**PaaS - Exploration and Deployment of RESTful Java web services with AWS Elastic Beanstalk**

**Objective:** Get practical experience in RESTful Java Web services development and deployment on Cloud PaaS with AWS Elastic Beanstalk as well as in creating and running web service clients.

**Tasks:**

1. Exploration and deployment of a RESTful Java Web Service with the Jersey framework
2. Testing a RESTful Java Web Service using SoapUI tool
3. Exploration and execution of a Web Service Java client
4. Deployment of a RESTful Java Web Service on AWS Elastic Beanstalk PaaS

**Lab environment**:

* Java 8 (JDK 1.8.0)
* Eclipse IDE for Enterprise Java developers
  + AWS Toolkit for Eclipse 2.0
  + Maven integration for Eclipse WTP 1.2.0
* Jersey framework 2.25.1
* Apache Tomcat web server 8.5
* SoapUI tool

*Contents*

[Step 1: Running Eclipse IDE and opening the Java project](#_heading=h.1fob9te) 3

[Running Eclipse IDE](#_heading=h.2vqyz5gs5mht) 3

[Opening the RESTServerCalc project](#_heading=h.enbdcyq63gnc) 3

[Configuring the Tomcat Web Server](#_heading=h.nd3awom4mzs4) 5

[Step 2: Exploring RESTful Java Web Service and Client](#_heading=h.lbzpwqs3en1l) 9

[Exploring the RESTServerCalc.java](#_heading=h.2et92p0) 9

[Launching and testing the RESTful Web Service](#_heading=h.9d8twi1v4r4v) 13

[Run a Web Service Java Client](#_heading=h.wak0phl7j0gs) 17

[Step 3: Deploying a Web Service to AWS Elastic Beanstalk PaaS](#_heading=h.vbxbscwawv7j) 20

[Installing AWS Plugin for Eclipse](#_heading=h.4kbe37qj1itq) 20

[Re-running Eclipse with the Jaxb library](#_heading=h.h0hutzhnadkt) 26

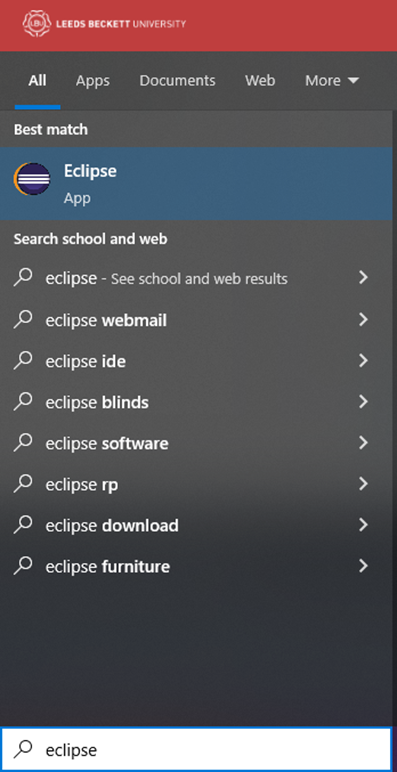
[Launching a Web Service on the AWS Elastic Beanstalk PaaS](#_heading=h.s3hgt33w7768) 27

[Test the RESTful Java Web Service deployed at AWS Elastic Beanstalk](#_heading=h.2s8eyo1) 31

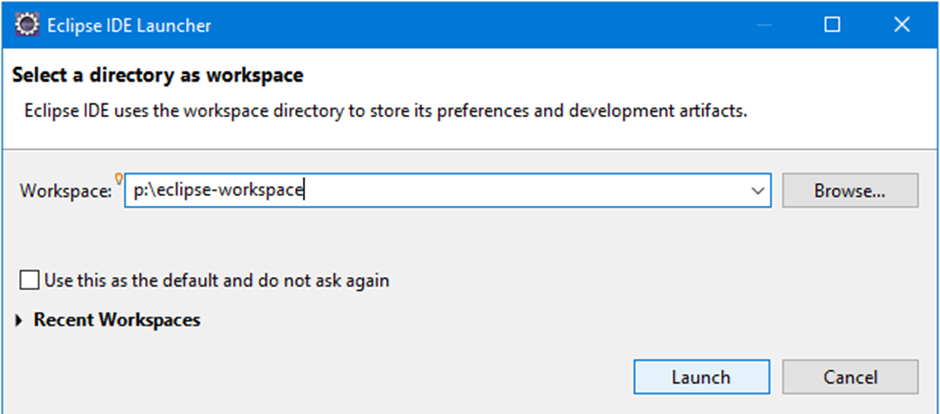
[Step 4: Deleting AWS Elastic Beanstalk resources after use](#_heading=h.17dp8vu) 32

# Step 1: Running Eclipse IDE and opening the Java project

## Running Eclipse IDE



Confirm the default workspace location:

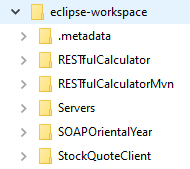


## Opening the RESTServerCalc project

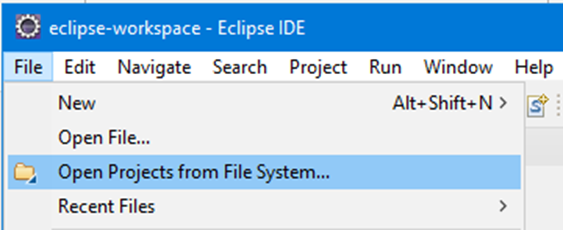
Download the eclipse-workspace.zip file from the module page.

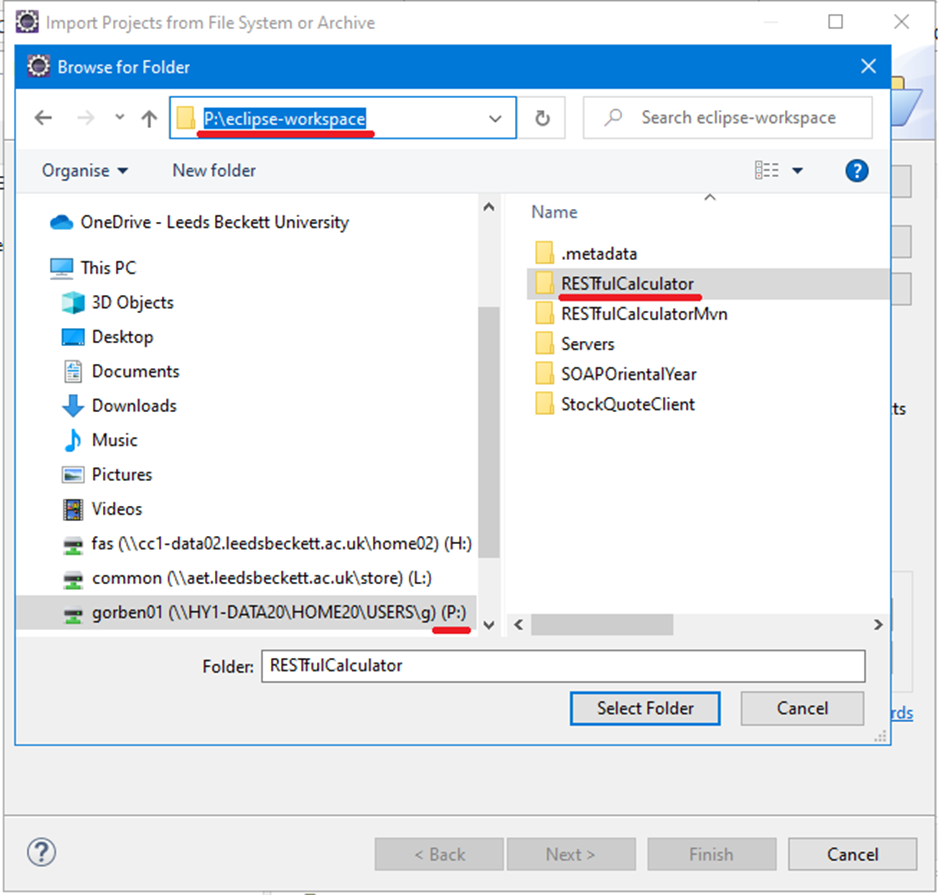
Unzip the file to the following location ‘P:\’.

