User Manual: MSE Project

November 5, 2010

Prepared by Doug Smith
Version 0.1

Table of Contents

Revision History

Version	Date	Changes
0.1	11/24/2010	First draft.

Introduction

This purpose of this manual is to document how to build, install, and configure the system components associated with the system, and to document the public web service interfaces offered by the system.

The audience for this manual are programmers and system administrators. Note the system documented in this manual was an architectural proof of concept, and does not have a user interface per se. Thus the orientation of this manual is more from a system installation and administration perspective than from an end user perspective.

Building the Software

There are two software components that must be built: the main software package, and a convenience package provided to make deploying and working with the database easier.

To build the software, the following items are required:

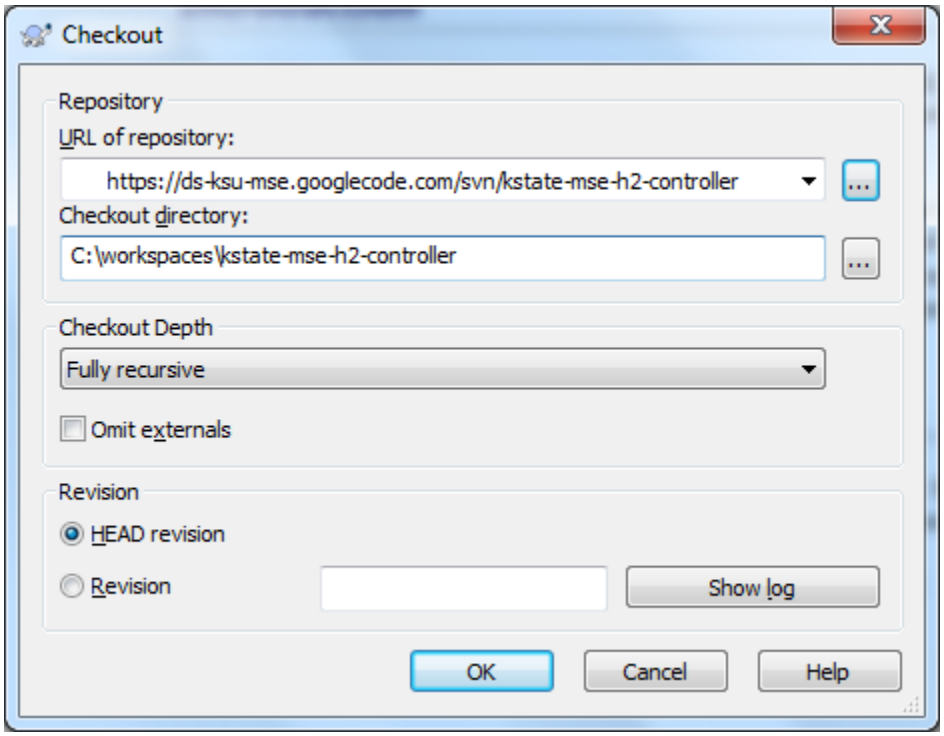
- A Java development kit, version 1.6.x
- A subversion source control client
- Maven 2

Building the H2Controller Web Application

The H2Controller web application provides a way to deploy the H2Database inside a war, along with a servlet to allow starting and stopping the database instance via the web.

To build the application:

1. Download the source code from https://ds-ksu-mse.googlecode.com/svn/kstate-mse-h2-controller using a subversion client.



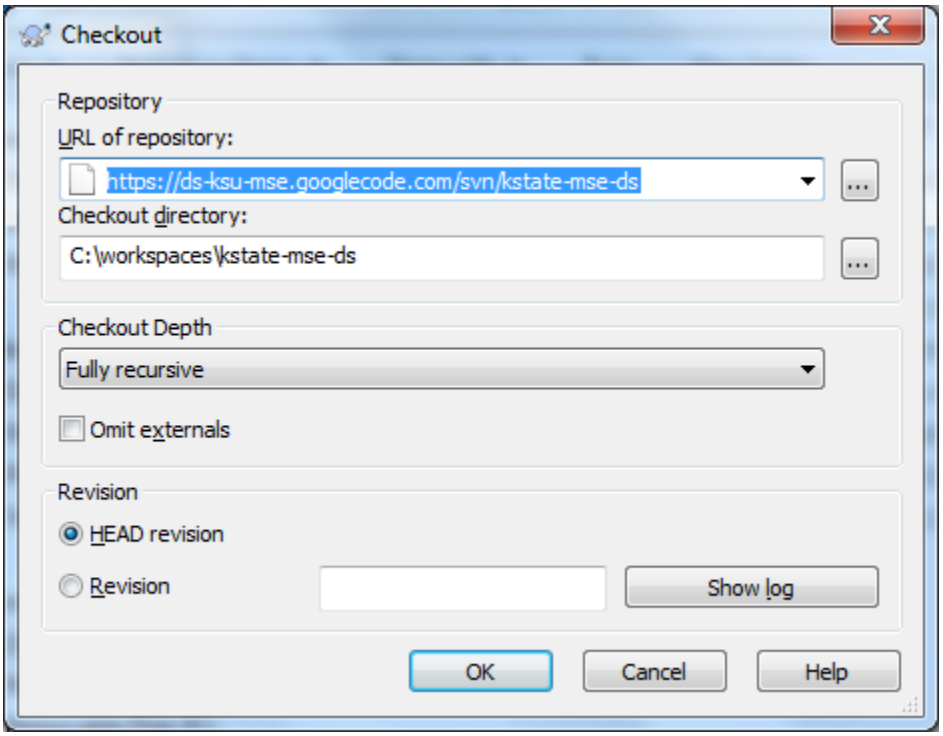
2. Open a command prompt in the directory the software was downloaded to, and build as follows. Note we are skipping running the unit tests as part of the build to avoid having to start a database server and install the schema just to build the software).

   mvn -Dmaven.test.skip=true clean package

3. When the build is completed, the war produced by the build will be located in the target directory created as part of the build (kstate-mse-h2-controller.war)

Building the kstate-mse-ds Web Application

1. Download the source code using a subversion client from https://ds-ksu-mse.googlecode.com/svn/kstate-mse-ds



2.Open a command prompt in the directory the software was downloaded to, and build as follows. Note we are skipping running the unit tests as part of the build to avoid having to start a database server and install the schema just to build the software).

mvn -Dmaven.test.skip=true clean package

3. When the build completes, the war produced by the build will be in the target directory created during the build (kstate-mse-ds.war).

