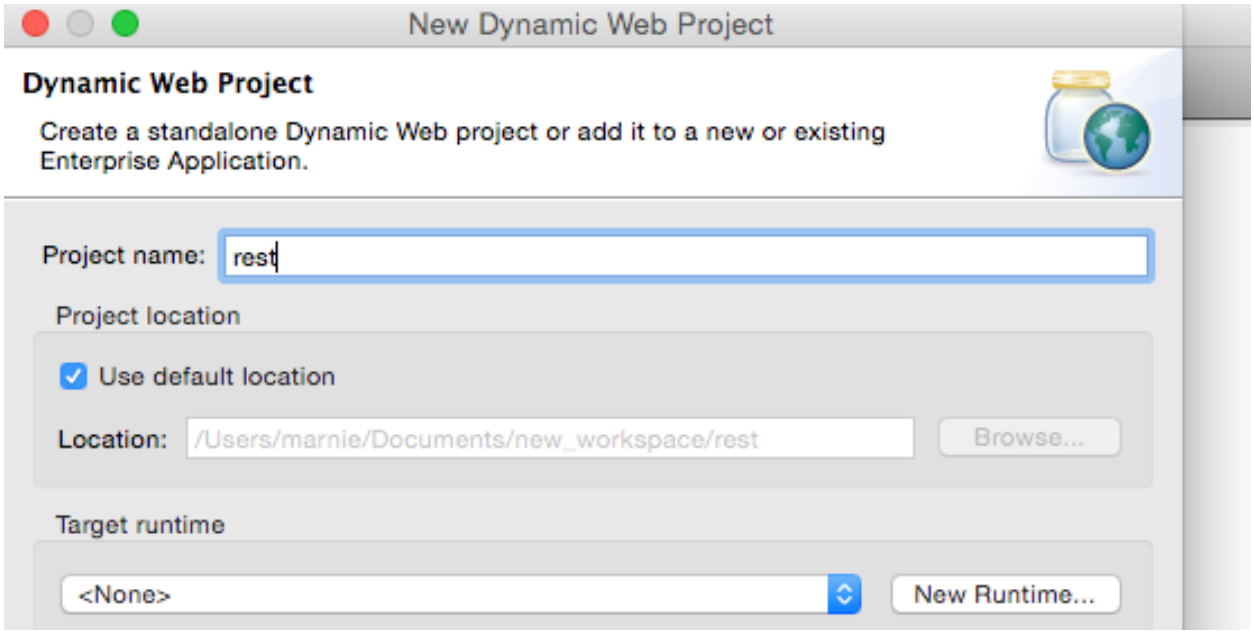
Problem 1:
Open Eclipse in a new workspace. Go to File -> New -> Dynamic Web Project. Call it "rest". This name is arbitrary, however, please do not change it or if you want to change it then modify all URL-s in the client code. Those URLs point to an application called "rest" on the localhost and port 8080. Expand attached ZIP file jax_rs_code.zip. Copy packages `com.rest.client` and `com.rest.ws` from directory `rest` of the expanded ZIP file to the `src` directory under `Java Resources` folder of your project. Similarly, move file `web.xml` to the `WebContet \WEB-INF` directory of your Eclipse project. If your `WEB_INF` directory has a `lib` subdirectory, populate it with the Jersey bundled jar mentioned above (OR if you downloaded the Jersey zip then copy all the libraries). Build your project and fix any errors you might encounter. Run your project on your Tomcat server. You run the project by right clicking on the project and selecting `Run As -> Run on Server`. If your Eclipse is not aware of your Tomcat, please add new server. If your Tomcat is version 8, select Tomcat-8 as the server type. Subsequently, run your client class `CustomerResourceClient` as a Java application. Show your output. Submit working version of all classes, even if you did not modify a thing.
Points: 30

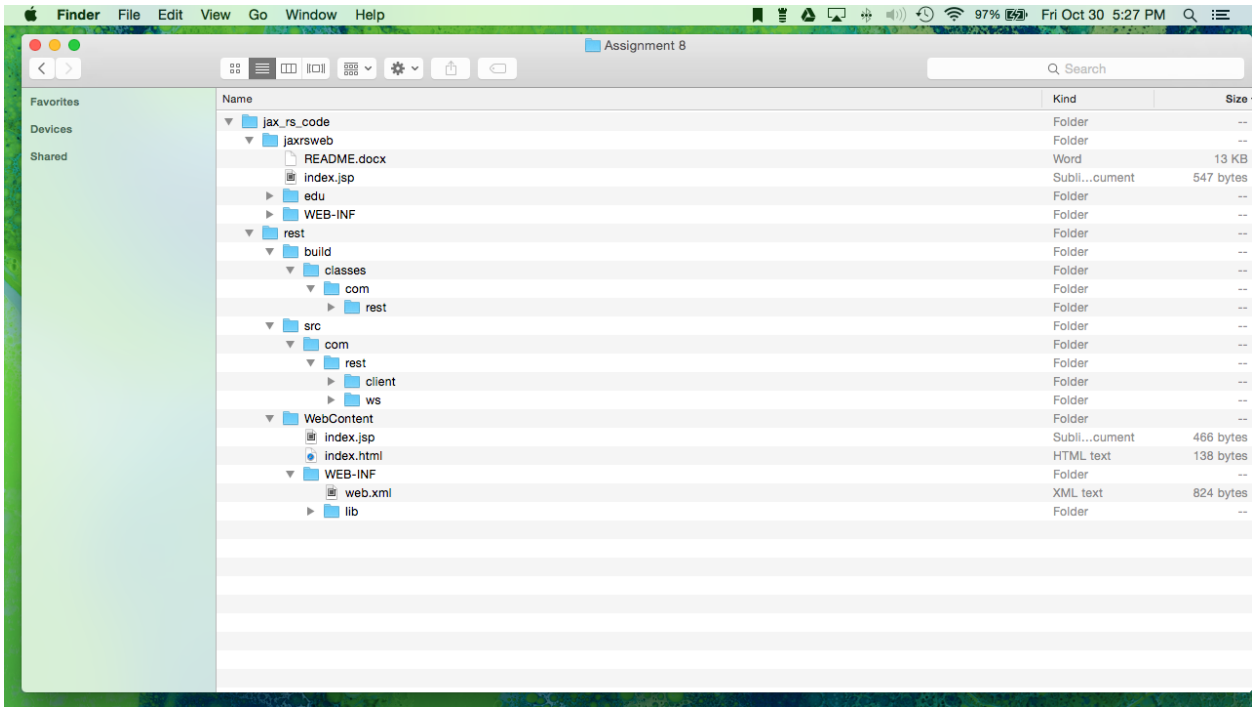
Open Eclipse in a new workspace. Go to File -> New -> Dynamic Web Project.



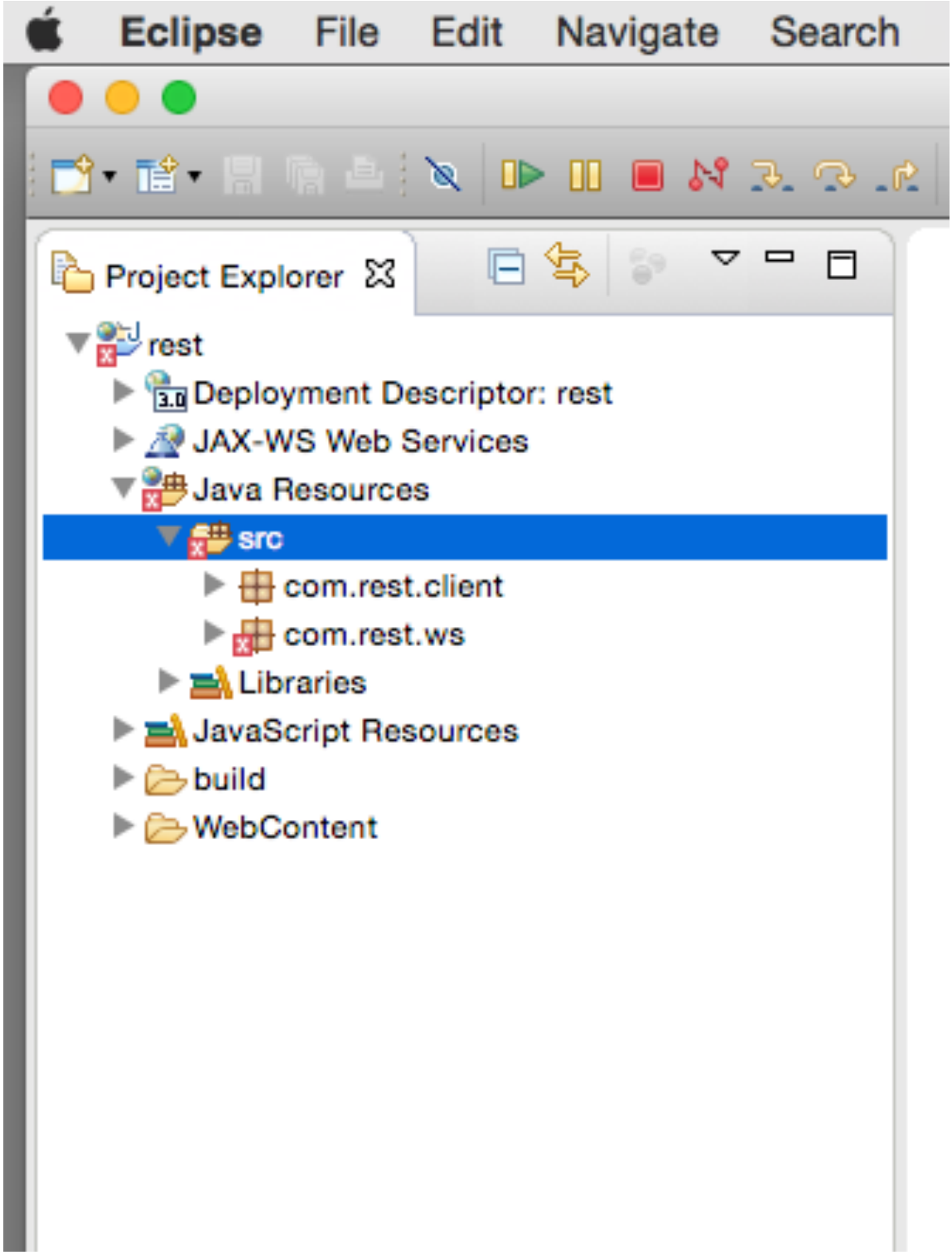
Name it "rest"



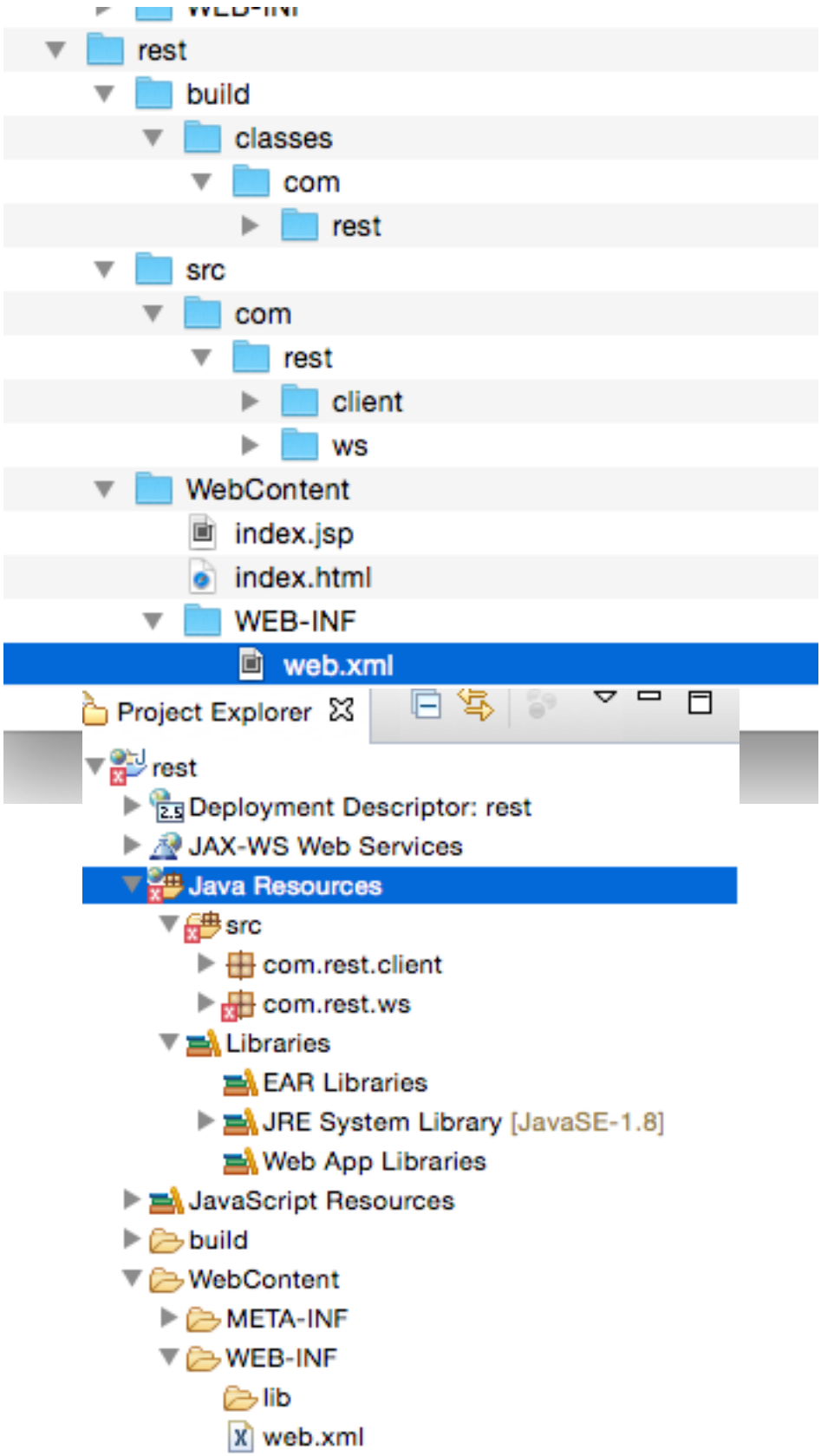
Expand attached ZIP file jax_rs_code.zip.