Make sure the folder structure looks like following (avoid having ‘eclipse-workspace’ inside of ‘eclipse-workspace’):



Switch back to Eclipse and open the project from the file system: ‘File’->’Open Projects from File System…’

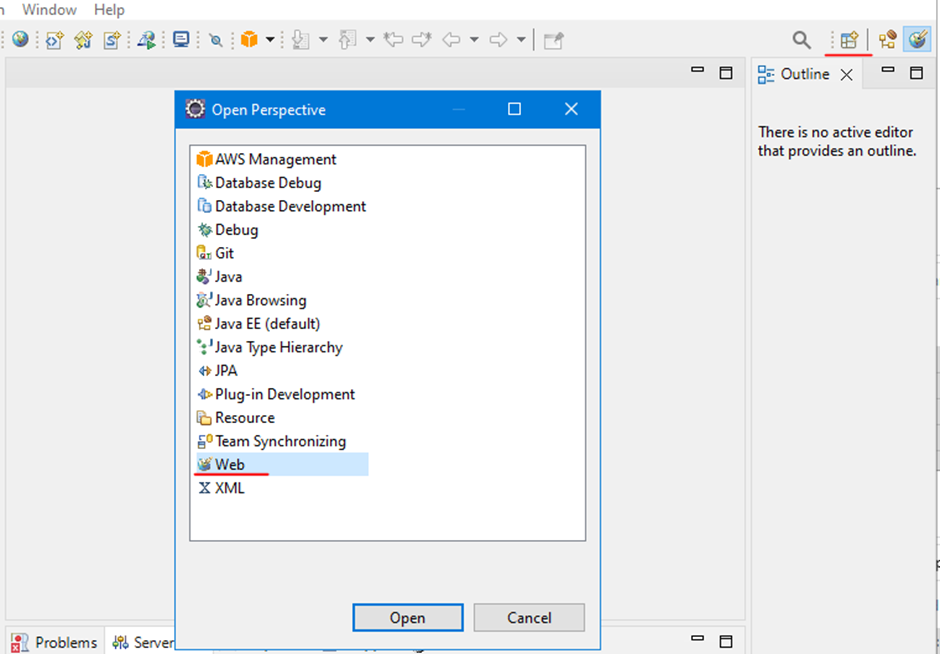




Select the project folder and ‘Finish’.

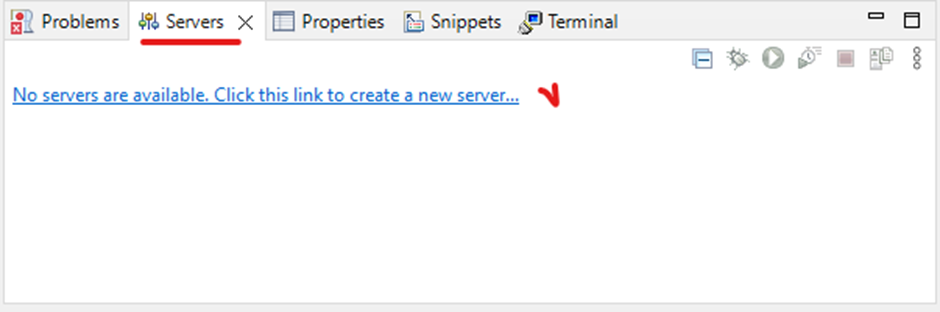
## Configuring the Tomcat Web Server

Select the ‘Web perspective’

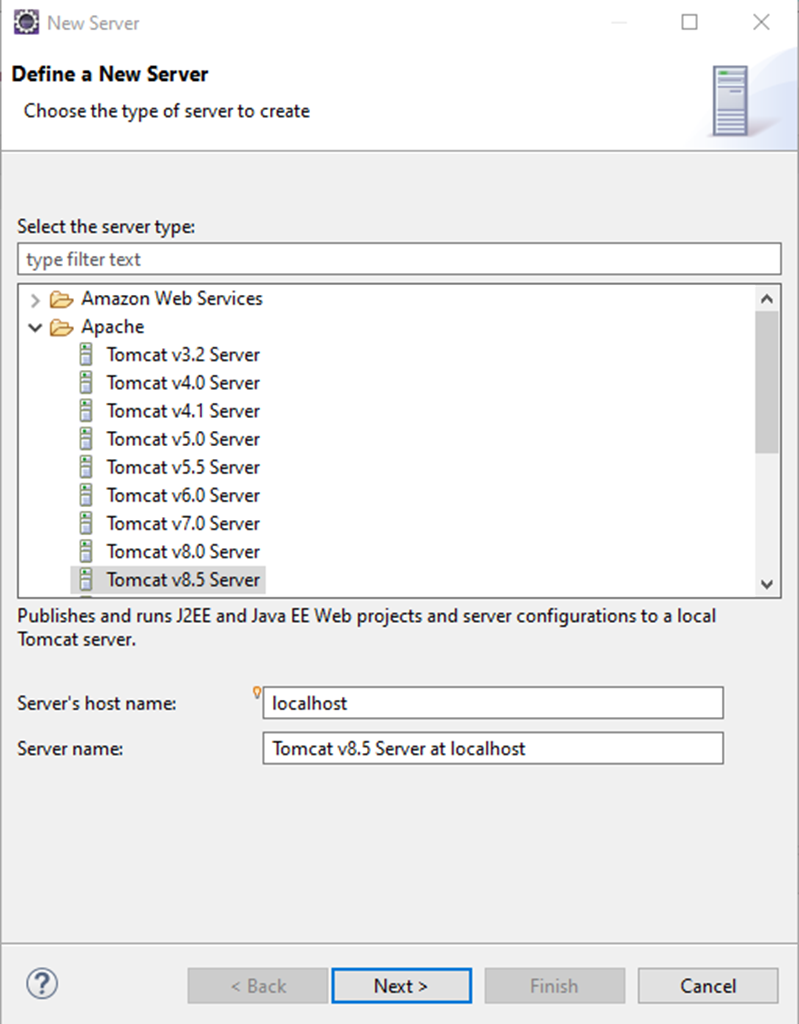


Check if Tomcat Server v8.5 is already added to your workspace.

If yes, skip the below steps. If not, you need to create a new server:



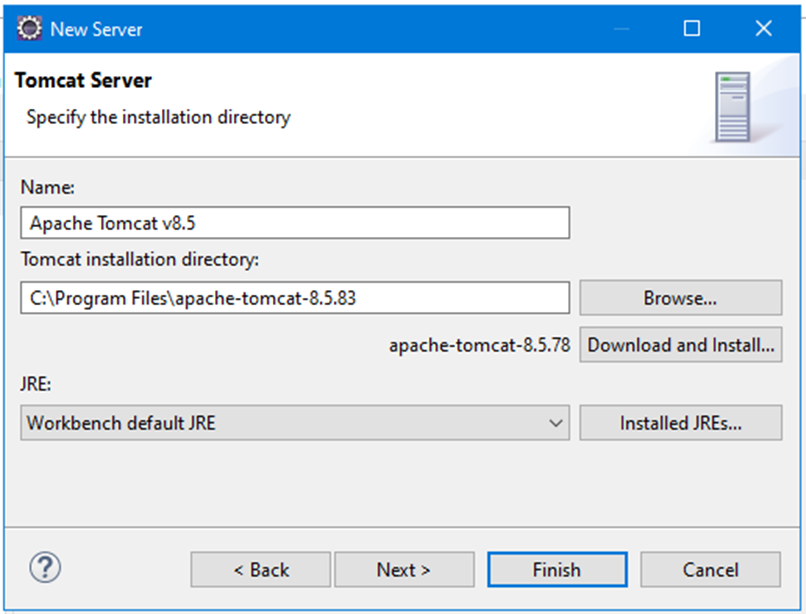
Select ‘Tomcat v8.5 Server’ under ‘Apache’ list of servers.



Click on ‘Next’ and browse to Tomcat installation directory: ‘C:\Program Files\apache-tomcat-8.5.83’



Select ‘apache-tomcat-8.5..83’ folder and finish installation.



# 

# Step 2: Exploring RESTful Java Web Service and Client

**Representational State Transfer (REST)** is an architectural style that specifies constraints, such as the uniform interface, that if applied to a web service induce desirable properties, such as performance, scalability, and modifiability, that enable services to work best on the Web. RESTful services provides a simpler alternative to SOAP and Web Services Description Language (WSDL) based Web services. In the REST architectural style, data and functionality are considered resources and are accessed using **Uniform Resource Identifiers (URIs)**, typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations. The **REST** architectural style constrains an architecture to a client/server architecture and is designed to use a stateless communication protocol, typically **HTTP**. In the REST architecture style, clients and servers exchange representations of resources by using a standardized interface and protocol.