Preparing the Server Image

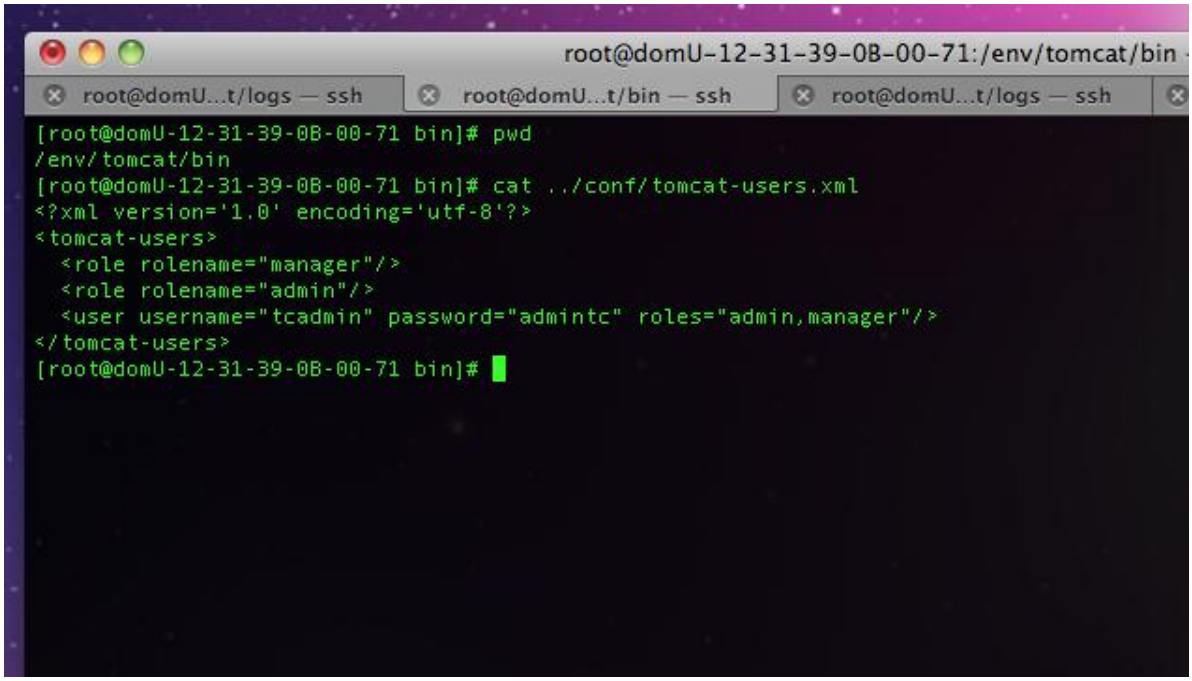
This sections documents how to prepare a suitable Amazon Elastic Cloud Computing (EC2) image needed to run the software. This is by no means meant to be a tutorial on how the use EC2. Fortunately, there is a wealth of information on how to use EC2 available on the Amazon web site: aws.amazon.com

The image used for my project was based on ami-11ca2d78, which is the default image used by the EC2 eclipse plugin. This image is a Fedora Linux image that includes Java 1.6 and Tomcat 6. Creating the image involved making some configuration tweaks to the base image, then saving the image such that I could have an image ready to start on demand with my changes included.

The changes made to the baseline are pretty straightforward.

- The .bashrc_profile file for root needs the following added:
  export JAVA_HOME=/env/jdk
  export CATALINA_OPTS="-Xms512m –Xmx512m"

- The /env/tomcat/conf/tomcat-users.xml file needs to have a user set up as follows:



One these changes have been made to the image, a custom image containing the changes can be created as follows:

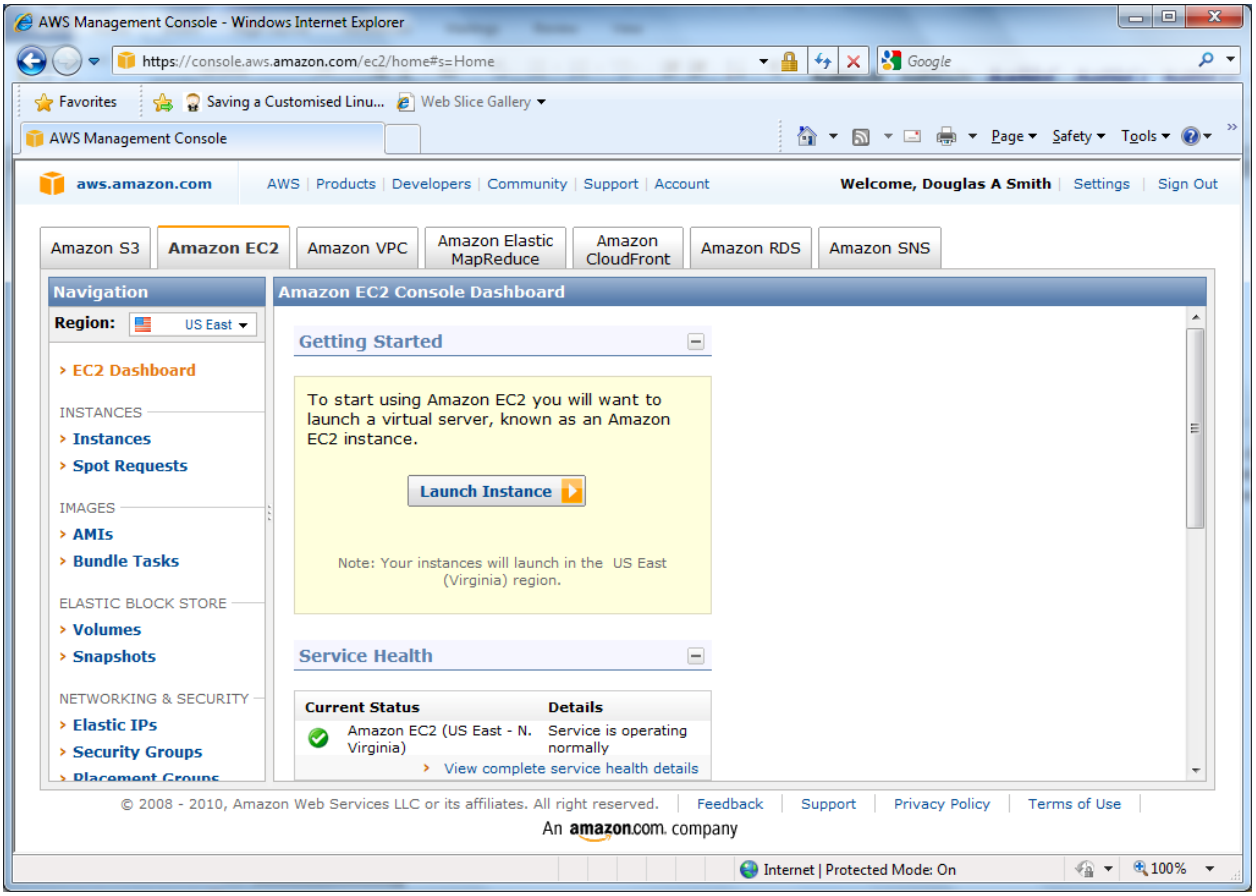
1. Copy your private key and certificate file to the /mnt directory. The keys are needed for the process of creating the instance, and they are placed in the /mnt directory to ensure they are not saved in the image that is created.
2. Create the AMI using the EC2 (this assumes the AMI tools and APIs have been installed): ec2-bundle-vol -d /mnt -k /mnt/pk-P4GHTRP23SBCOO5KMZAX66WKMX2N6C57.pem -c /mnt/cert-P4GHTRP23SBCOO5KMZAX66WKMX2N6C57.pem -u amazon-account-number
3. Upload the image files into S3 storage: ec2-upload-bundle -b kstate-mse-ds-bucket -m /mnt/image.manifest.xml -a amazon-access-key -s my-secret-key

4.Finally, register the instance: ec2-register kstate-mse-ds-bucket/image.manifest.xml

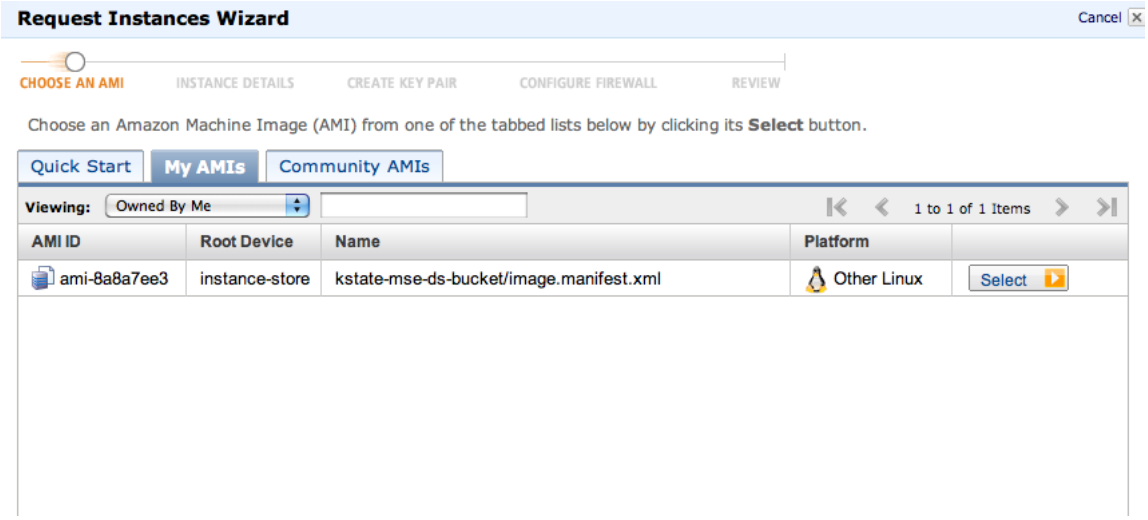
Starting Machine Images

Once the image has been created, it can be started from the AWS Console. In general, note that Amazon provides a toolkit and API to allow the scripting of everything shown in the document; adoption of EC2 in a real project would involve automation of the steps shown in this document.