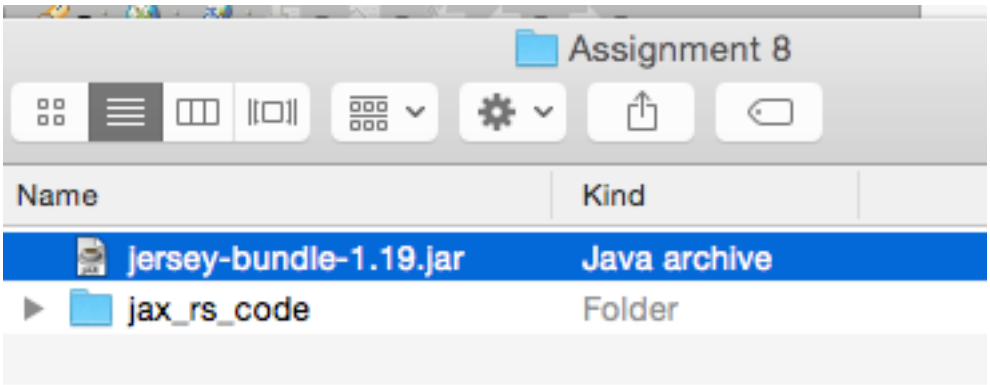
Copy packages `com.rest.client` and `com.rest.ws` from directory `rest` of the expanded ZIP file to the `src` directory under `Java Resources` folder of your project.

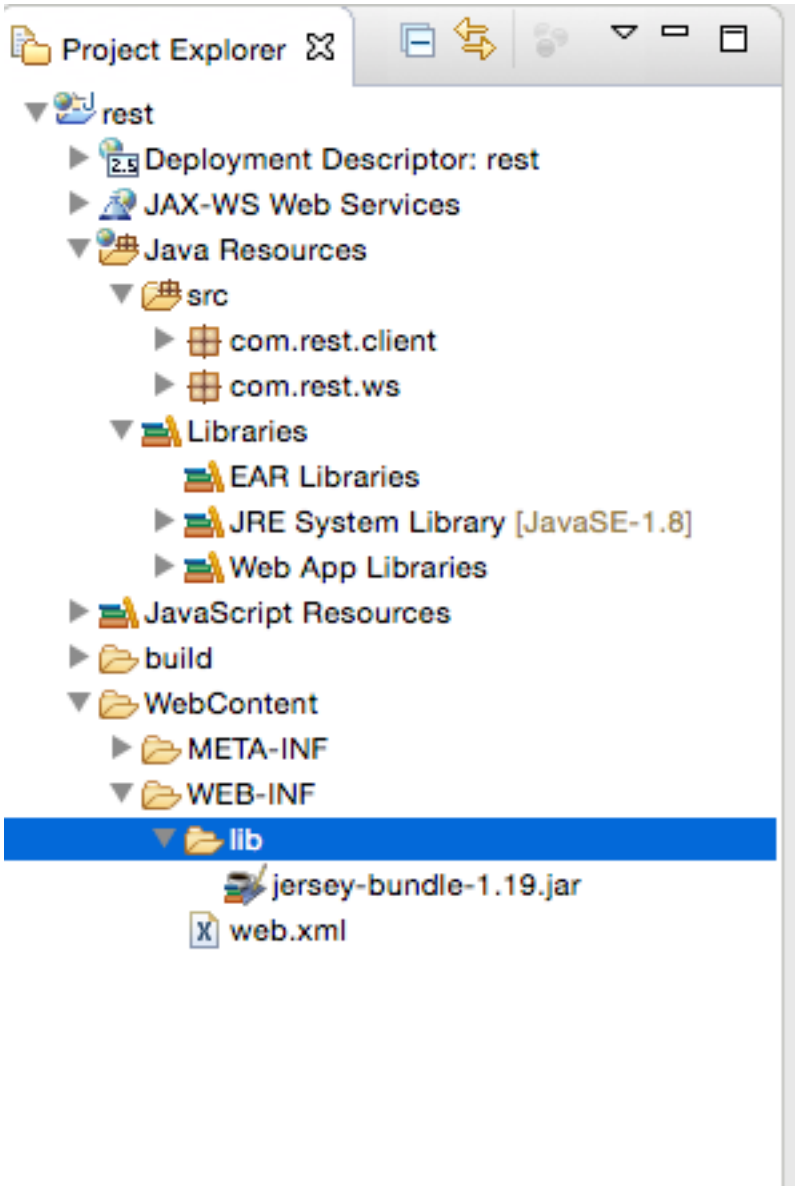


Copy file `web.xml` to the `WebContet\WEB-INF` directory of your Eclipse project.

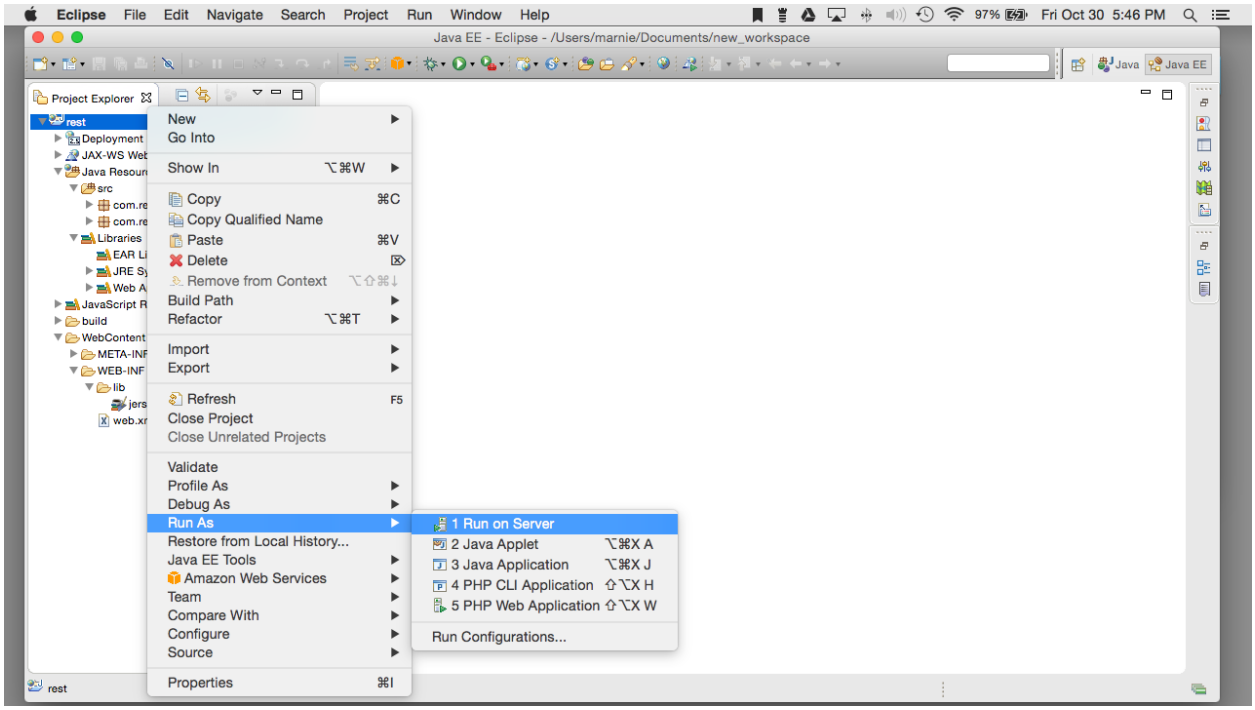


Copy the Jersey bundled jar into the `WEB_INF/lib` directory.

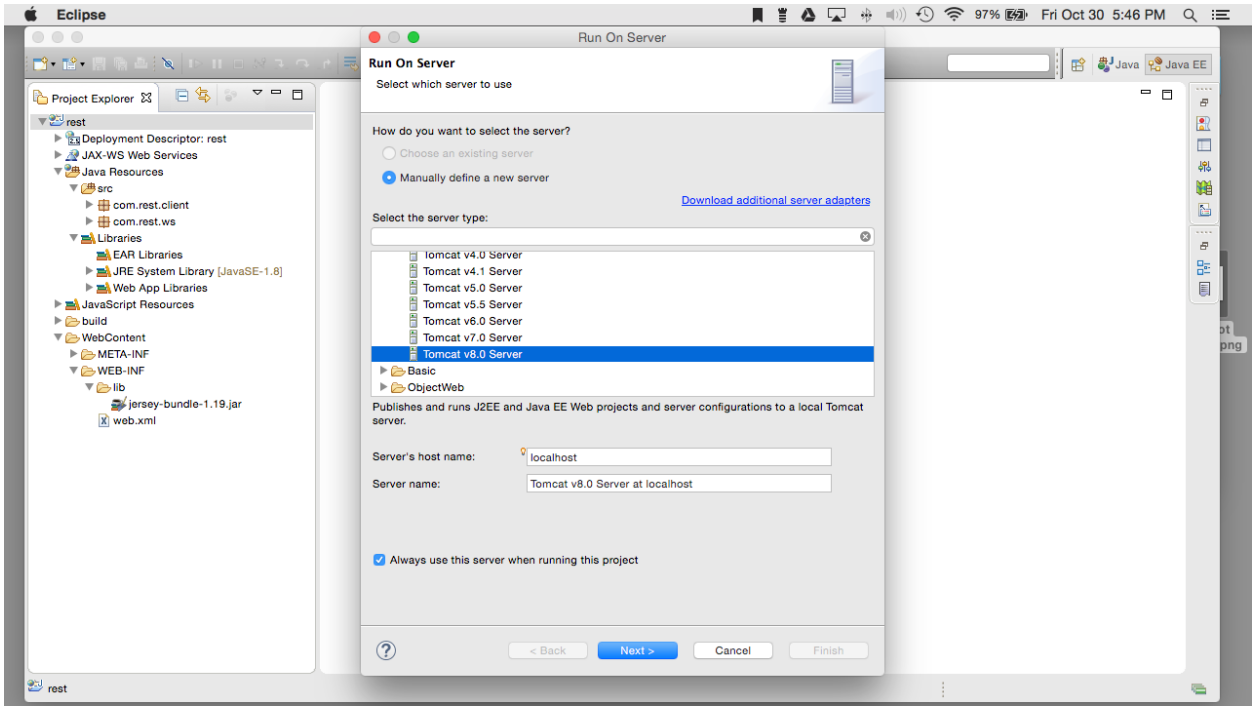




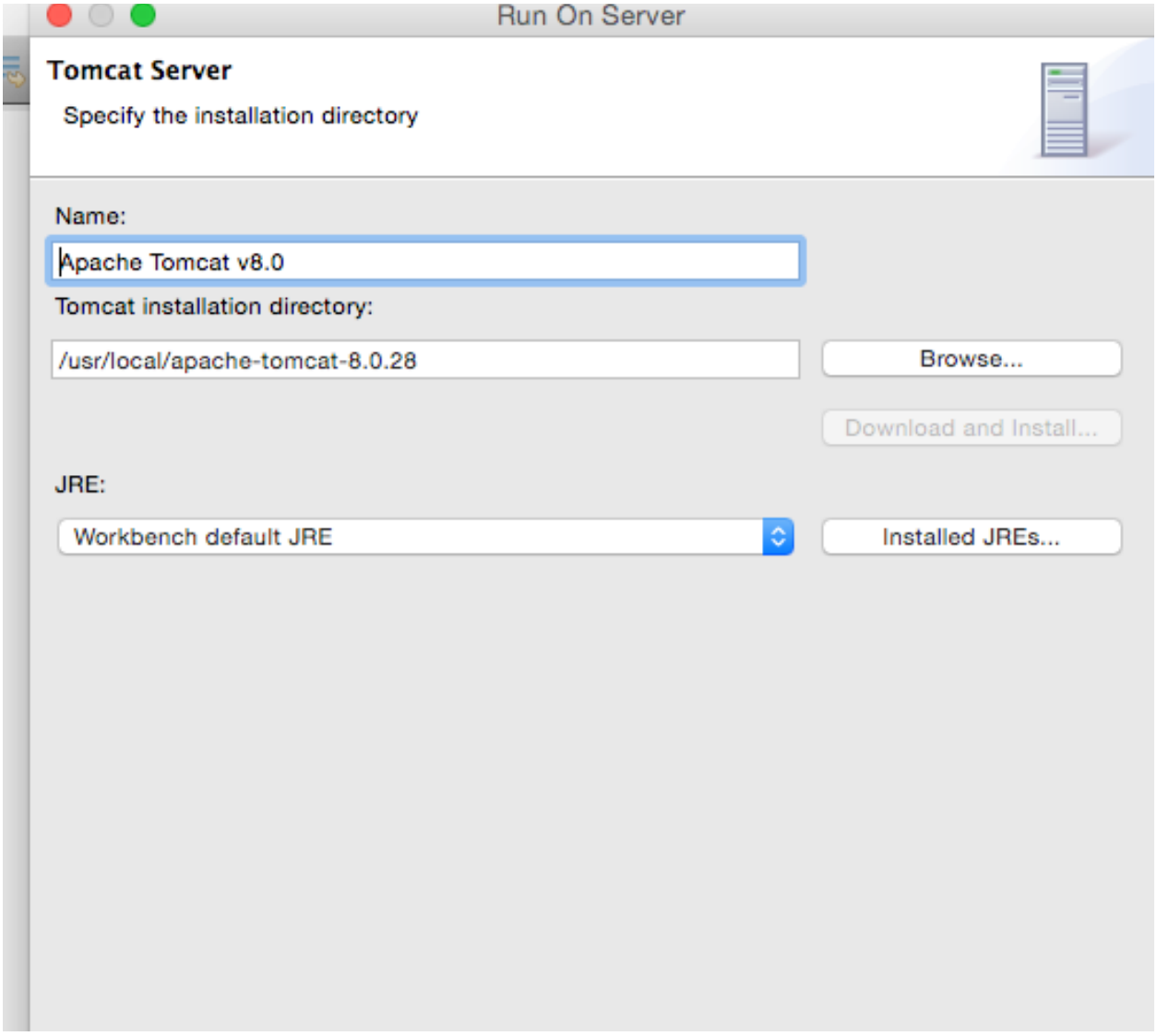
Right-click on the project and selecting `Run As -> Run on Server`.