Key RESTful principles:

* **Resource identification through URI**: A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by URIs, which provide a global addressing space for resource and service discovery (see the @Pathannotations in Java code).
* Uniform interface: Resources are manipulated using a fixed set of four create (PUT), read (GET), update (POST) and delete (DELETE) operations (see @PUT, @GET, @POST, and @DELETE annotations).
* Self-descriptive messages: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others (see @Produces and @Consumes annotations).
* Stateful interactions through hyperlinks: Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields.

P.S. **Java annotation** is a form of syntactic metadata that can be added to Java source code.

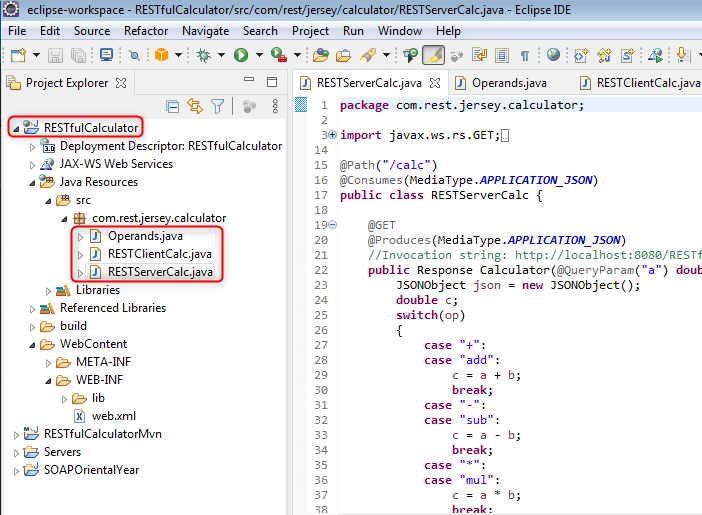
**Self-study:** analyse key differences between **SOAP** and **RESTful** web services.

## 

## Exploring the RESTServerCalc.java

Make sure you use Web Eclipse perspective: 

Expand the **RESTfulCalculator** project in the Project Explorer panel and open the following Java source files: *RESTServerCalc.java*, *RESTclientCalc.java* and *Operands.java*.



**Note:** If you can not see the **RESTfulCalculator** project in the Project Explorer panel,   
Click on **File -> Open Projects from File System …** and specify the path to the project from ‘P:\eclipse-workspace’ folder.

The Java class **RESTServerCalc** implements an arithmetic calculator. It also demonstrates different techniques, which can be used to exchange parameters between a web service and a client. In this example we use **Jersey** implementation of **JAX-RS** specification which is an open source framework for developing RESTFul Web Services (notice imported classes and .jar files in WEB-INF\lib folder which are added into Java build path).

**JAX-RS** (Java API for RESTful Web Services), is a set if APIs to developer REST service.   
**Jersey** is the open source, production quality, JAX-RS Reference Implementation for building RESTful Web services.

***Input parameters transfer.*** When a client invokes a web services, the input parameters can usually be passed using one of the following techniques:

* GET method with the query parameters (see @GET and @QueryParam annotations), e.g. http://localhost:8080/RESTfulCalculator/calc**?a=6&b=7&op=add**
* GET method with the input parameters used as a part of the URI path (see @GET and @Path annotations), e.g. http://localhost:8080/RESTfulCalculator/calc**/sub/5/9**
* A combination of path and query parameters, e.g. [http://localhost:8080/RESTfulCalculator/calc**/add?a=5&b=9**](http://localhost:8080/RESTfulCalculator/calc/add?a=5&b=9)

@GET

@Path("/add")

@Produces(MediaType.APPLICATION\_JSON)

public Response addJSON(@QueryParam("a") double a,

@QueryParam("b") double b) throws JSONException {

JSONObject json = new JSONObject();

json.put("a", a);

json.put("b", b);

json.put("op", "add");

json.put("c", a + b);

String result = ""+json;

return Response.status(200).entity(result).build();

}

* POST method (see @POST annotation; can be also combined with the @Path annotation) with parameters transferred as enclosed in the body of the request message usually as JSON or XML, e.g. [http://localhost:8080/RESTfulCalculator/calc**/mul**](http://localhost:8080/RESTfulCalculator/calc/mul)

@POST

@Path("/mul")

@Consumes(MediaType.APPLICATION\_JSON)

@Produces(MediaType.APPLICATION\_JSON)

// Operands is a Java class used to map JSON input, see below

public Response mulJSON(Operands operands) throws JSONException {

JSONObject json = new JSONObject();

json.put("a", operands.getA());

json.put("b", operands.getB());

json.put("op", "mul");

json.put("c", operands.getA() \* operands.getB());

String result = ""+json;

return Response.status(200).entity(result).build();

}

@XmlRootElement

public class Operands {

public double a;

public double b;

public double getA() {

return a;

}

public double getB() {

return b;

}

public void setA(double a) {

this.a = a;

}

public void setB(double b) {

this.b = b;

}

public Operands() {

}

}

***Type of input consumed.*** A RESTful web service using the POST method can consume input data of different types. The most common types are JSON and XML. It is defined using @Consumes annotation, e.g. @Consumes(MediaType.APPLICATION\_JSON).

***Type of output produced.*** A RESTful web service can produce output of different types. The most common types are JSON and XML. It is defined using @Produces annotation, e.g. @Produces(MediaType.APPLICATION\_JSON) or @Produces(MediaType.TEXT\_XML):

@GET

@Path("/add")

@Produces(MediaType.APPLICATION\_JSON)

public Response addJSON(@QueryParam("a") double a,

@QueryParam("b") double b) throws JSONException {

JSONObject json = new JSONObject();

json.put("a", a);

json.put("b", b);

json.put("op", "add");

json.put("c", a + b);

String result = ""+json;

return Response.status(200).entity(result).build();

}

@GET

@Path("/sub/{a}/{b}")

@Produces(MediaType.TEXT\_XML)

public String sub(@PathParam("a") double a, @PathParam("b") double b) {

return "<?xml version=\"1.0\"?>\n" +

"<calc>\n" +

" <operands>\n" +

" <a>" + a + "</a>\n" +

" <b>" + b + "</b>\n" +

" </operands>\n" +

" <operation>sub</operation>\n" +

" <result>\n" +

" <c>" + (a - b) + "</c>\n" +

" </result>\n" +

"</calc>";

}

When a web service returns a result together with the status of the operation, e.g.

return Response.status(200).entity(result).build();

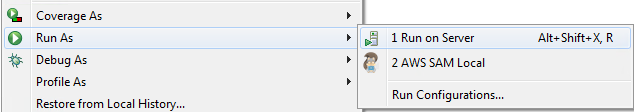
or

return Response.status(400).entity("400 Arithmetic exception: division by zero error").build();

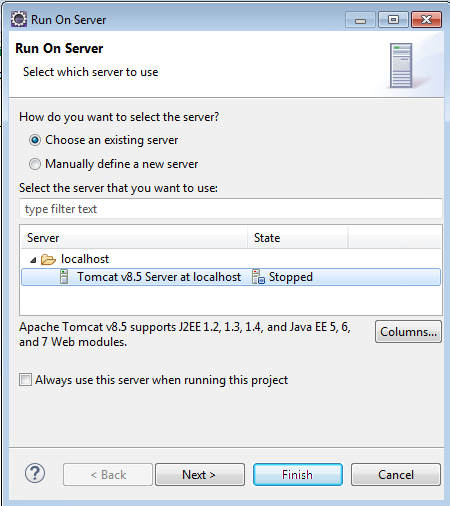
## 

## Launching and testing the RESTful Web Service

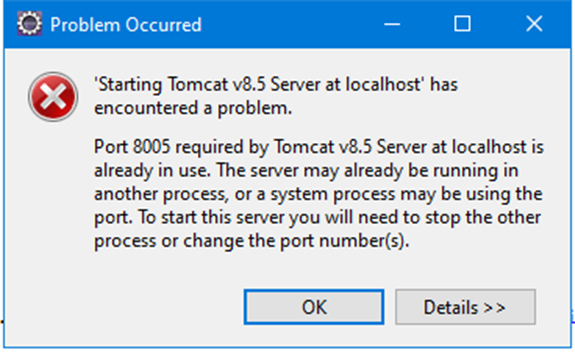
1. Right click on RESTfulCalculator project and select Run As -> Run on Server