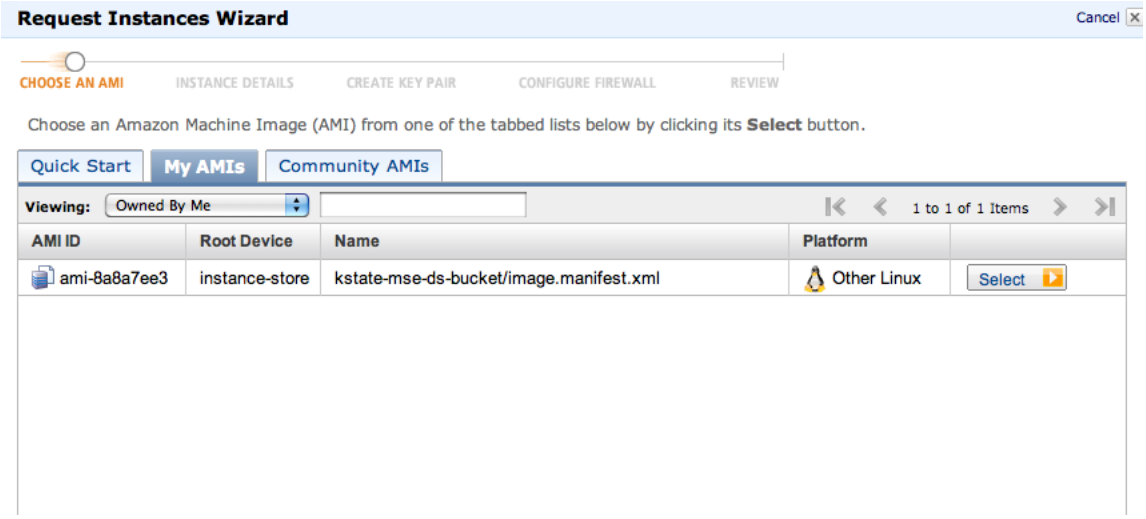
EC2 images can be started from the main EC2 console dashboard:



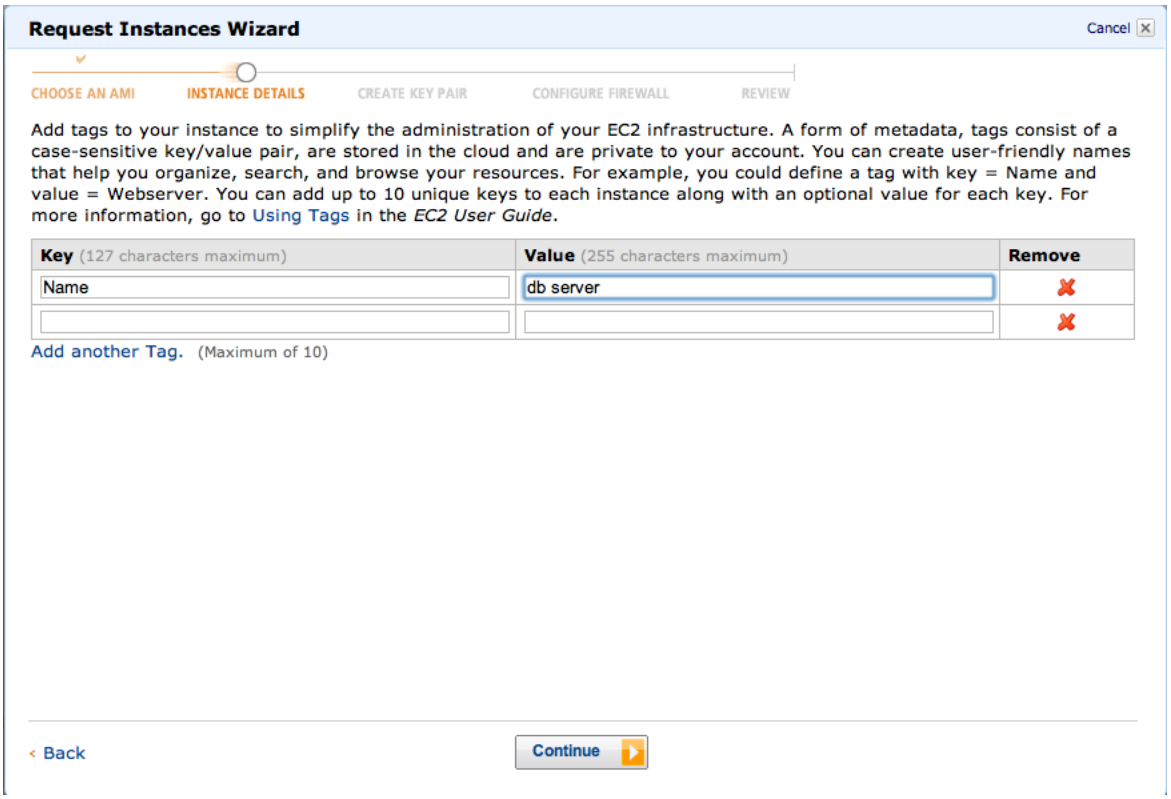
Select the image prepared for the project under the 'My Images' tab:



Next, select the availability zone and the instance size. Note that when booting multiple servers that will be load balanced, it is desirable to spread them among multiple availability zones to guard against an outage at the zone level taking out the entire application. Server instances should also be allocated evenly across all the availability zones used as the load balancer distributes load across zones first, then servers within a zone.



Next accept the defaults in the Advanced Options page, and continue. On the next screen, give the instance a tag to help sort out what it is being used for (very useful when running multiple servers).



Next, select the key pair representing the keys used for security credentials when accessing the image:

Next, select the security group configuration.

Continue, review the options on the next screen, then launch the instance if everything looks correct.

The security group specified above essentially represents firewall rules for the instance, controlling access to the instance via different protocols and port settings. There are some important things to note:

- Ports need to be opened to allow access to the H2 database using different protocols. This means 8082 and 9082 are opened.
- The port used for Hazelcast intercluster communication must be opened (I used port 12000).
- Ports for SSH and HTTP are also needed.