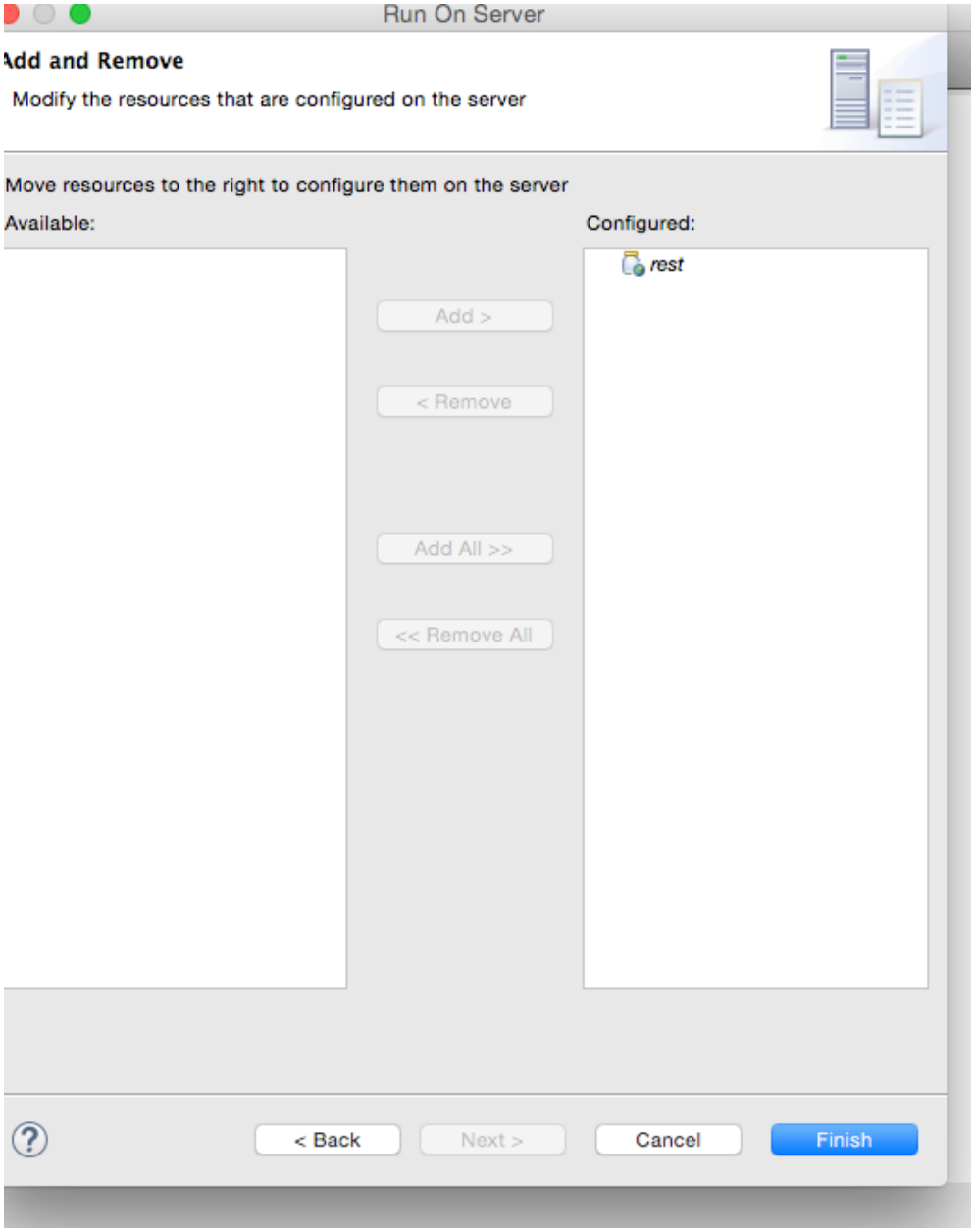
Choose the latest installation of Tomcat



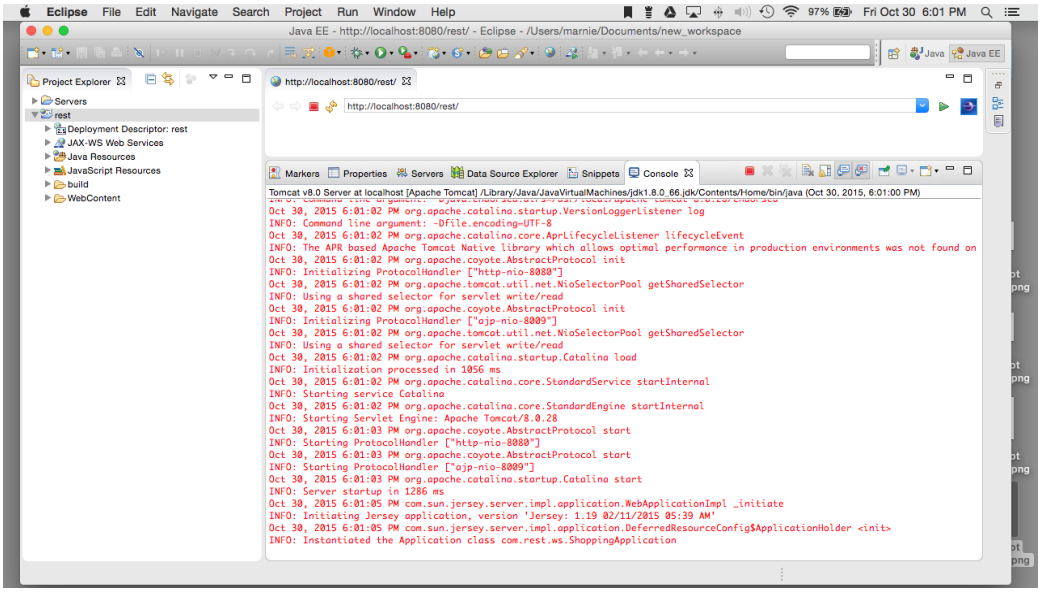
Map server to the installation directory of Tomcat



The Rest application is configured and click finish.

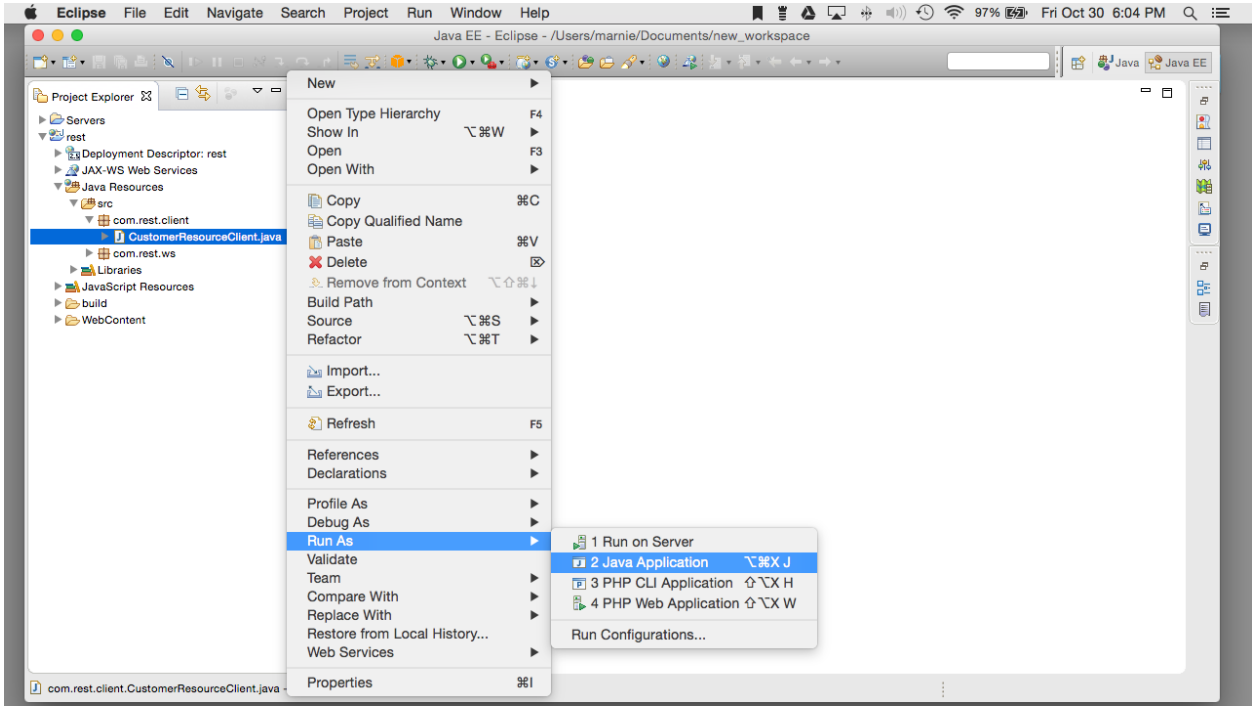


Console Output after running rest on Server
It's a blank browser window and the Shopping Applications initiated.

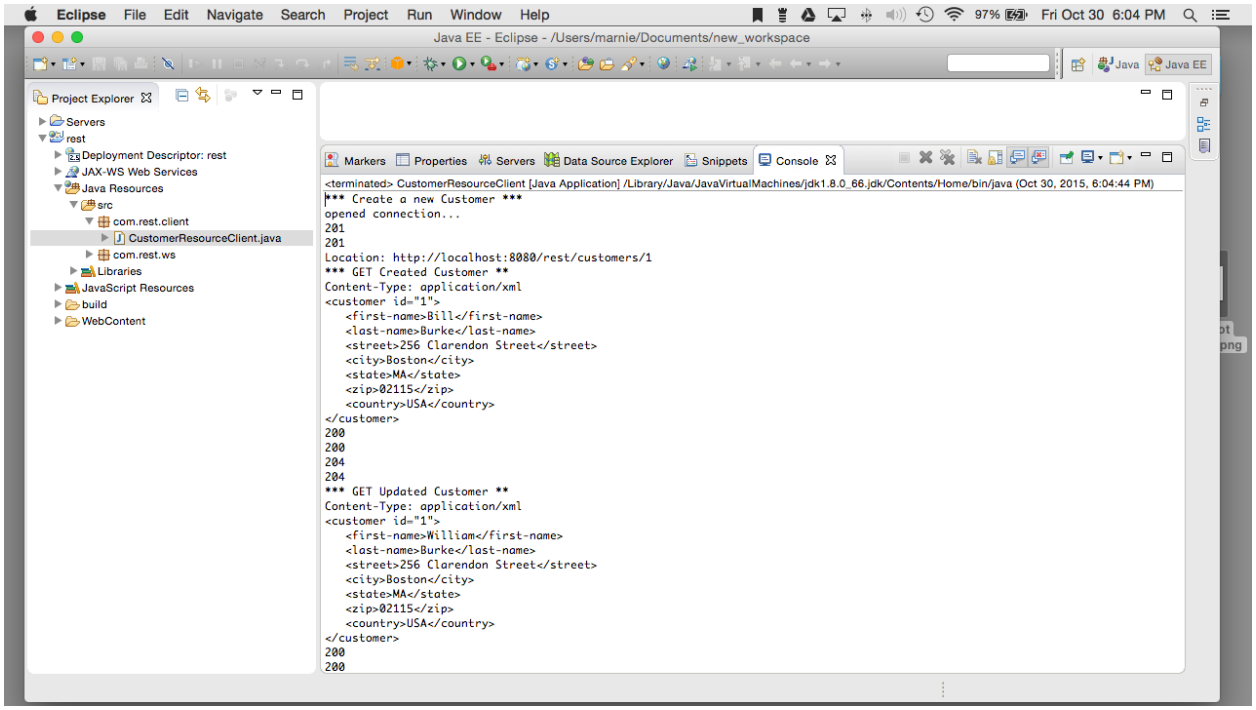



```
Oct 30, 2015 6:01:02 PM org.apache.catalina.core.StandardEngine startInternal
INFO: Starting Servlet Engine: Apache Tomcat/8.0.28
Oct 30, 2015 6:01:03 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-8080"]
Oct 30, 2015 6:01:03 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-nio-8009"]
Oct 30, 2015 6:01:03 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 1286 ms
Oct 30, 2015 6:01:05 PM com.sun.jersey.server.impl.application.WebApplicationImpl _initiate
INFO: Initiating Jersey application, version 'Jersey: 1.19 02/11/2015 05:39 AM'
Oct 30, 2015 6:01:05 PM com.sun.jersey.server.impl.application.DeferredResourceConfig$Appli
INFO: Instantiated the Application class com.rest.ws.ShoppingApplication
```

Run your client class `CustomerResourceClient` as a Java application.