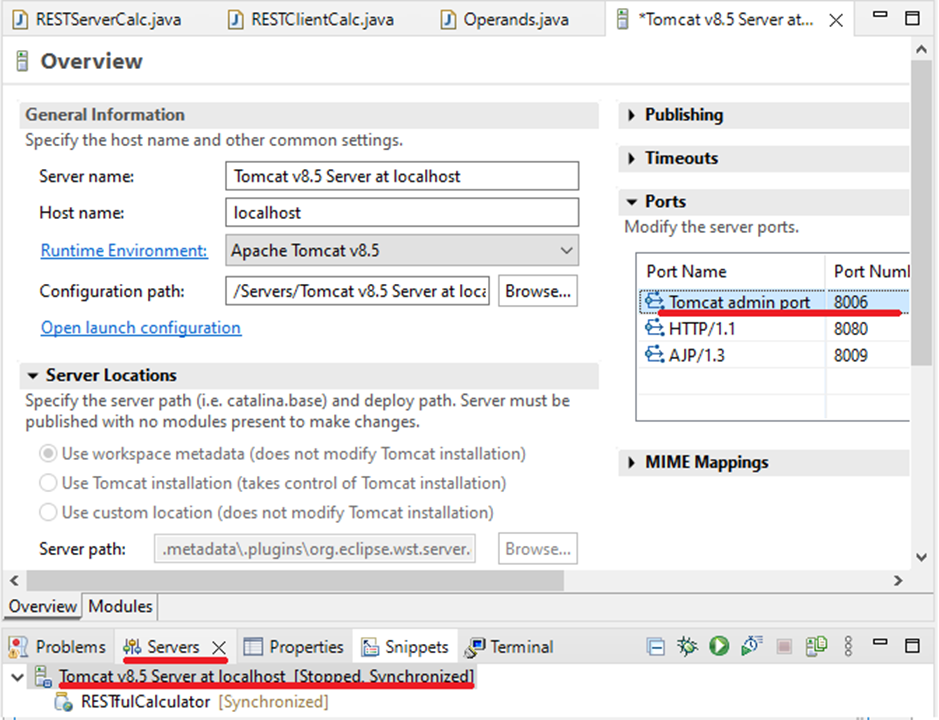
1. Run the project by clicking on Finish



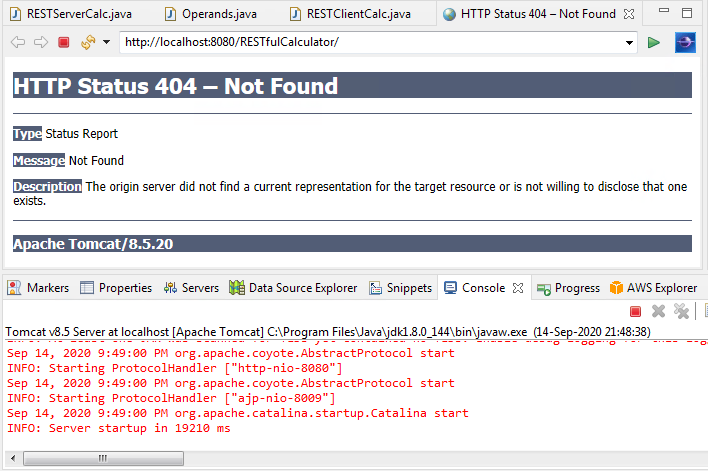
If you encounter the following error message, then you need to modify TCP ports used by Tomcat to start.



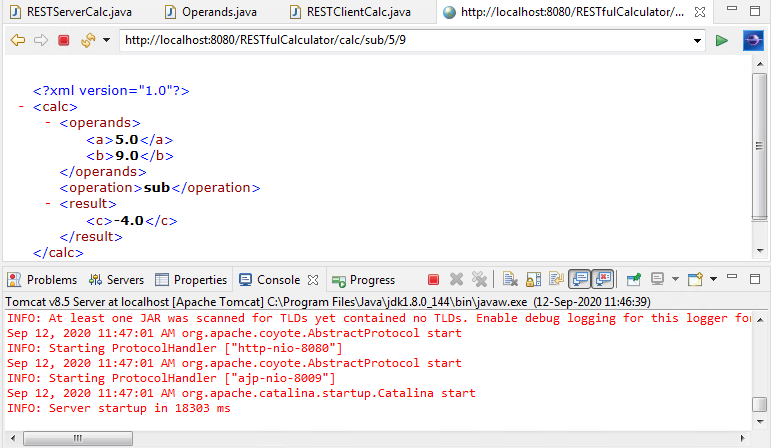
To do this, double click on Tomcat 8.5 in the ‘Servers’ panel and change **8005** port to another number, e.g. **8006**.



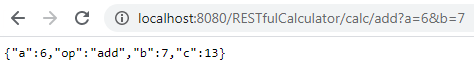
1. Wait for Apache Tomcat to start. In a minute or so you will see a new Explorer window displaying the following:



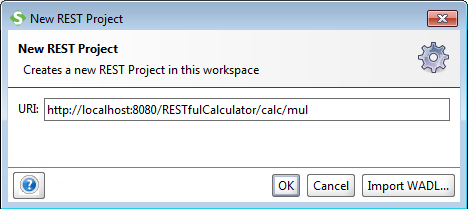
1. In the opened browser window modify the URL address to invoke different operations of the Calculator web service, e.g.: <http://localhost:8080/RESTfulCalculator/calc/sub/5/9> and check the XML output:



1. Open Google Chrome and test other invocation methods returning JSON:
   1. <http://localhost:8080/RESTfulCalculator/calc?a=6&b=7&op=add>
   2. <http://localhost:8080/RESTfulCalculator/calc/add?a=6&b=7>



1. To test the POST request method run SoapUI on your VM and create a new REST Project:



Choose the POST method, enter input JSON string {"a":5, "b":7}, execute the request and check the JSON output:



1. Use SoapUI to test GET methods supporting path and query parameter transfer

## Run a Web Service Java Client

1. Open Explore RESTClientCalc.java



1. Notice imported classes.
2. Explore different ways of invoking a web service and passing input/output parameters
   1. Using GET method and URL QUERY parameters to invoke a RESTful service returning JSON output

private static final String webServiceURI = "http://localhost:8080/RESTfulCalculator";

ClientConfig clientConfig = new ClientConfig();

Client client = ClientBuilder.newClient(clientConfig);

URI serviceURI = UriBuilder.fromUri(webServiceURI).build();

WebTarget webTarget = client.target(serviceURI);

System.out.println(webTarget.path("calc").queryParam("a", "74")

.queryParam("b", "12").queryParam("op", "+").request()

.accept(MediaType.APPLICATION\_JSON).get(String.class));

* 1. Using GET method and URL PATH parameters to invoke a RESTful service returning XML output

System.out.println(webTarget.path("calc").path("sub")

.path("67").path("45").request().accept(MediaType.TEXT\_XML)

.get(String.class));

* 1. Using GET method combining both URL PATH and QUERY parameters

System.out.println(webTarget.path("calc").path("add")

.queryParam("a", "56").queryParam("b", "34").request() .accept(MediaType.APPLICATION\_JSON).get(String.class));

* 1. Using POST with Raw JSON parameters

double a = 8.0;

double b = 6.0;

String request = String.format("{\"a\":%s,\"b\":%s}", a, b);

System.out.println(webTarget.path("calc").path("div").request()

.post(Entity.entity(request, MediaType.APPLICATION\_JSON),

String.class));

* 1. Using POST with Object2JSON parameters

Operands operands = new Operands();

operands.setA(6.0);

operands.setB(7.0);

System.out.println(webTarget.path("calc").path("mul")

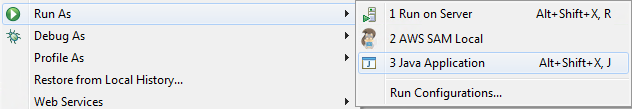
.request(MediaType.APPLICATION\_JSON)

.accept(MediaType.APPLICATION\_JSON)