The following screen shot shows the security group configuration used for this project:

Security Groups

Create Security Group | Delete | Show/Hide | Refresh | Help

Viewing: All Security Groups — 1 to 2 of 2 Items

	Name	Description
☐	default	default group
☑	GSG_SecurityGroup	SSH and HTTP

Allowed connections:

Connection Method	Protocol	From Port	To Port	Source (IP or group)	Actions
-	tcp	12000	12000	0.0.0.0/0	Remove
SSH	tcp	22	22	0.0.0.0/0	Remove
SSH	tcp	22	22	213.208.100.0/24	Remove
SSH	tcp	22	22	217.41.224.0/20	Remove
SSH	tcp	22	22	62.50.192.0/21	Remove
SSH	tcp	22	22	67.182.192.0/18	Remove
SSH	tcp	22	22	67.182.211.101/32	Remove
-	tcp	23000	23000	0.0.0.0/0	Remove
HTTPS	tcp	443	443	0.0.0.0/0	Remove
HTTPS	tcp	443	443	213.208.100.0/24	Remove
HTTPS	tcp	443	443	217.41.224.0/20	Remove
HTTPS	tcp	443	443	62.50.192.0/21	Remove
HTTPS	tcp	443	443	67.182.192.0/18	Remove
HTTPS	tcp	443	443	67.182.211.101/32	Remove
HTTP	tcp	80	80	0.0.0.0/0	Remove
-	tcp	8080	8080	0.0.0.0/0	Remove
-	tcp	8082	8082	0.0.0.0/0	Remove
-	tcp	9092	9092	0.0.0.0/0	Remove

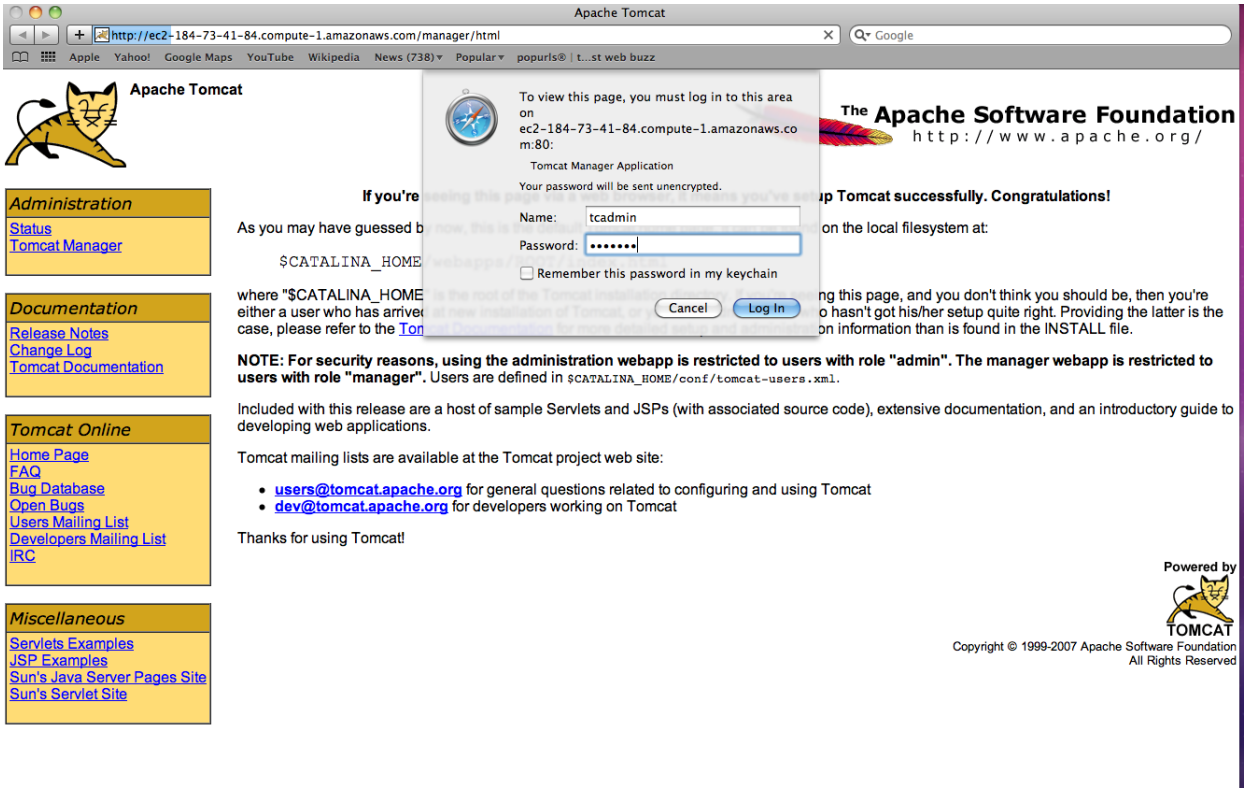
Deploying the Database Controller App

Once an image has been started, applications can be deployed to it and run. This section covers deploying the database controller application.