Console Output after running `CustomerResourceClient` as a Java application.



```
*** Create a new Customer ***
opened connection...
201
201
Location: http://localhost:8080/rest/customers/1
*** GET Created Customer **
Content-Type: application/xml
<customer id="1">
  <first-name>Bill</first-name>
  <last-name>Burke</last-name>
  <street>256 Clarendon Street</street>
  <city>Boston</city>
  <state>MA</state>
  <zip>02115</zip>
  <country>USA</country>
</customer>
200
200
204
204
*** GET Updated Customer **
Content-Type: application/xml
<customer id="1">
  <first-name>William</first-name>
  <last-name>Burke</last-name>
  <street>256 Clarendon Street</street>
  <city>Boston</city>
  <state>MA</state>
  <zip>02115</zip>
  <country>USA</country>
</customer>
200
200
```

Problem 2:
Modify your `CustomerResource` class so that it could support a request that one particular `Customer` gets deleted. Also, add necessary methods that would allow your resource class to return a specified number of customers (2 or 3 is fine), starting with a particular customer id. To test new resource class you will have to modify your original client class `CustomerResourceClient` so that it creates 3 or 4 new customers before you could start requesting multiples of them back or before you could delete any of them. Code for creating 4 customers could be just four copy-pasted segments of existing code. Change customer names and perhaps their addresses. Run your code. Show your output. Submit the code you modified.
Points: 30

Copy and Paste the code from Problem 1 for creating a customer 4 times changing the name and some of the address of the customer. (This is the code that must be copied. Change and update the variable name for the customer accordingly.)

```
// Create a new customer
String newCustomer2 = "<customer>" + "<first-name>Sally</first-
name>" + "<last-name>Burke</last-name>"+ "<street>256 Clarendon
Street</street>" + "<city>Boston</city>" + "<state>MA</state>"+
"<zip>02115</zip>" + "<country>USA</country>" + "</customer>";

postUrl = new URL("http://localhost:8080/rest/customers");
connection = (HttpURLConnection) postUrl.openConnection();
System.out.println("opened connection...");
connection.setDoOutput(true);
connection.setInstanceFollowRedirects(false);
connection.setRequestMethod("POST");
connection.setRequestProperty("Content-Type", "application/
xml");
os = connection.getOutputStream();
os.write(newCustomer2.getBytes());
os.flush();
System.out.println(HttpURLConnection.HTTP_CREATED);
System.out.println(connection.getResponseCode());
System.out.println("Location: " +
connection.getHeaderField("Location"));
connection.disconnect();
```

Add Code in the CustomerResource class to support deleting a customer if given a customer ID as a parameter

```
@DELETE
@Path ("{id}")
public void deleteCustomer(@PathParam("id") int id) {
     customerDB.remove(id);
}
```

Add Code in the CustomerResourceClient class to make the request to delete the specific customer

```
//Delete a customer
System.out.println("*** DELETE a Customer **");
URL deleteURL = new URL("http://localhost:8080/rest/customers/
1");
connection = (HttpURLConnection) deleteURL.openConnection();
System.out.println("opened connection...");
connection.setRequestMethod("DELETE");

System.out.println(connection.getResponseCode());
connection.disconnect();
```

Add Code in the CustomerResource class to support displaying multiple customers if given a customer ID and a number of customers to display as parameters
(Modified the getCustomer method)

```
@GET
@Path("{id}/{numOfCustomersToReturn}")
@Produces("application/xml")
public StreamingOutput getCustomerById(@PathParam("id") int id,
@PathParam("numOfCustomersToReturn") int number) {
     final Customer customer = customerDB.get(id);
     if (customer == null) {
          throw new
WebApplicationException(Response.Status.NOT_FOUND);
     }
     return new StreamingOutput() {
          public void write(OutputStream outputStream) throws
IOException, WebApplicationException {
          outputAllCustomersById(outputStream, customer,number);
               }
          };
     }
```

```
protected void outputAllCustomersById(OutputStream os, Customer
cust, int customersToReturn) throws IOException {
     PrintStream writer = new PrintStream(os);
     Collection<Customer> customers = customerDB.values();
     Iterator<Customer> iter = customers.iterator();
     int count = 0;
     while (iter.hasNext() && count < customersToReturn) {
          cust = iter.next();
          writer.println("<customer id=\"" + cust.getId() +
"\">");
          writer.println("   <first-name>" + cust.getFirstName()
+ "</first-name>");
          writer.println("   <last-name>" + cust.getLastName() +
"</last-name>");
          writer.println("   <street>" + cust.getStreet() + "</
street>");
          writer.println("   <city>" + cust.getCity() + "</
city>");
          writer.println("   <state>" + cust.getState() + "</
state>");
          writer.println("   <zip>" + cust.getZip() + "</zip>");
          writer.println("   <country>" + cust.getCountry() +
"</country>");
          writer.println("</customer>");
               count++;
          }
     }
```

Add Code in the CustomerResourceClient class to make the request to display multiple customers if given a customer ID and a number of customers to display as parameters

```
// Get the customers with a starting ID and number of customers
System.out.println("*** GET the customers with a starting ID and
number of customers**");
getUrl = new URL("http://localhost:8080/rest/customers/2/2");
connection = (HttpURLConnection) getUrl.openConnection();
connection.setRequestMethod("GET");
System.out.println("Content-Type: " +
connection.getContentType());
reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));

line = reader.readLine();
while (line != null) {
```

```
     System.out.println(line);
     line = reader.readLine();
}
System.out.println(HttpURLConnection.HTTP_OK);
System.out.println(connection.getResponseCode());
connection.disconnect();
```

Console Output

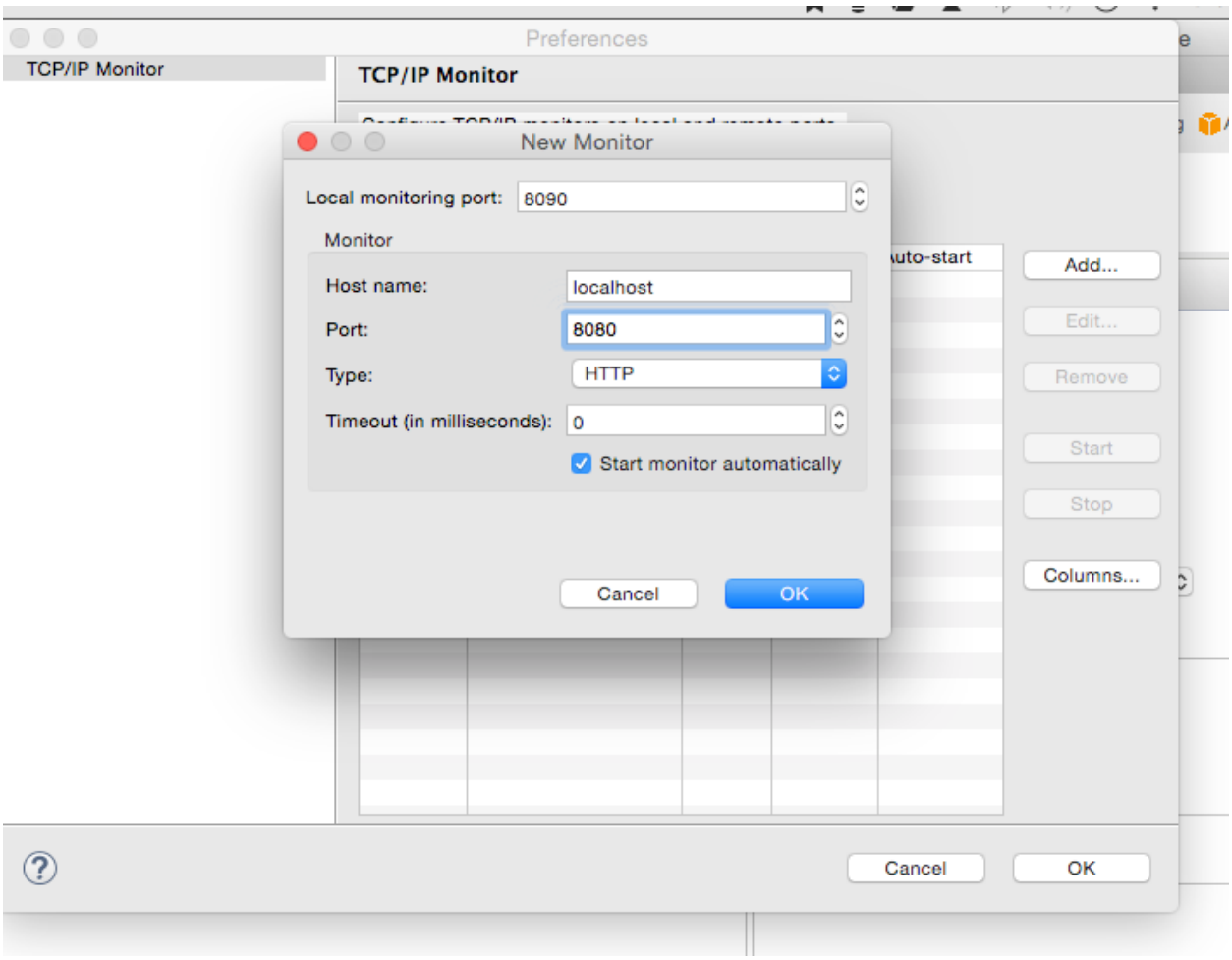
```
*** Create a new Customer ***
opened connection...
201
201
Location: http://localhost:8080/rest/customers/1
*** GET Created Customer **
Content-Type: application/xml
<customer id="1">
   <first-name>Bill</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
204
204
*** GET Updated Customer **
Content-Type: application/xml
<customer id="1">
   <first-name>William</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
opened connection...
201
201
Location: http://localhost:8080/rest/customers/2
```

```
*** GET Created Customer 2**
Content-Type: application/xml
<customer id="2">
   <first-name>Sally</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
opened connection...
201
201
Location: http://localhost:8080/rest/customers/3
*** GET Created Customer 3**
Content-Type: application/xml
<customer id="3">
   <first-name>Jimmy</first-name>
   <last-name>Smith</last-name>
   <street>123 Main Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
opened connection...
201
201
Location: http://localhost:8080/rest/customers/4
*** GET Created Customer 4**
Content-Type: application/xml
<customer id="4">
   <first-name>Agatha</first-name>
   <last-name>Edwards</last-name>
   <street>123 Pine Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
```

```
200
*** Verify Customer 1 Exists*
Content-Type: application/xml
<customer id="1">
   <first-name>William</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
*** DELETE a Customer **
opened connection...
204
*** GET the customers with a starting ID and number of
customers**
Content-Type: application/xml
<customer id="2">
   <first-name>Sally</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
<customer id="3">
   <first-name>Jimmy</first-name>
   <last-name>Smith</last-name>
   <street>123 Main Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
```

Problem 3:
Capture HTTP requests and responses for your `GET`, `PUT`, `POST` and `DELETE` method calls from Problem 2. Use TCPMon or some other HTTP monitoring tool to convince yourself that network traffic is indeed as advertised, meaning that you are sending HTTP messages with expected content and receiving HTTP messages with a similar content. Show your HTTP output (`GET`, `PUT`, `POST`, `DELETE`).
Points: 10

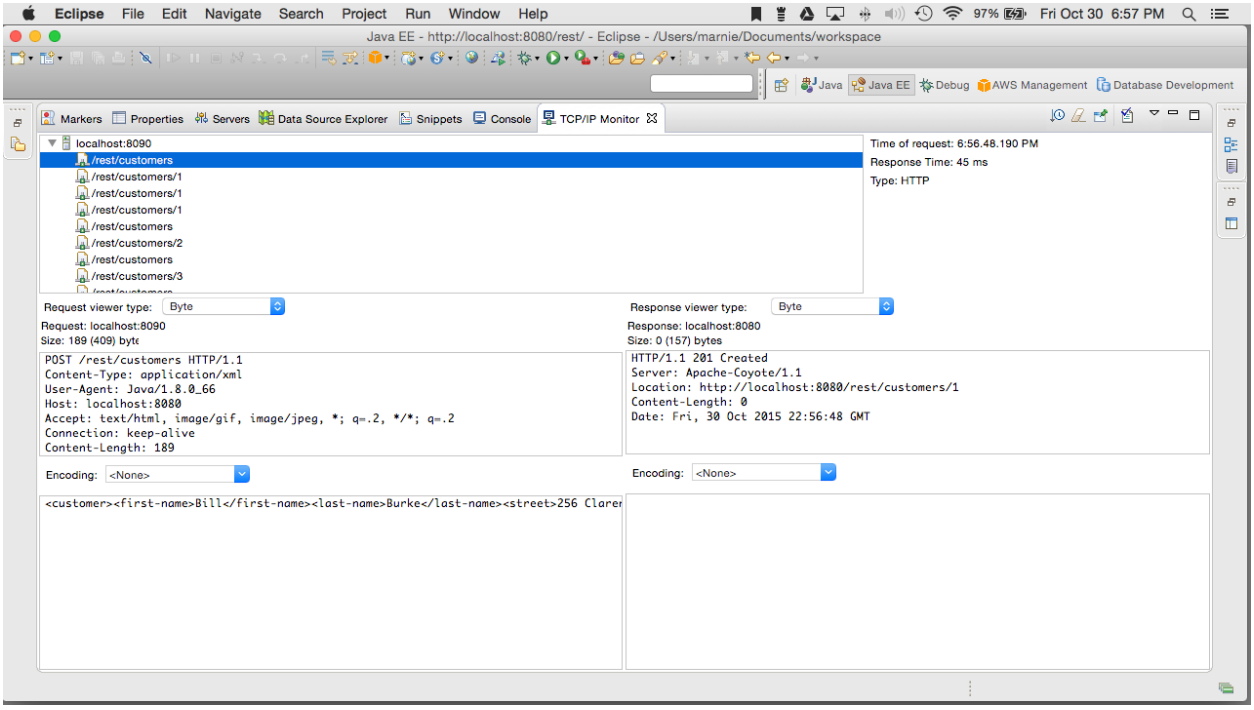
Configure TCP/IP Monitoring in Eclipse



Edit all links in CustomerResourceClient to point to the monitoring port 8090 instead of 8080

```
URL postUrl = new URL("http://localhost:8080/rest/customers");
to
URL postUrl = new URL("http://localhost:8090/rest/customers");
```

Run the CustomerResourceClient as an application and view TCP/IP monitor in Eclipse



You will see a list of all of the requests and can scan them for each request and response header.