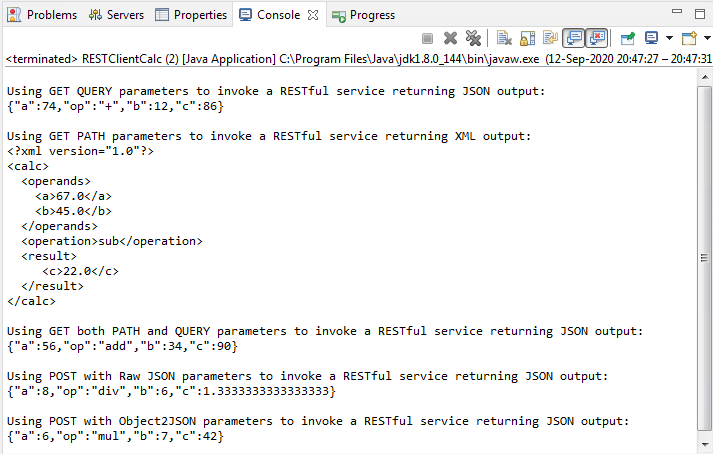
.post(Entity.json(operands), String.class));

1. Execute the RESTClientCalc.java by right clicking on the file in the Project Explorer and selecting Run As -> Java Application

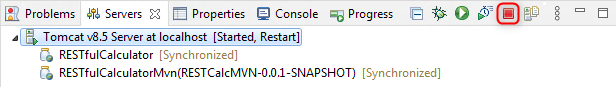
Note: make sure RESTServerCalc.java is still running on a web server.



1. Explore the Console output:



1. Stop the Apache Tomcat web server in the Servers tab



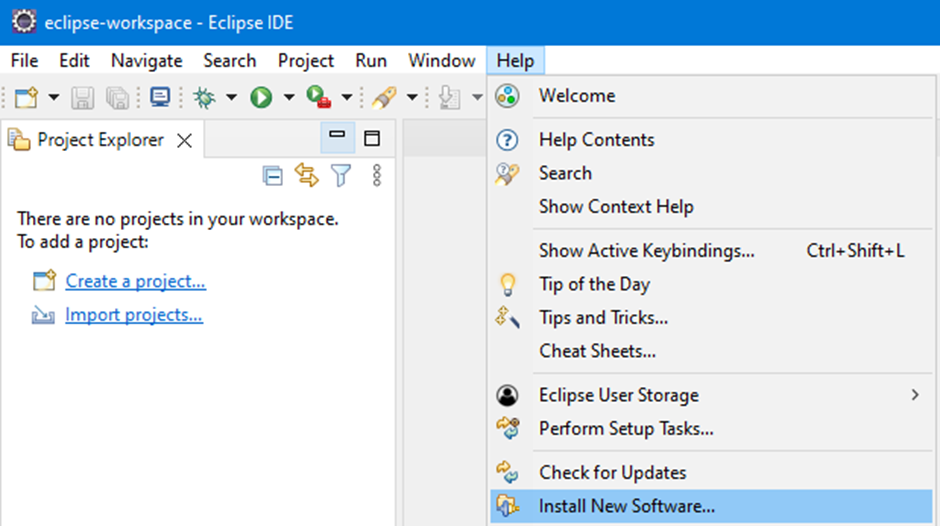
# 

# Step 3: Deploying a Web Service to AWS Elastic Beanstalk PaaS

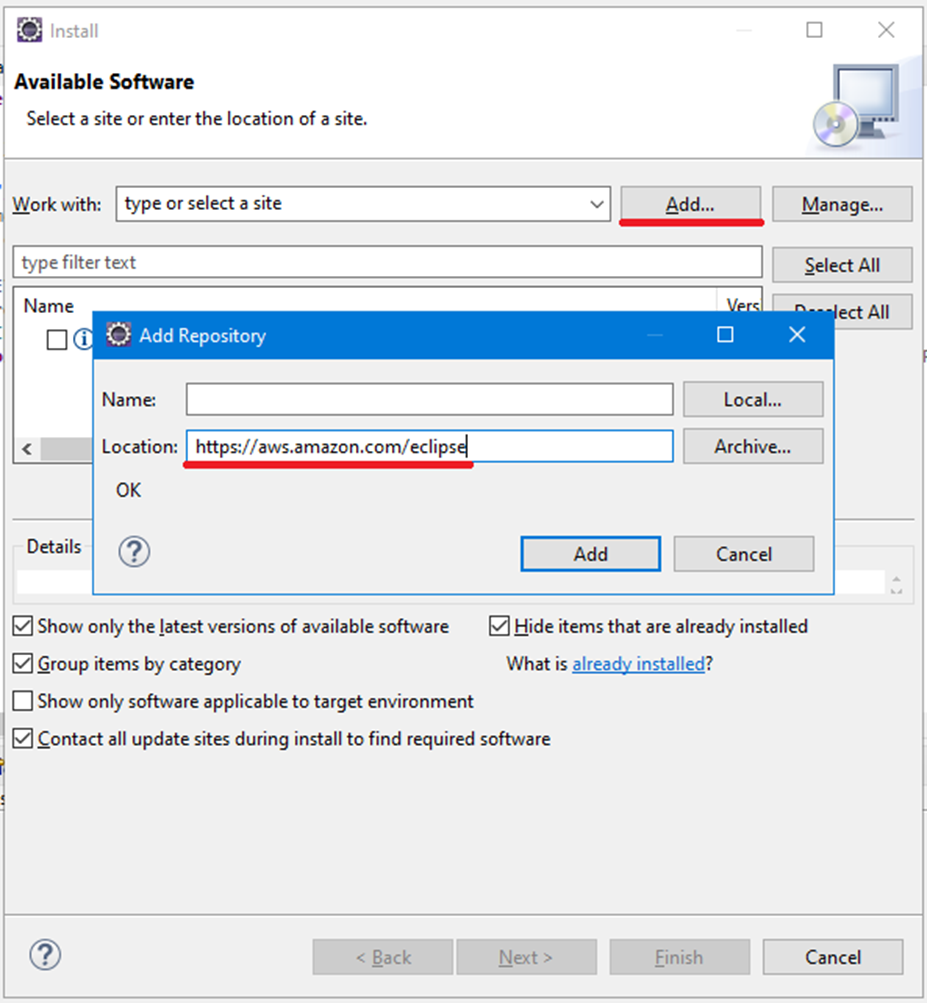
## 

## Installing AWS Plugin for Eclipse

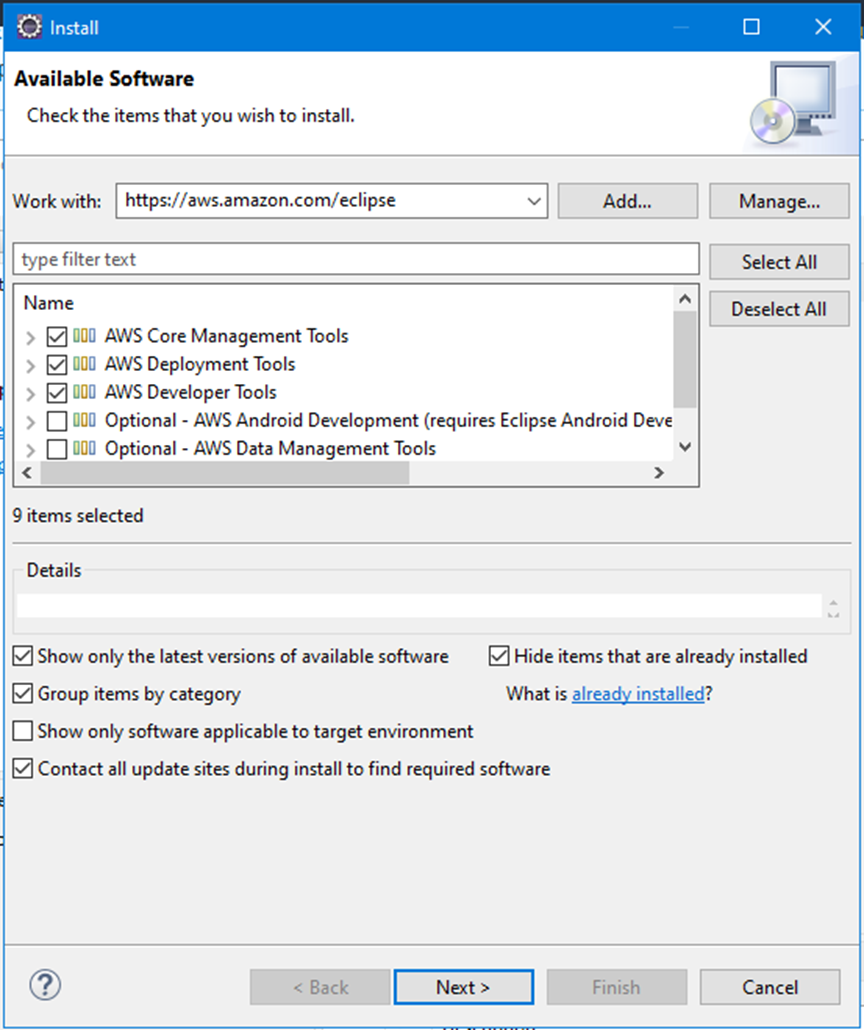
Open ‘Help’ -> ‘Install New Software…’