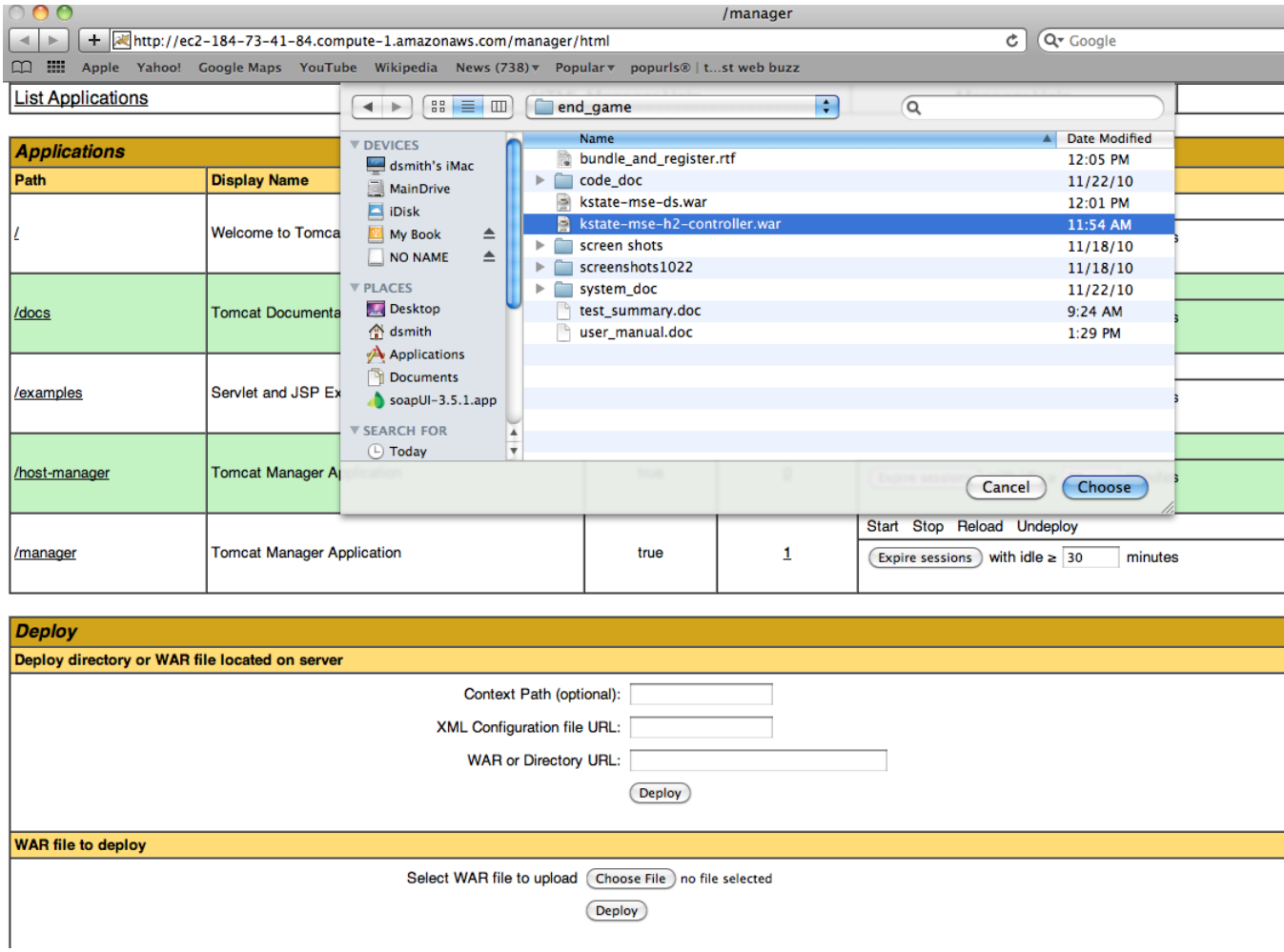
Before an application can be deployed, tomcat must be started. As the current state of the image does not automatically start tomcat, after starting the image, log in and start tomcat:

```
root@ip-10-204-73-113:/env/tomcat/bin — ssh — 104×31
bash    java    wrapper    root@ip-1...bin — ssh

bash-3.2$ ssh -i KSUKeyPair.pem root@ec2-184-73-41-84.compute-1.amazonaws.com
The authenticity of host 'ec2-184-73-41-84.compute-1.amazonaws.com (184.73.41.84)' can't be established.
RSA key fingerprint is 90:e9:af:8f:c0:d9:44:0a:49:c4:b1:da:5c:98:e1:31.
Are you sure you want to continue connecting (yes/no)? yes
Warning: Permanently added 'ec2-184-73-41-84.compute-1.amazonaws.com,184.73.41.84' (RSA) to the list of
known hosts.

         __|  __|_  )  Fedora 8
         _|  (     /    32-bit
        ___|\___|___|

 Welcome to an EC2 Public Image
                        :-)
    Base

[root@ip-10-204-73-113 ~]# cd /env/tomcat/bin
[root@ip-10-204-73-113 bin]# ls
bootstrap.jar       digest.bat        shutdown.bat          tomcat6.exe
catalina-tasks.xml  digest.sh         shutdown.sh           tomcat6w.exe
catalina.bat        jsvc.tar.gz       startup.bat           tool-wrapper.bat
catalina.sh         service.bat       startup.sh            tool-wrapper.sh
commons-daemon.jar  setclasspath.bat  tomcat-juli.jar       version.bat
cpappend.bat        setclasspath.sh   tomcat-native.tar.gz  version.sh
[root@ip-10-204-73-113 bin]# ./startup.sh
Using CATALINA_BASE:   /env/tomcat
Using CATALINA_HOME:   /env/tomcat
Using CATALINA_TMPDIR: /env/tomcat/temp
Using JRE_HOME:       /env/jdk
[root@ip-10-204-73-113 bin]#
```

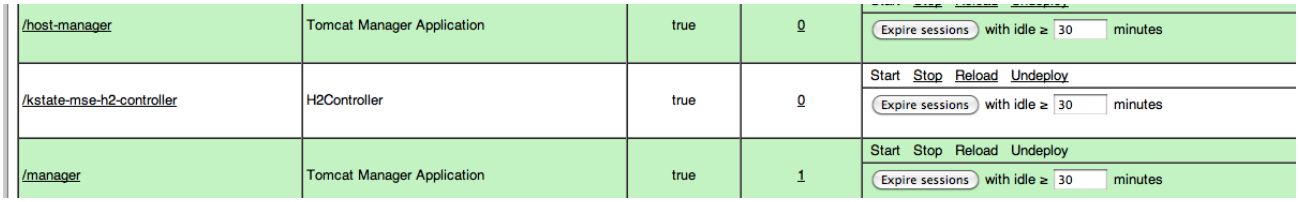
Once tomcat is started, use a browser to connect to the tomcat management application, and log in with the user name and password set when configuring the image:



After logging into the console, scroll down to the Deploy section, and select the war to deploy from the file system:



After selecting the war file, press the Deploy button. When the deployment is finished, the status page is updated to include the application that was just deployed:



Starting the Database and Installing the Schema

Once the database controller application has been deployed and started, the H2 database can be started using the controller application, then the H2 console can be used to create the schema.