HTTP GET OUTPUT

Get a Customer (Similar GET headers exist for each customer)

Request

```
GET /rest/customers/1 HTTP/1.1
User-Agent: Java/1.8.0_66
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml
Content-Length: 226
Date: Fri, 30 Oct 2015 22:56:48 GMT
```

Get several Customers

Request

```
GET /rest/customers/2/2 HTTP/1.1
User-Agent: Java/1.8.0_66
```

```
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

Response

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: application/xml
Content-Length: 449
Date: Fri, 30 Oct 2015 22:56:48 GMT
```

HTTP PUT OUTPUT

Update Customer

Request

```
PUT /rest/customers/1 HTTP/1.1
Content-Type: application/xml
User-Agent: Java/1.8.0_66
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 192
```

Response

```
HTTP/1.1 204 No Content
Server: Apache-Coyote/1.1
Date: Fri, 30 Oct 2015 22:56:48 GMT
```

HTTP POST OUTPUT

Create Customer 1 (Similar POST headers exist for each customer)

Request

```
POST /rest/customers HTTP/1.1
Content-Type: application/xml
User-Agent: Java/1.8.0_66
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
Content-Length: 189
```

Response

```
HTTP/1.1 201 Created
Server: Apache-Coyote/1.1
Location: http://localhost:8080/rest/customers/1
Content-Length: 0
```

```
Date: Fri, 30 Oct 2015 22:56:48 GMT
```

HTTP DELETE OUTPUT

Request

```
DELETE /rest/customers/1 HTTP/1.1
User-Agent: Java/1.8.0_66
Host: localhost:8080
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive
```

Response

```
HTTP/1.1 204 No Content
Server: Apache-Coyote/1.1
Date: Fri, 30 Oct 2015 22:56:48 GMT
```

Problem 4:

Launch 2 AWS EC2 instances (Windows or Linux) in the same availability zone using Amazon Console or AWS CLI. Install Java JDK and Tomcat on one AWS Instance and startup Tomcat. Make sure you have security group set up (SSH, HTTP, etc.) and inbound rules. This is your Server and serves as your host of your RESTful web service. You can also use an AWS EC2 Instance that has Tomcat pre-installed (make sure Java JDK is installed and Tomcat is running. On your Server check that you can reach Tomcat in a browser: http://localhost:8080/). Assign a private IP address to your Server by creating an Elastic IP. You can do this in AWS Console or AWS CLI. Export your project (CustomerResource class) from problem 2 in Eclipse (as rest.war file) and copy it to your Server's Tomcat \webapps directory. Make sure it has proper executable permissions. Restart your Tomcat server.

Launch a second AWS EC2 instance (Windows or Linux similar to your first EC2 instance with security group then install Java JDK.

On your laptop in Eclipse - add the private IP address of your Server to your CustomerResourceClient class for post, get, delete customer and recompile.

Copy your .jar folder to similar folder structure as you used in your package to your client (2nd AWS EC2 Instance). . Run your class: java … You should see customers being created, queried, … Show your output. Submit the code you have modified.

Points: 30

Problem 4:
Launch an AWS EC2 instance (Windows or Linux) using Amazon Console or AWS CLI.
I named this instance WebServer.

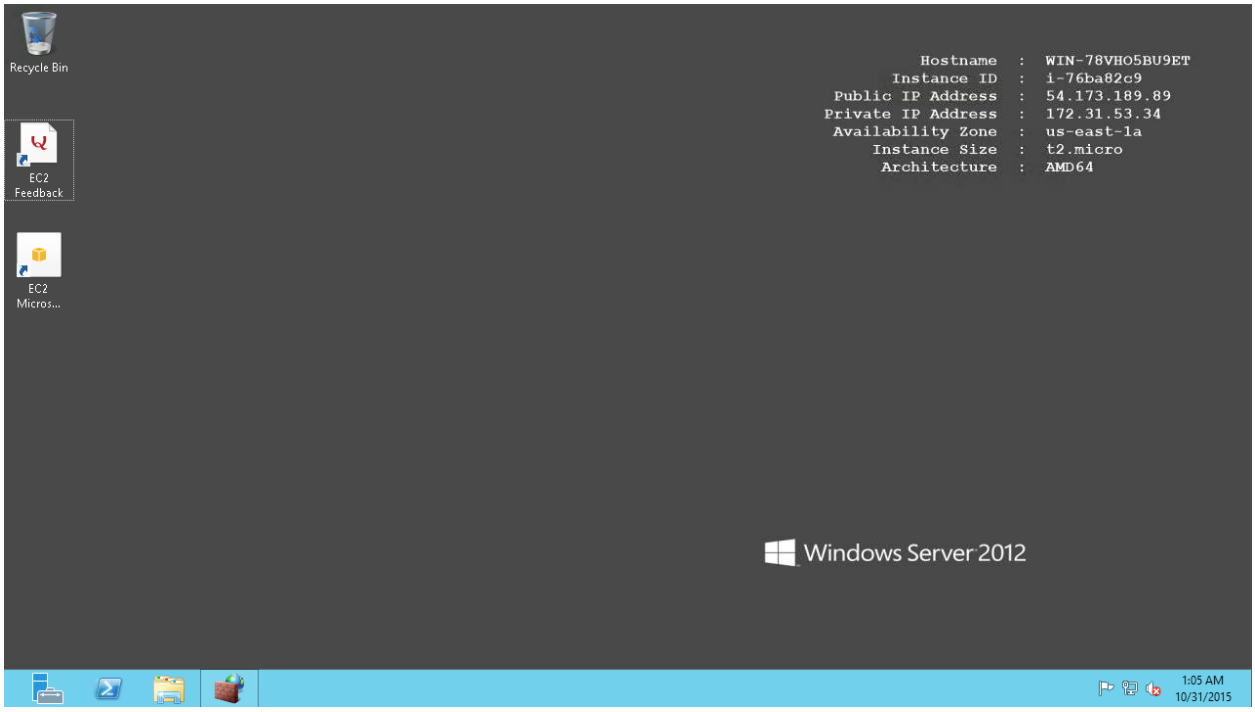
WebServer	i-76ba82c9	t2.micro	us-east-1a	running	2/2 checks ...	None	ec2-54-173-189-89.co...	54.173.189.89

Instance: i-76ba82c9 (WebServer) Public DNS: ec2-54-173-189-89.compute-1.amazonaws.com

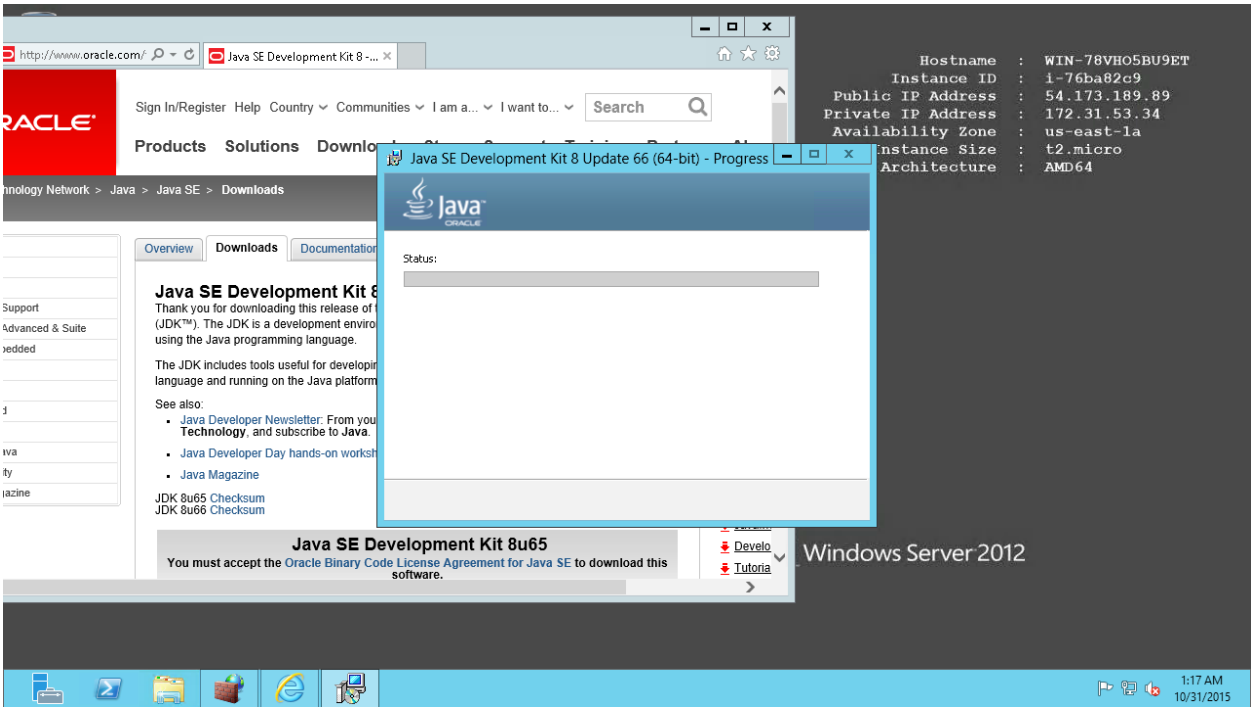
Description | Status Checks | Monitoring | Tags

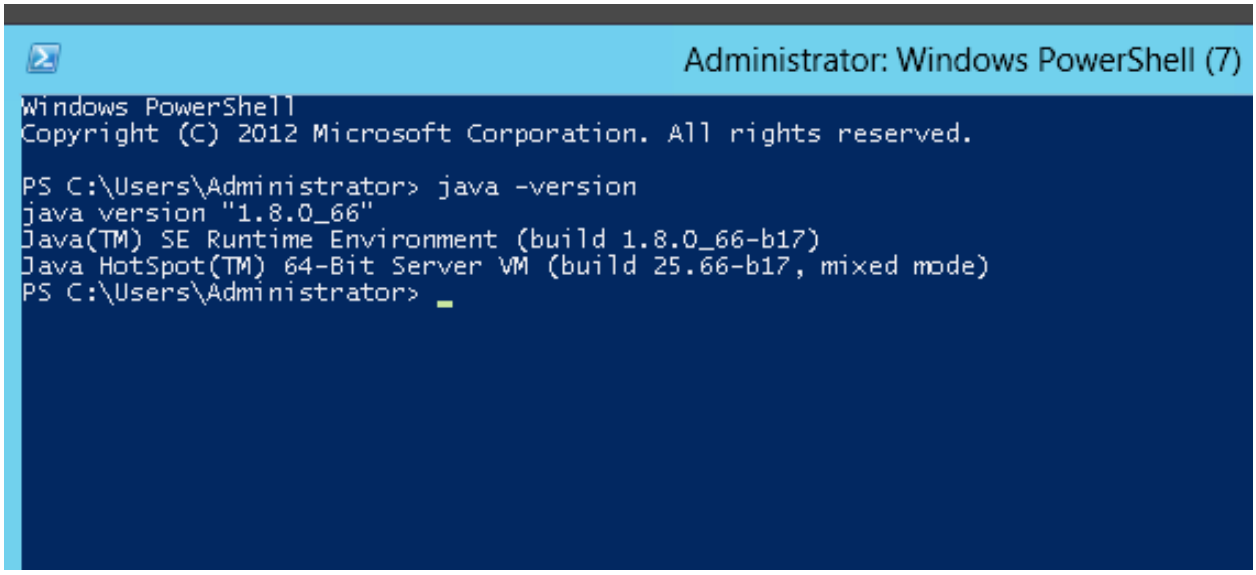
Instance ID	i-76ba82c9	Public DNS	ec2-54-173-189-89.compute-1.amazonaws.com
Instance state	running	Public IP	54.173.189.89
Instance type	t2.micro	Elastic IP	-
Private DNS	ip-172-31-53-34.ec2.internal	Availability zone	us-east-1a
Private IPs	172.31.53.34	Security groups	RDPGroup. view rules
Secondary private IPs		Scheduled events	No scheduled events
VPC ID	vpc-914989f5	AMI ID	Windows_Server-2012-RTM-English-64Bit-Base-2015.10.26 (ami-35e0bc50)
Subnet ID	subnet-0109582a	Platform	windows
Network interfaces	eth0	IAM role	-
Source/dest. check	True	Key pair name	CliKeyPair