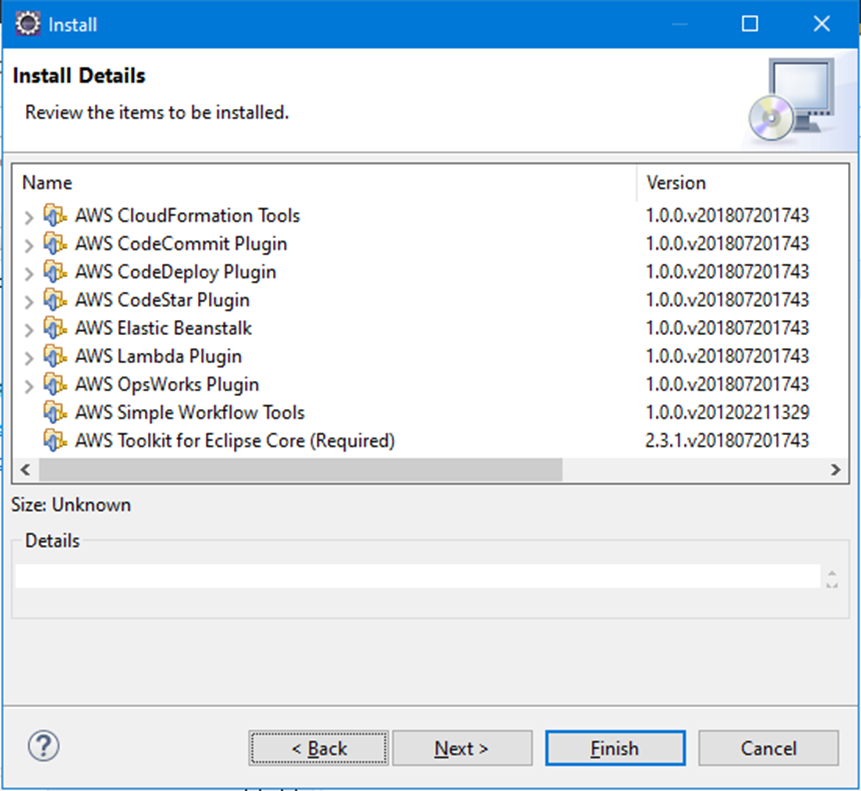


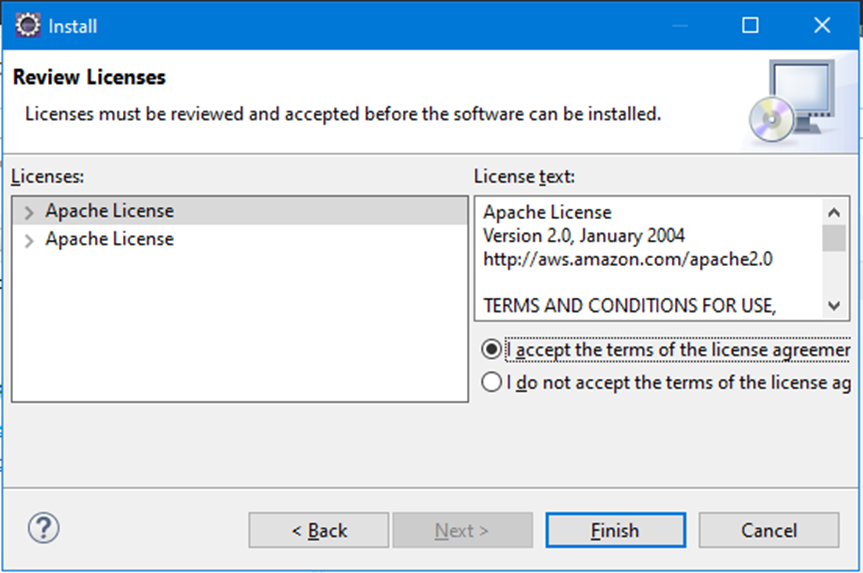
Add new repository:<https://aws.amazon.com/eclipse>

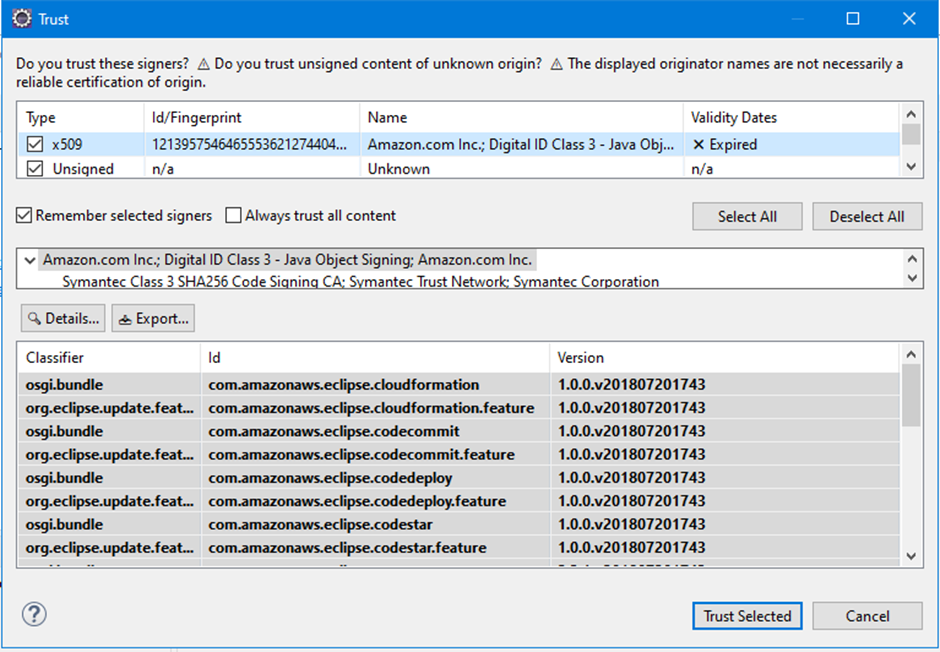


Select the following AWS plugins:

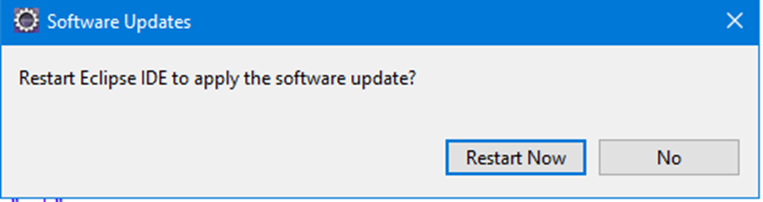






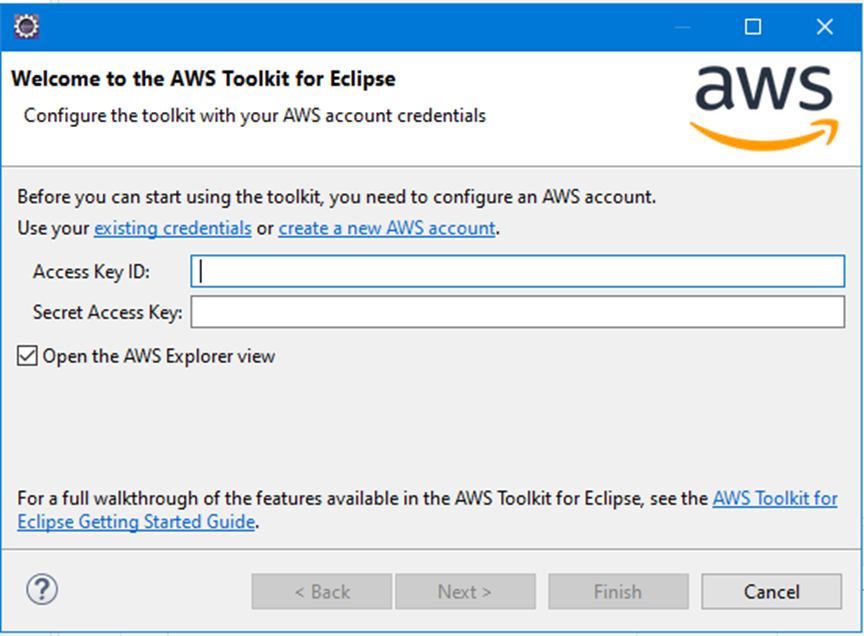


Restart Eclipse if prompted.



After you run Eclipse next time you will need to provide AWS credentials to access AWS services.

The following window should pop-up automatically after you restart Eclipse:

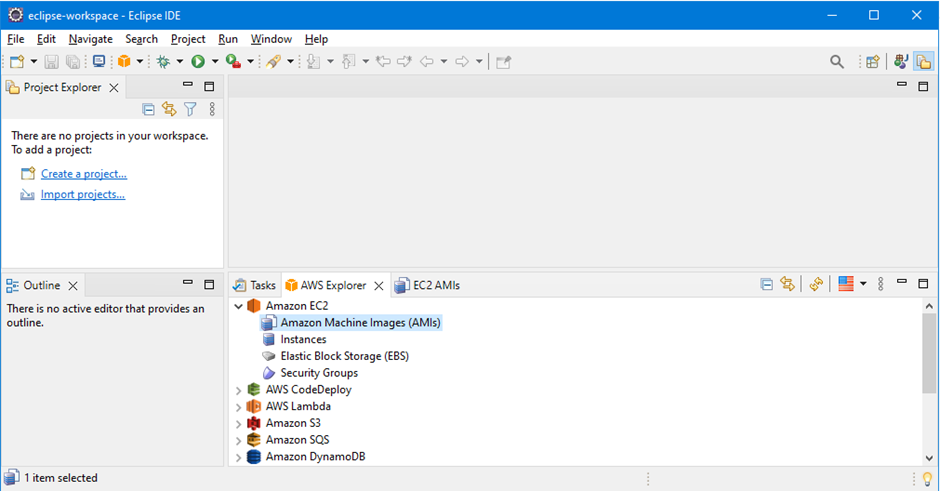


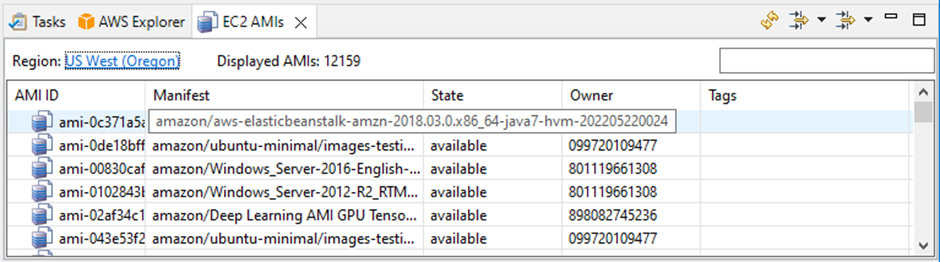


Select ‘AWS Management’ perspective:



Check that you can browse Amazon AWS Services by clicking on ‘Amazon EC2’->’Amazon Machine Images (AMIs)’. If your credentials are accepted, you should see a list of AMIs available for you in the specified AWS Region:





## 

## Re-running Eclipse with the Jaxb library

Download jaxb-api-2.2.7.jar file from the module web page.

Close Eclipse IDE.

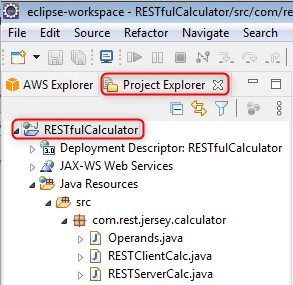
Run Eclipse IDE from the command line providing a reference to the downloaded jar file. e.g.:

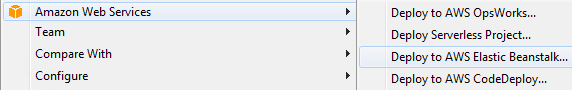
C:\eclipse\eclipse.exe -dev C:\Users\c123456\Downloads\jaxb-api-2.2.7.jar

**Note:** The current version of the AWS plugin for Eclipse does not include the jaxb library. Thus, to enable AWS plugin deploying your Java application to AWS you need to to manually refer to this library when running Eclipse. If you do not plan to deploy your application to AWS, you can run Eclipse using the Start menu.

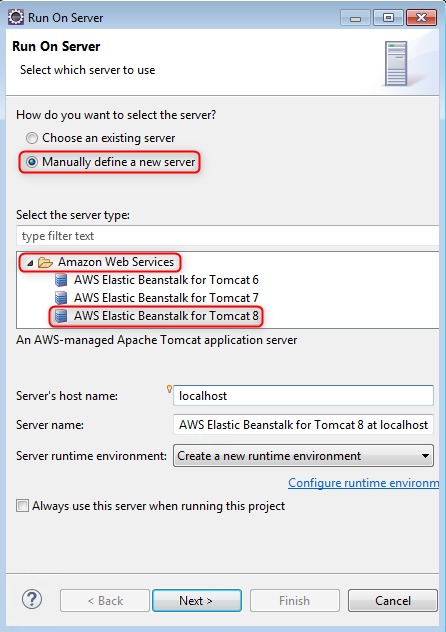
## Launching a Web Service on the AWS Elastic Beanstalk PaaS

1. Select the Project Explorer tab, Right Click on the **RESTfulCalculator** project and select Amazon Web Services -> Deploy to AWS Elastic Beanstalk …

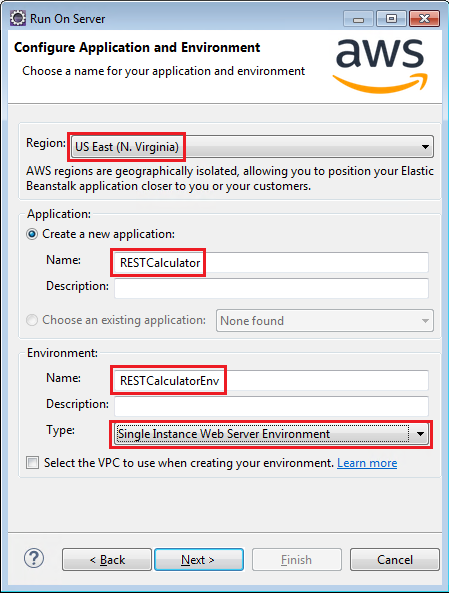




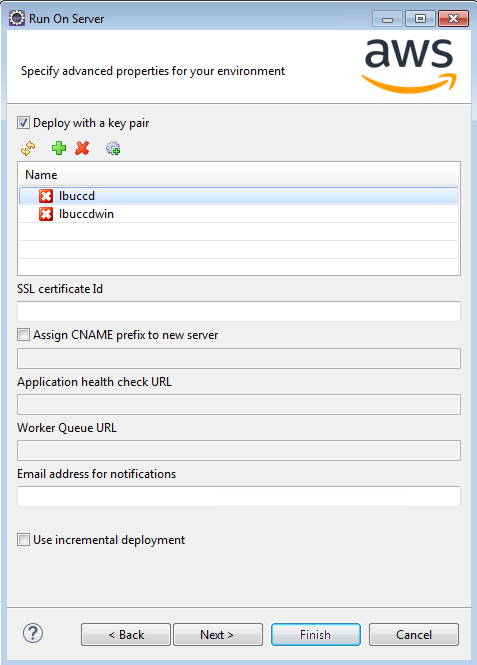
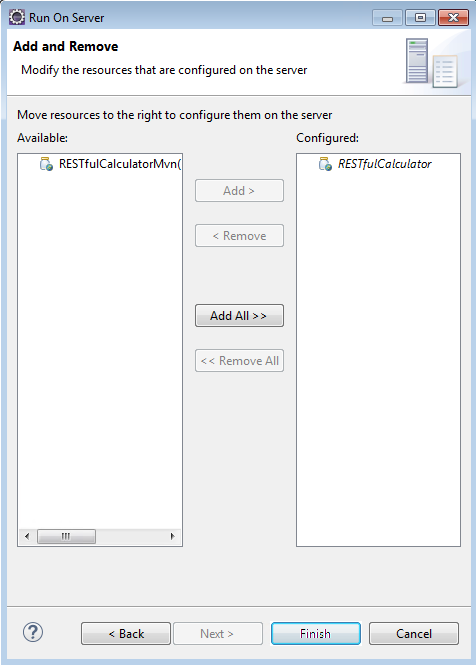
1. Now you will be guided through the AWS Elastic Beanstalk deployment process
   1. Choose AWS Elastic Beanstalk for Tomcat8 as a new server and click on Next