Log into WebServer with Remote Desktop Client

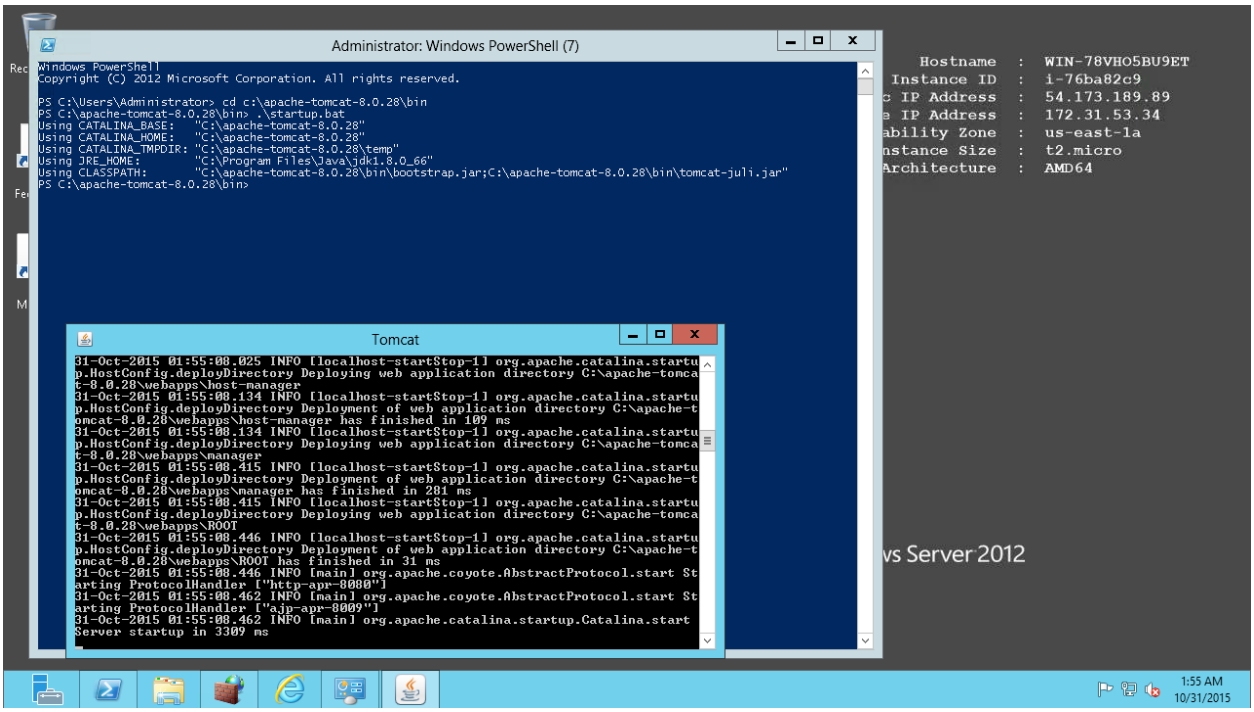


Install Java JDK and Tomcat on the AWS Instance

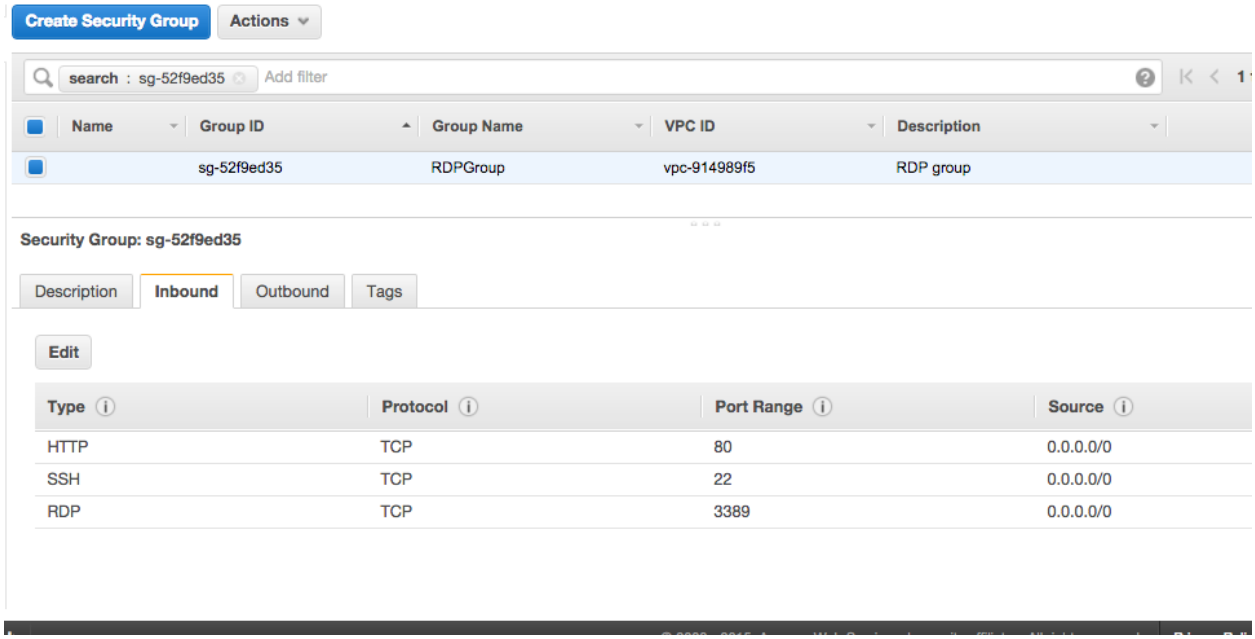




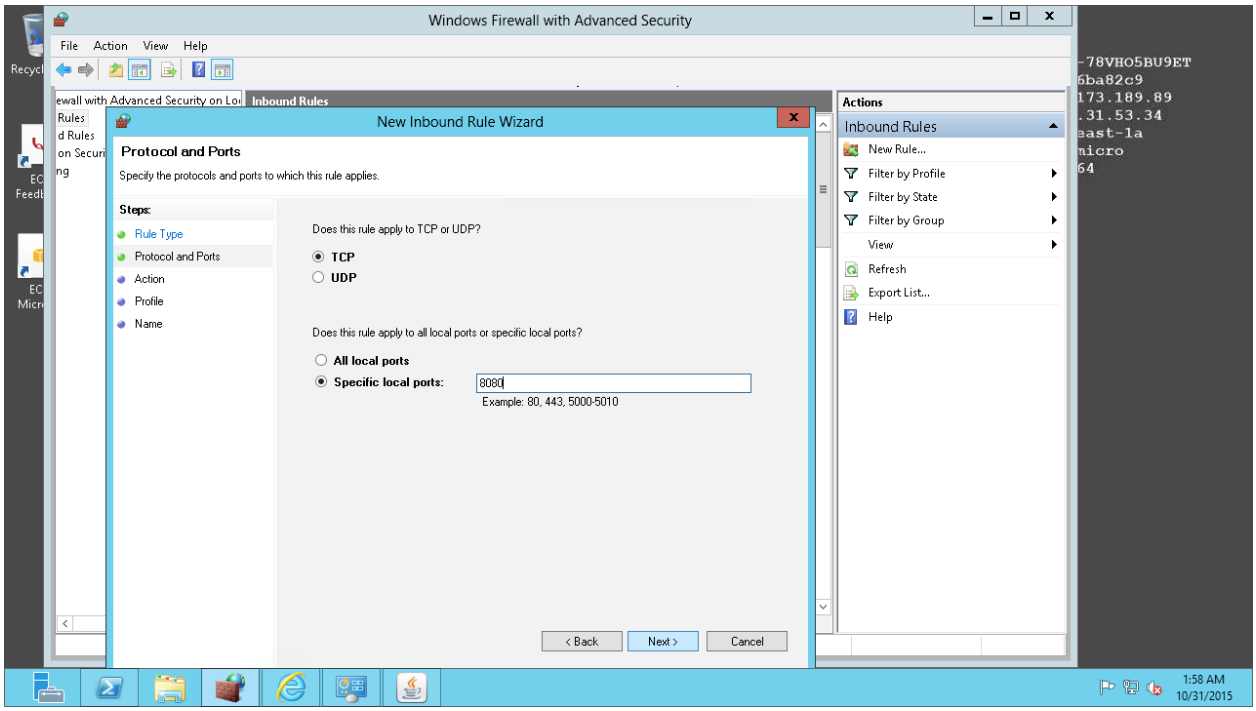
Startup Tomcat.

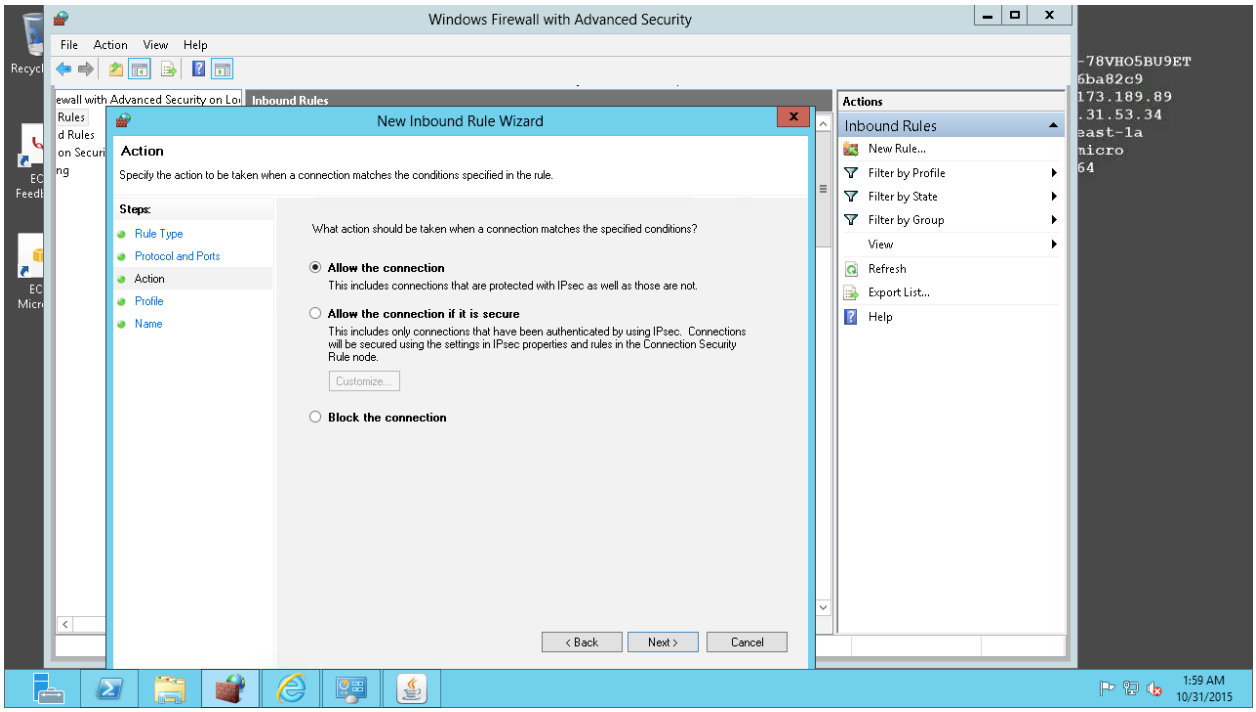


Make sure you have a security group set up on the Server EC2 Instance (SSH, HTTP, etc.) with inbound rules open to your local IP address or all IP addresses.