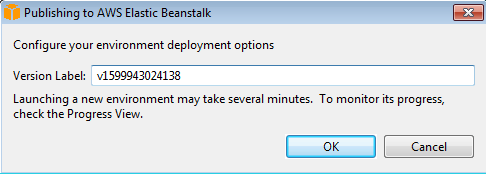
* 1. Make sure your deploy your application in the same AWS region you selected when configured your default account
  2. Specify a name of your application and a name of an environment (a virtual instance) it will be deployed on; choose the Single Instance Web Server Environment; click on Next

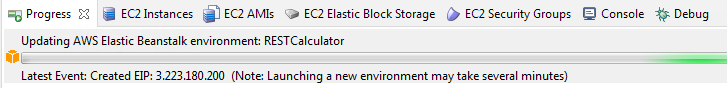


* 1. Choose a key pair (or create a new one) you can use later to get a remote shell access to a virtual instance your web service will be deployed on and click on Next and Finish

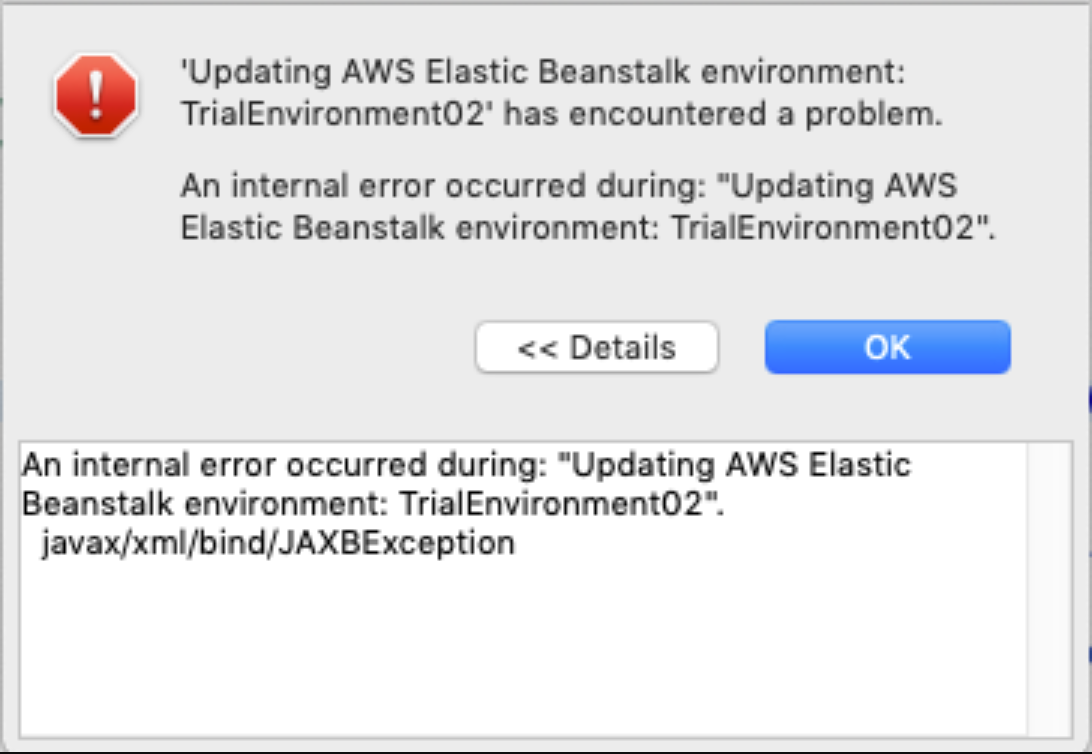
 

* 1. Agree with the version label and wait for AWS Elastic Beanstalk environment to start



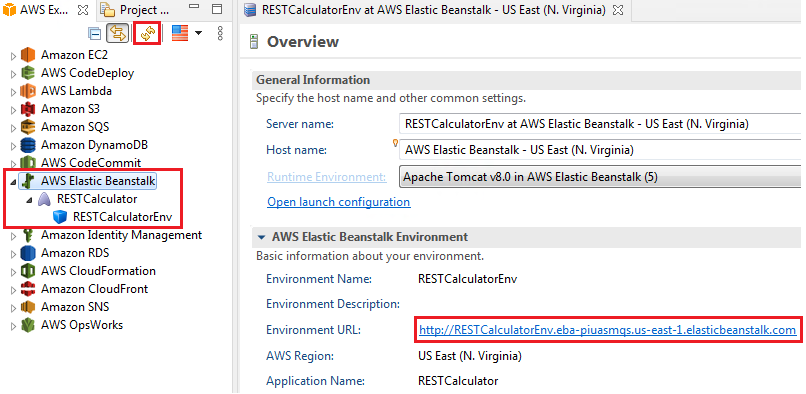


Deployment may take several minutes

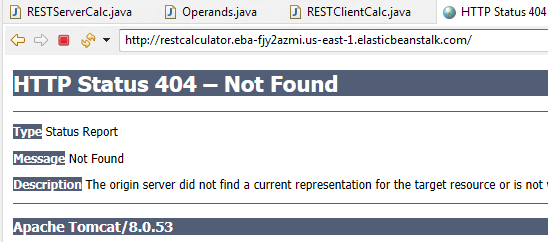


If during deployment you encounter an exception message “***An internal error occurred during: "Updating AWS Elastic Beanstalk environment: RESTfulCalculatorEnv". javax/xml/bind/JAXBException***”, then you need to close Eclipse and re-run it again from the command line with the Jaxb library as discussed above.

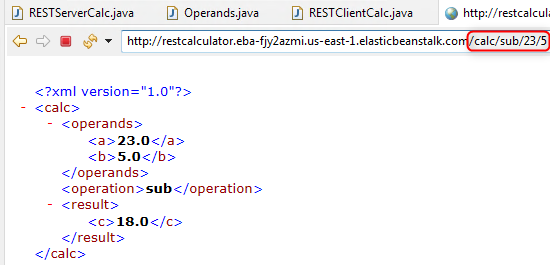
* 1. Check periodically the status of your environment and application using AWS Management console; notice the URL of your web service



* 1. Copy the URL and open in in a web browser



* 1. Modify the URL string to invoke the RESTful Calculator web service



## Test the RESTful Java Web Service deployed at AWS Elastic Beanstalk

You can further use SoapUI and RESTClientCalc.java to test your RESTful Java Web Service deployed at AWS Elastic Beanstalk.

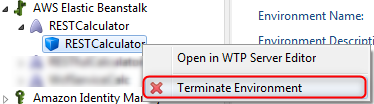
**Note:**

1. Do not forget to provide a correct URL pointing to your RESTful Java Web Service deployed at AWS Elastic Beanstalk instead of localhost:8080 used to test its local deployment

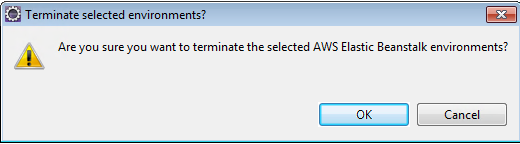


# Step 4: Deleting AWS Elastic Beanstalk resources after use

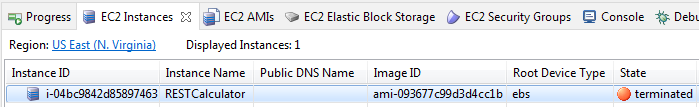
1. Firstly, terminate the environment your web service is running on



1. Confirm your choice



1. Wait until the status of the EC2 Instance changes to terminated



1. Now you can delete your web service from AWS Beanstalk



1. Confirm your choice