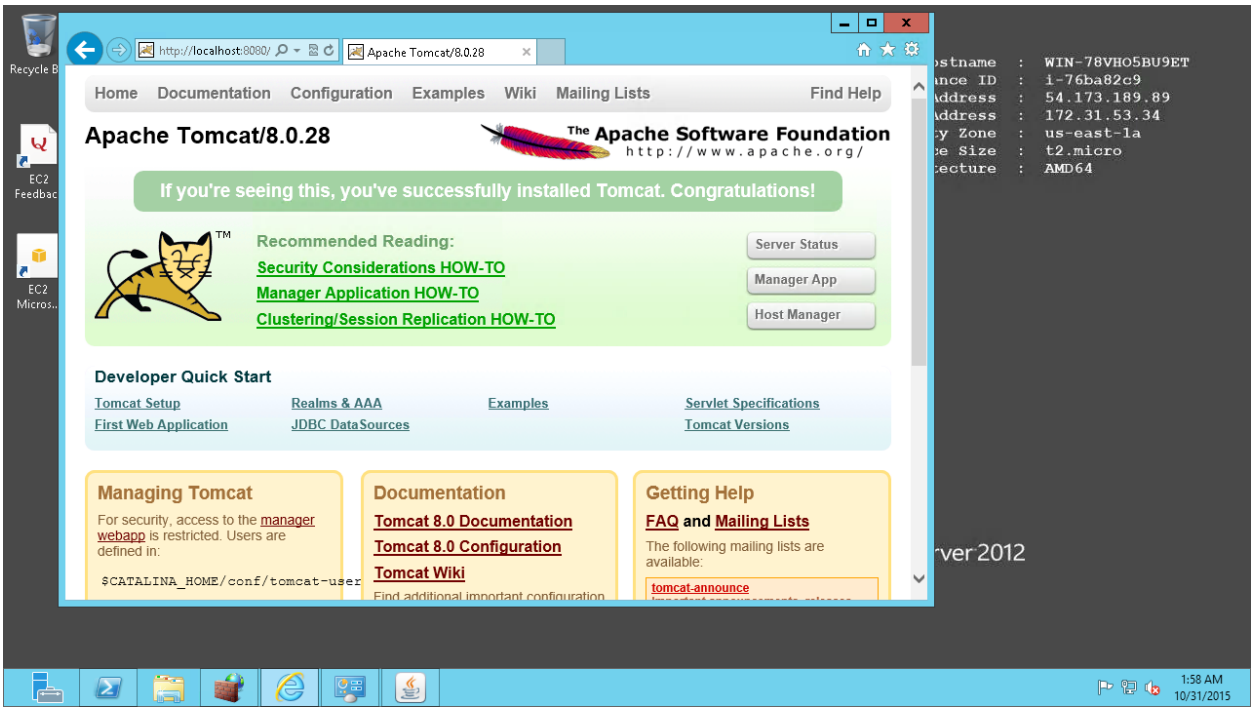


Make sure server has port 8080 open to incoming traffic



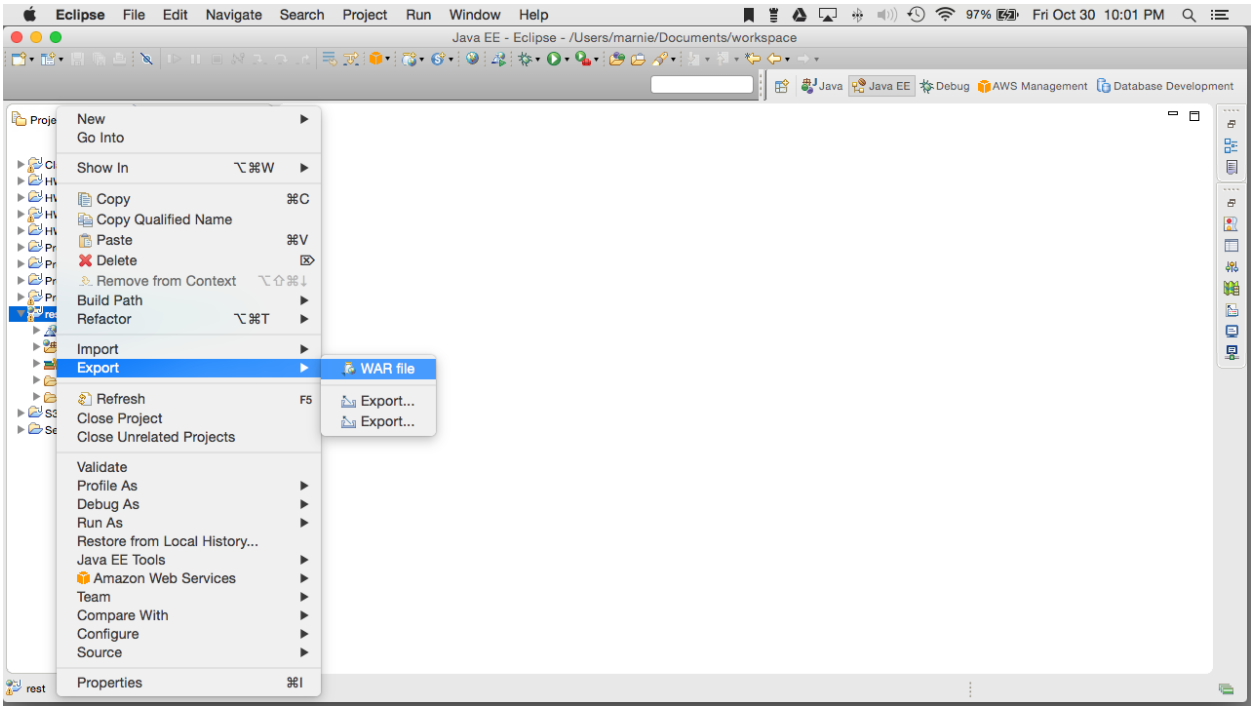


On your Server check that you can reach Tomcat in a browser: http://localhost:8080/

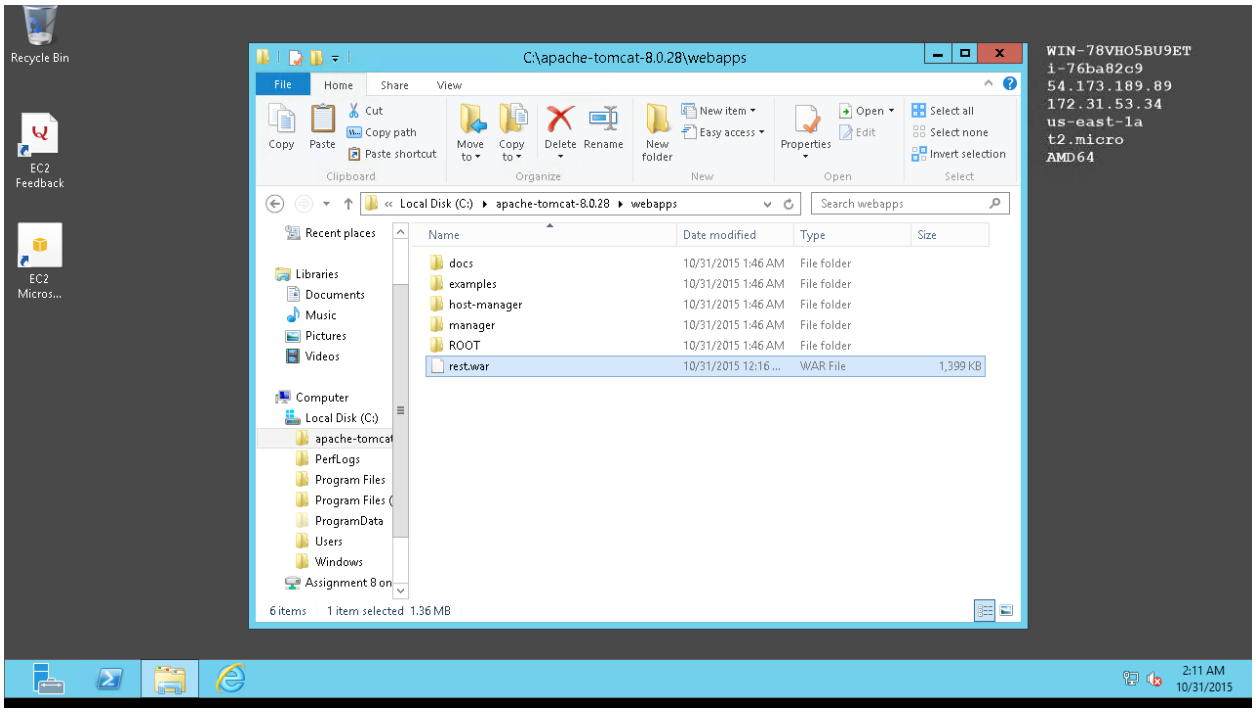


Note the Public and Private IP addresses of the Server Instance. Use the Private IP address in the client code running on the client instance. Use the Public IP address to test the server remotely from your local PC.

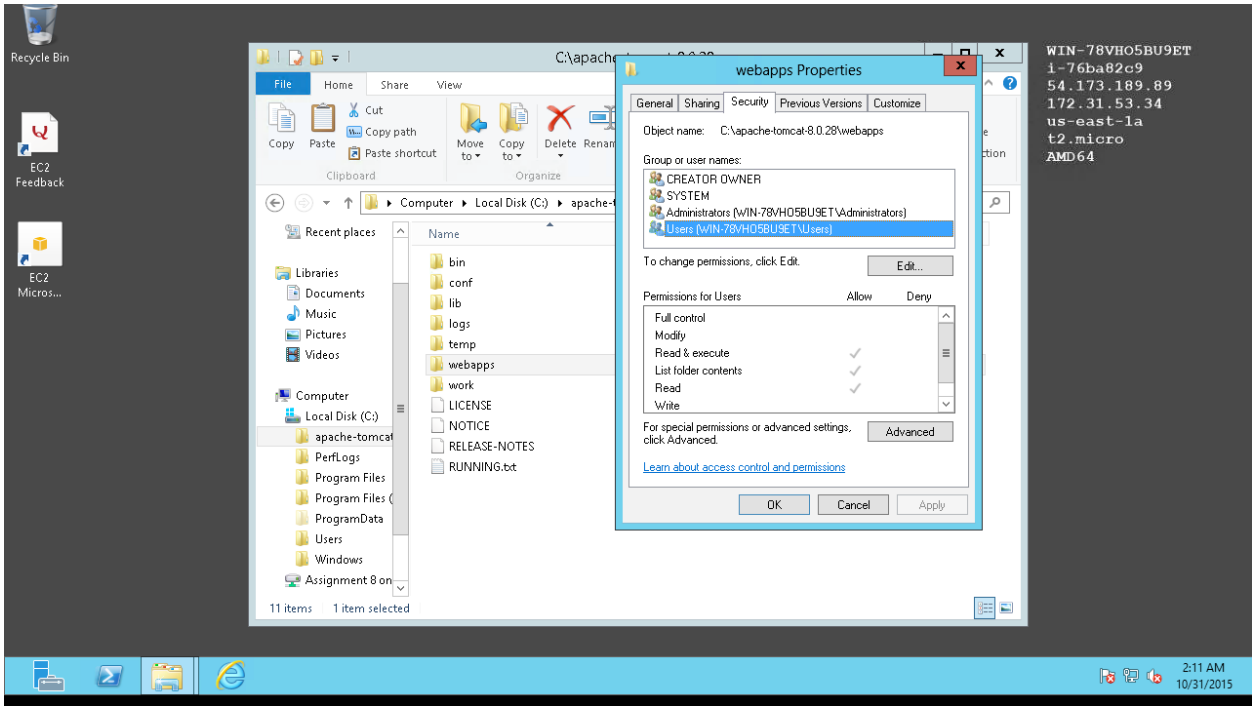
Export your project (CustomerResource class) from problem 2 in Eclipse (as rest.war file)



Copy rest.war file to your Server's Tomcat \webapps directory.



Make sure your Server's Tomcat \webapps directory has proper executable permissions.



Restart your Tomcat server.

```
S C:\Users\Administrator> cd c:\apache-tomcat-8.0.28\bin
S C:\apache-tomcat-8.0.28\bin> .\startup.bat
sing CATALINA_BASE:   "C:\apache-tomcat-8.0.28"
sing CATALINA_HOME:   "C:\apache-tomcat-8.0.28"
sing CATALINA_TMPDIR: "C:\apache-tomcat-8.0.28\temp"
sing JRE_HOME:        "C:\Program Files\Java\jdk1.8.0_66"
sing CLASSPATH:       "C:\apache-tomcat-8.0.28\bin\bootstrap.jar;C:\apache-tomcat-8.0.28\bin\tomcat-juli.jar"
S C:\apache-tomcat-8.0.28\bin>
```

Test the server (TomCat, Java JDK and rest.war) from your local PC using its public IP address

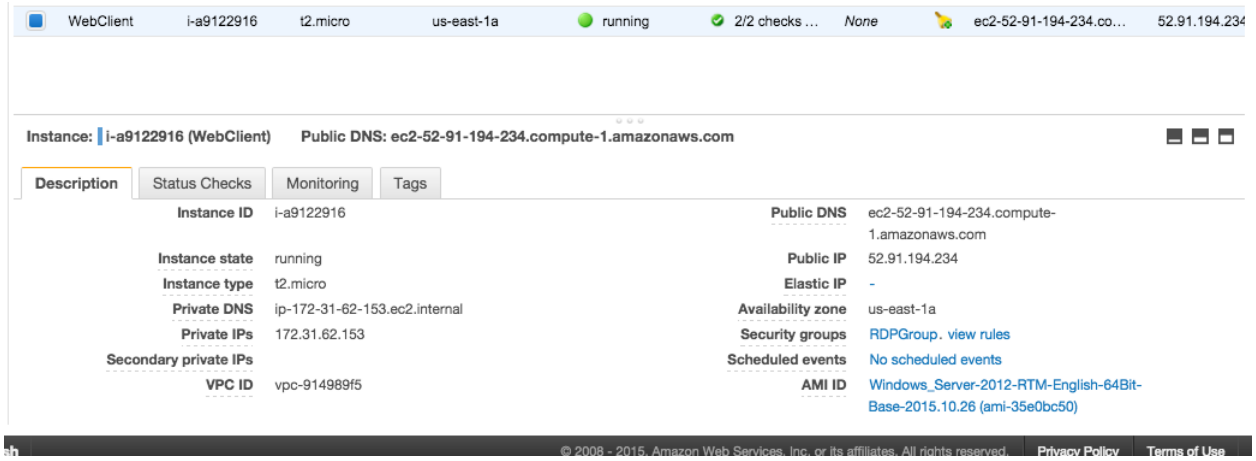
Output in console

```
*** Create a new Customer ***
opened connection...
201
201
Location: http://54.173.189.89:8080/rest/customers/1
*** GET Created Customer **
Content-Type: application/xml
<customer id="1">
   <first-name>Bill</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
204
204
*** GET Updated Customer **
Content-Type: application/xml
<customer id="1">
   <first-name>William</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
opened connection...
```

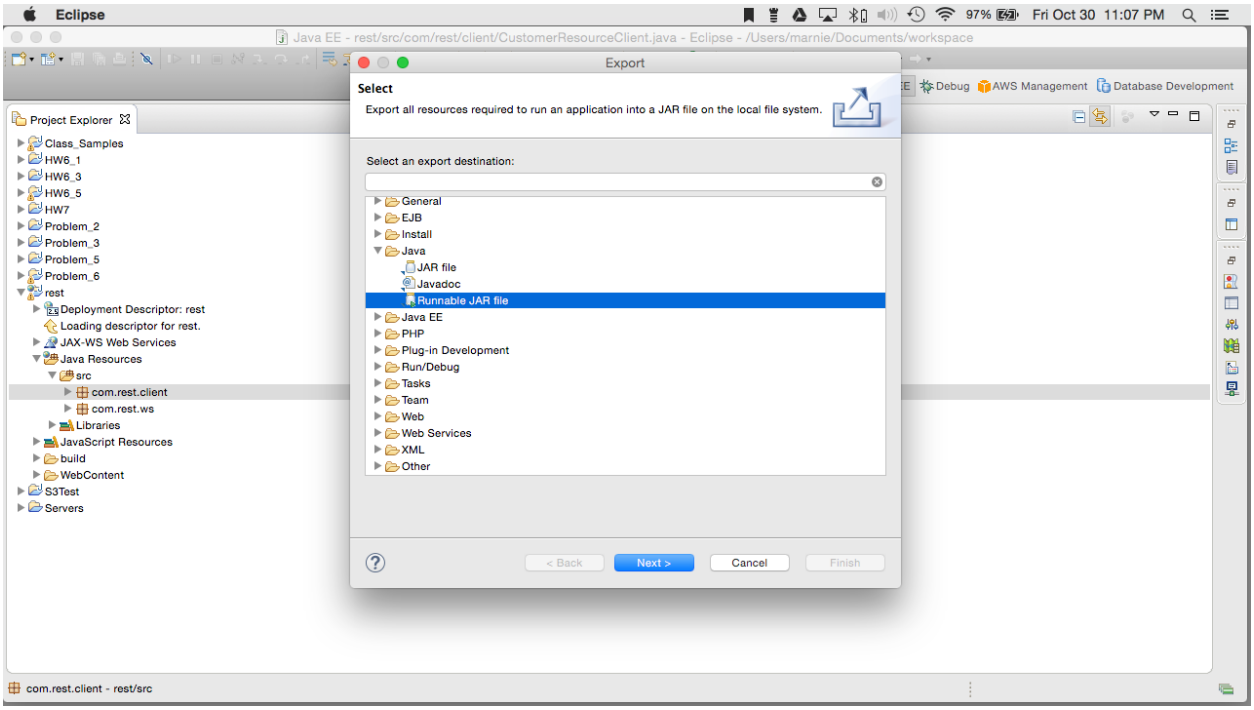
```
201
201
Location: http://54.173.189.89:8080/rest/customers/2
*** GET Created Customer 2**
Content-Type: application/xml
<customer id="2">
   <first-name>Sally</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
opened connection...
201
201
Location: http://54.173.189.89:8080/rest/customers/3
*** GET Created Customer 3**
Content-Type: application/xml
<customer id="3">
   <first-name>Jimmy</first-name>
   <last-name>Smith</last-name>
   <street>123 Main Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
opened connection...
201
201
Location: http://54.173.189.89:8080/rest/customers/4
*** GET Created Customer 4**
Content-Type: application/xml
<customer id="4">
   <first-name>Agatha</first-name>
   <last-name>Edwards</last-name>
   <street>123 Pine Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
```

```
   <country>USA</country>
</customer>
200
200
*** Verify Customer 1 Exists*
Content-Type: application/xml
<customer id="1">
   <first-name>William</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
*** DELETE a Customer **
opened connection...
204
*** GET the customers with a starting ID and number of
customers**
Content-Type: application/xml
<customer id="2">
   <first-name>Sally</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
<customer id="3">
   <first-name>Jimmy</first-name>
   <last-name>Smith</last-name>
   <street>123 Main Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
```

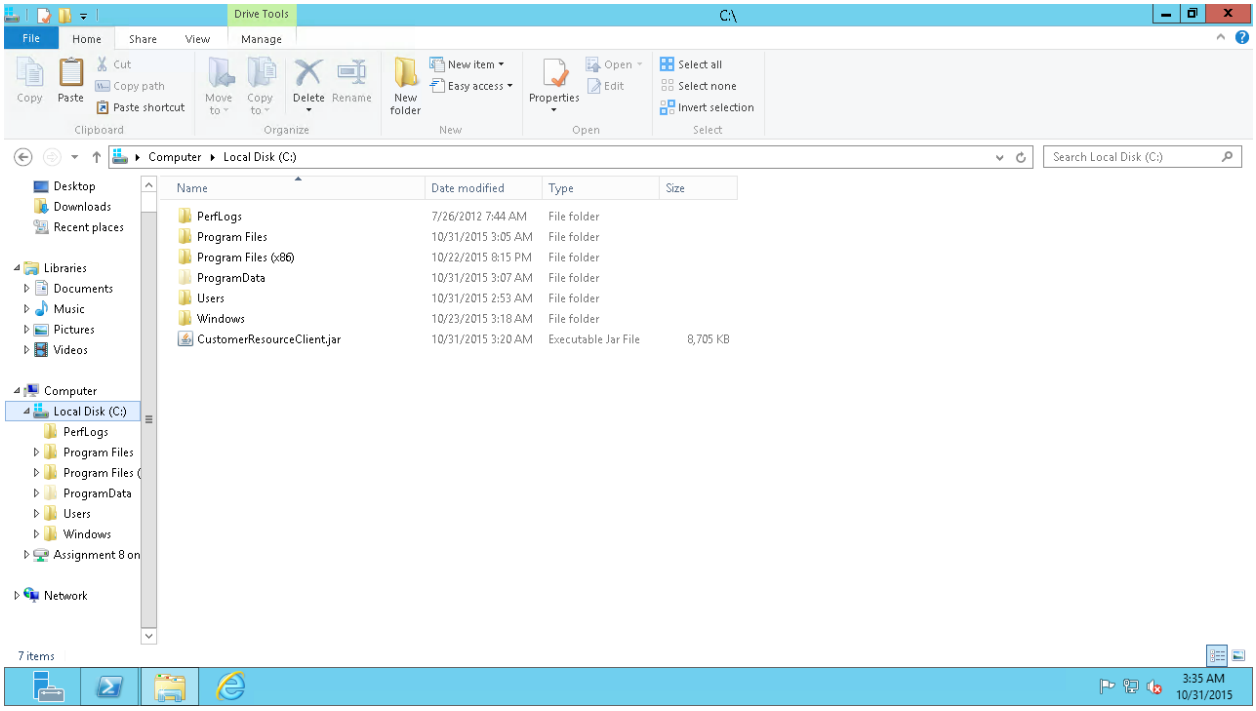
Launch a second AWS EC2 instance (Windows or Linux similar to your first EC2 instance) in the same availability zone with the same security group settings. (Install Java JDK on this new instance if needed.)



On your laptop in Eclipse - add the private IP address of your Server to your CustomerResourceClient class for post, get, delete customer and recompile.



Copy your .jar folder to (created from the package com.rest.client) to your client AWS EC2 Instance.



Run CustomerResourceClient.jar

Output on Client

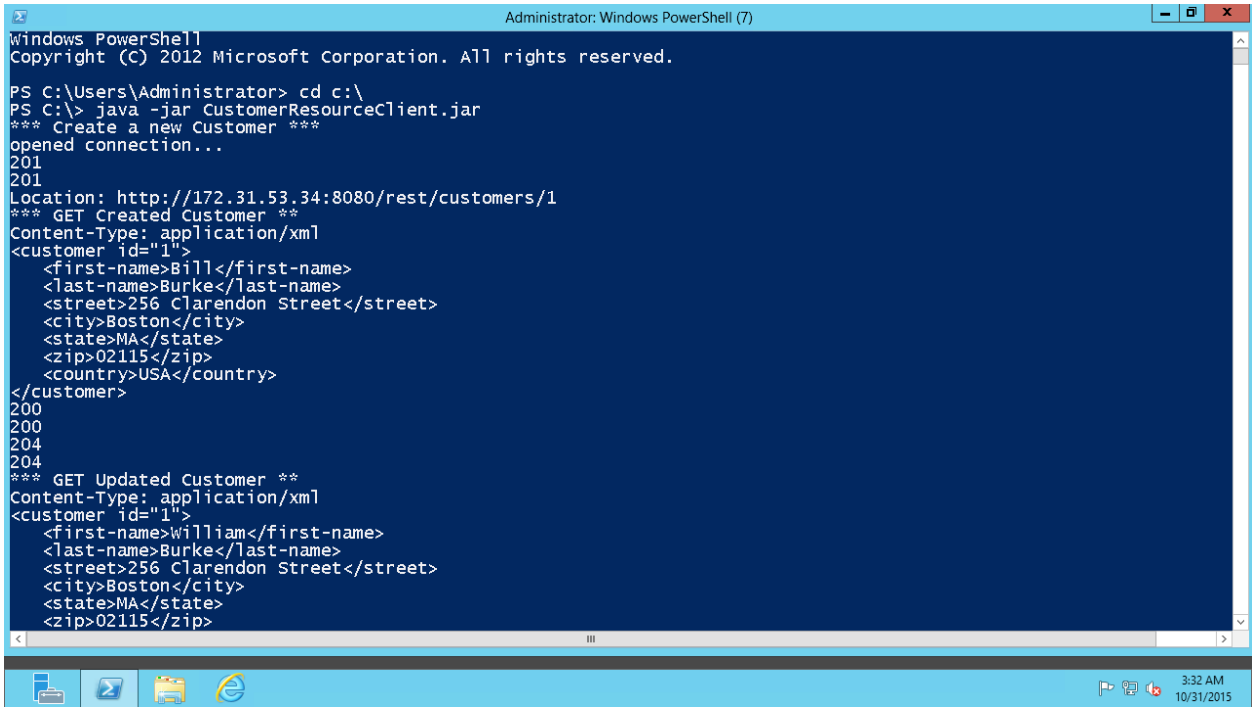
```
Windows PowerShell
Copyright (C) 2012 Microsoft Corporation. All rights reserved.

PS C:\Users\Administrator> cd c:\
PS C:\> java -jar CustomerResourceClient.jar
*** Create a new Customer ***
opened connection...
201
201
Location: http://172.31.53.34:8080/rest/customers/1
*** GET Created Customer **
Content-Type: application/xml
<customer id="1">
   <first-name>Bill</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
```

```
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
204
204
*** GET Updated Customer **
Content-Type: application/xml
<customer id="1">
   <first-name>William</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
opened connection...
201
201
Location: http://172.31.53.34:8080/rest/customers/2
*** GET Created Customer 2**
Content-Type: application/xml
<customer id="2">
   <first-name>Sally</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
opened connection...
201
201
Location: http://172.31.53.34:8080/rest/customers/3
*** GET Created Customer 3**
Content-Type: application/xml
<customer id="3">
   <first-name>Jimmy</first-name>
```

```
   <last-name>Smith</last-name>
   <street>123 Main Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
opened connection...
201
201
Location: http://172.31.53.34:8080/rest/customers/4
*** GET Created Customer 4**
Content-Type: application/xml
<customer id="4">
   <first-name>Agatha</first-name>
   <last-name>Edwards</last-name>
   <street>123 Pine Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
*** Verify Customer 1 Exists*
Content-Type: application/xml
<customer id="1">
   <first-name>William</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
*** DELETE a Customer **
opened connection...
204
*** GET the customers with a starting ID and number of
customers**
Content-Type: application/xml
<customer id="2">
```

```
   <first-name>Sally</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
<customer id="3">
   <first-name>Jimmy</first-name>
   <last-name>Smith</last-name>
   <street>123 Main Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
PS C:\>
```



Administrator: Windows PowerShell (7)

```
</customer>
200
200
204
204
*** GET Updated Customer **
Content-Type: application/xml
<customer id="1">
   <first-name>William</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
opened connection...
201
201
Location: http://172.31.53.34:8080/rest/customers/2
*** GET Created Customer 2**
Content-Type: application/xml
<customer id="2">
   <first-name>Sally</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
```

3:39 AM
10/31/2015

Administrator: Windows PowerShell (7)

```
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
*** DELETE a Customer **
opened connection...
204
*** GET the customers with a starting ID and number of customers**
Content-Type: application/xml
<customer id="2">
   <first-name>Sally</first-name>
   <last-name>Burke</last-name>
   <street>256 Clarendon Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
<customer id="3">
   <first-name>Jimmy</first-name>
   <last-name>Smith</last-name>
   <street>123 Main Street</street>
   <city>Boston</city>
   <state>MA</state>
   <zip>02115</zip>
   <country>USA</country>
</customer>
200
200
PS C:\>
```

3:33 AM
10/31/2015

Output on Server