The controller application is accessed via the a URL that embeds the command:

http://<ec2 public address>/kstate-mse-h2-controller/ctl?cmd=start

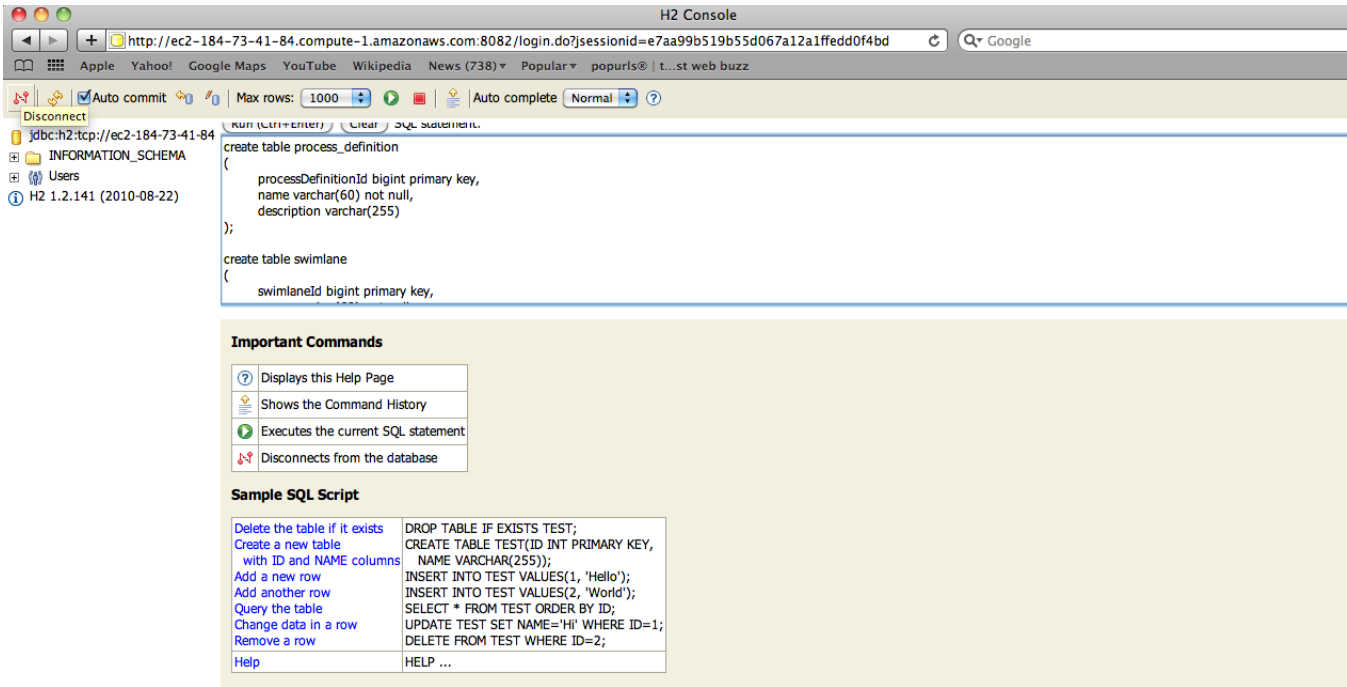
Valid values for cmd are start and stop to start and stop the database, respectively.

After the database is started, connect to the H2 console application, available on port 8082. Enter the jdbc URL to the database, the user name from applicationConfig used to access the database (sa), and the password (I used no password).

The JDBC URL has the following form:

jdbc:h2:tcp://<ec2 address>/~msedb;MVCC=TRUE

After establishing the connection to the database, create the database schema by pasting the contents of database.txt from http://code.google.com/p/ds-ksu-mse/source/browse/kstate-mse-h2-controller/src/main/resources/database.txt into the sql box and press the run button. This will create the database schema and seed the workflow definitions needed to execute the scenarios:



Deploying the Main Application

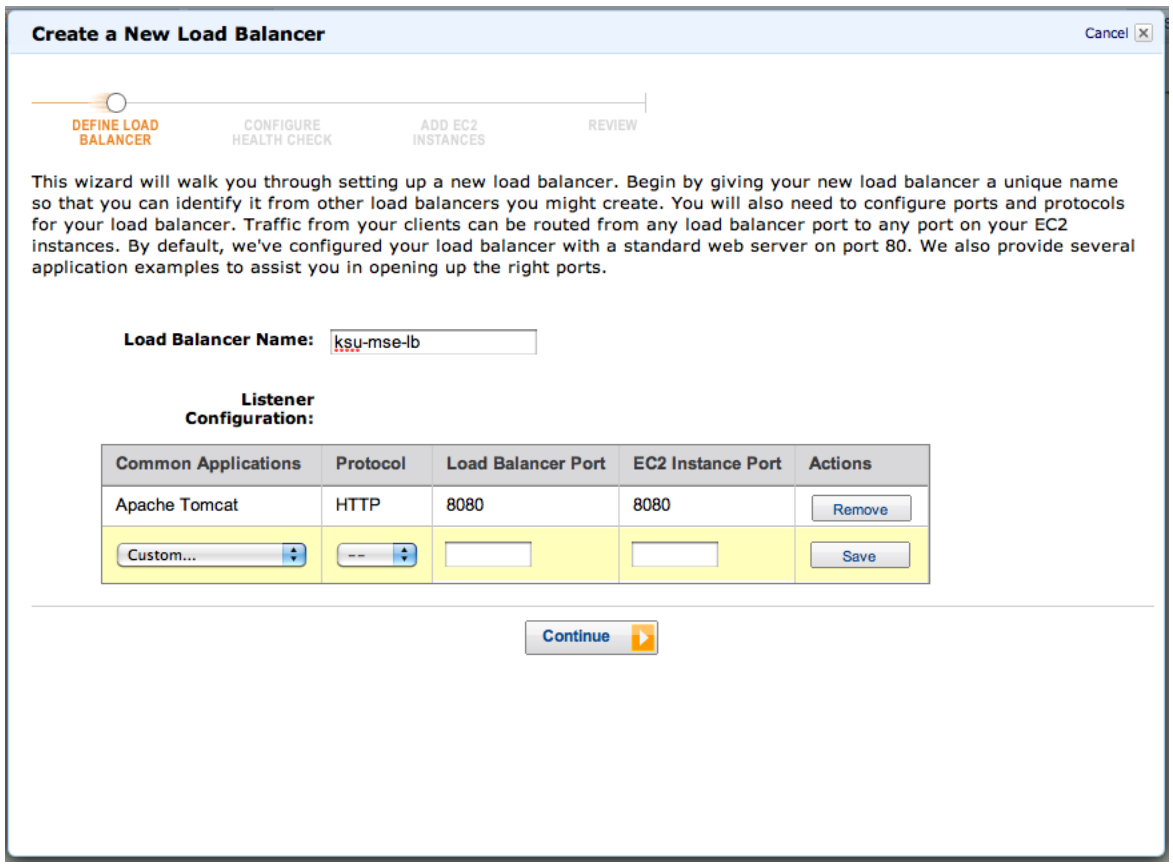
Once the database application has been deployed and started (and the schema installed), the main application is deployed as follows:

1. The first cluster member to host the application is started. After it starts, make a note of it's internal IP address from the console: this will be used as the known hazelcast cluster member. When multicast communication is used, one or more known cluster members are needed for the configuration of the system.
2. Edit the applicationConfig.xml file associated with the main application, and update the database JDBC URL with that corresponding to the database app that was just started. There are two URLs in the file.
3. Edit the hazelcast.xml file, using the IP of the known cluster member noted in step 1 in the configuration.
4. Deploy and start the application. This is done the same way as the database controller application was deployed and started.

Load Balancer Configuration

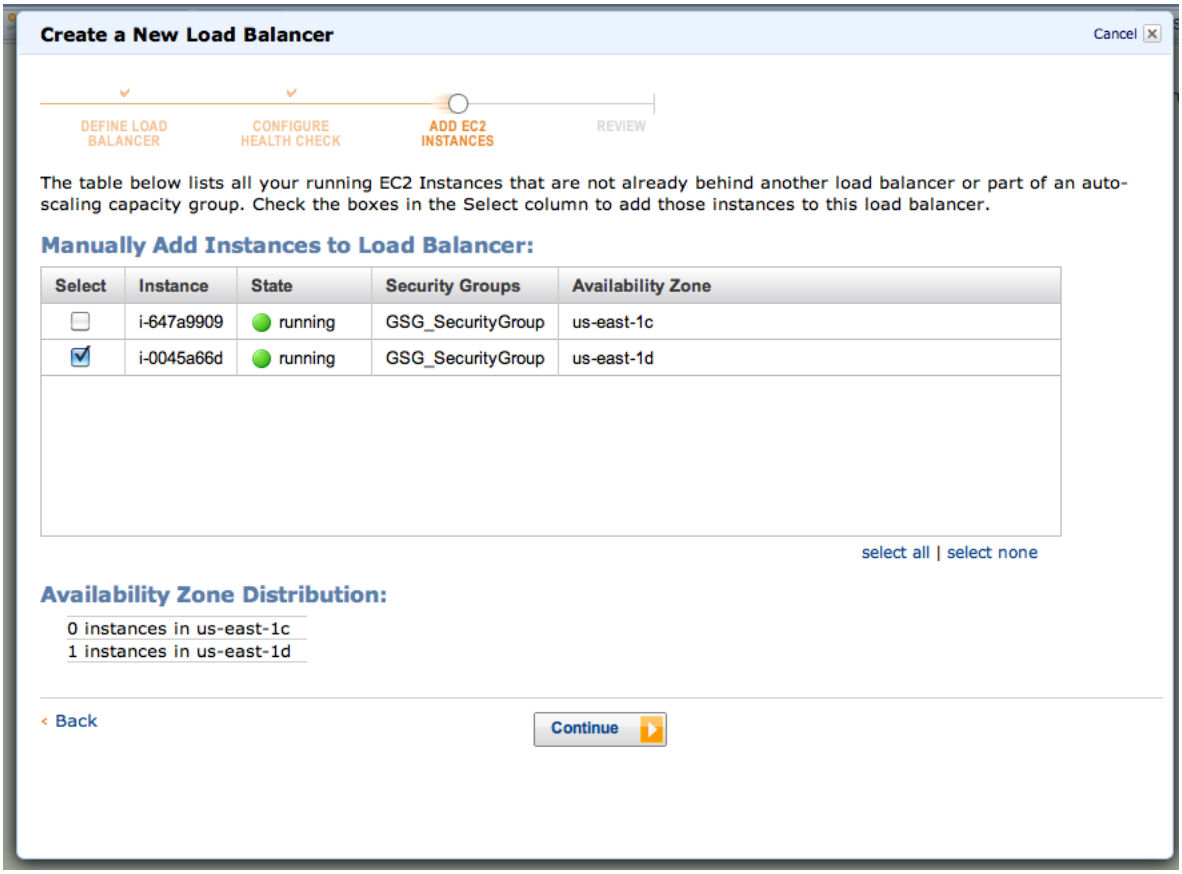
Once cluster members have been deployed running the main application, they should be fronted with a load balancer. Setting up a load balancer in EC2 is quite easy:

From the AWS EC2 console, select load balancers and create a new load balancer:

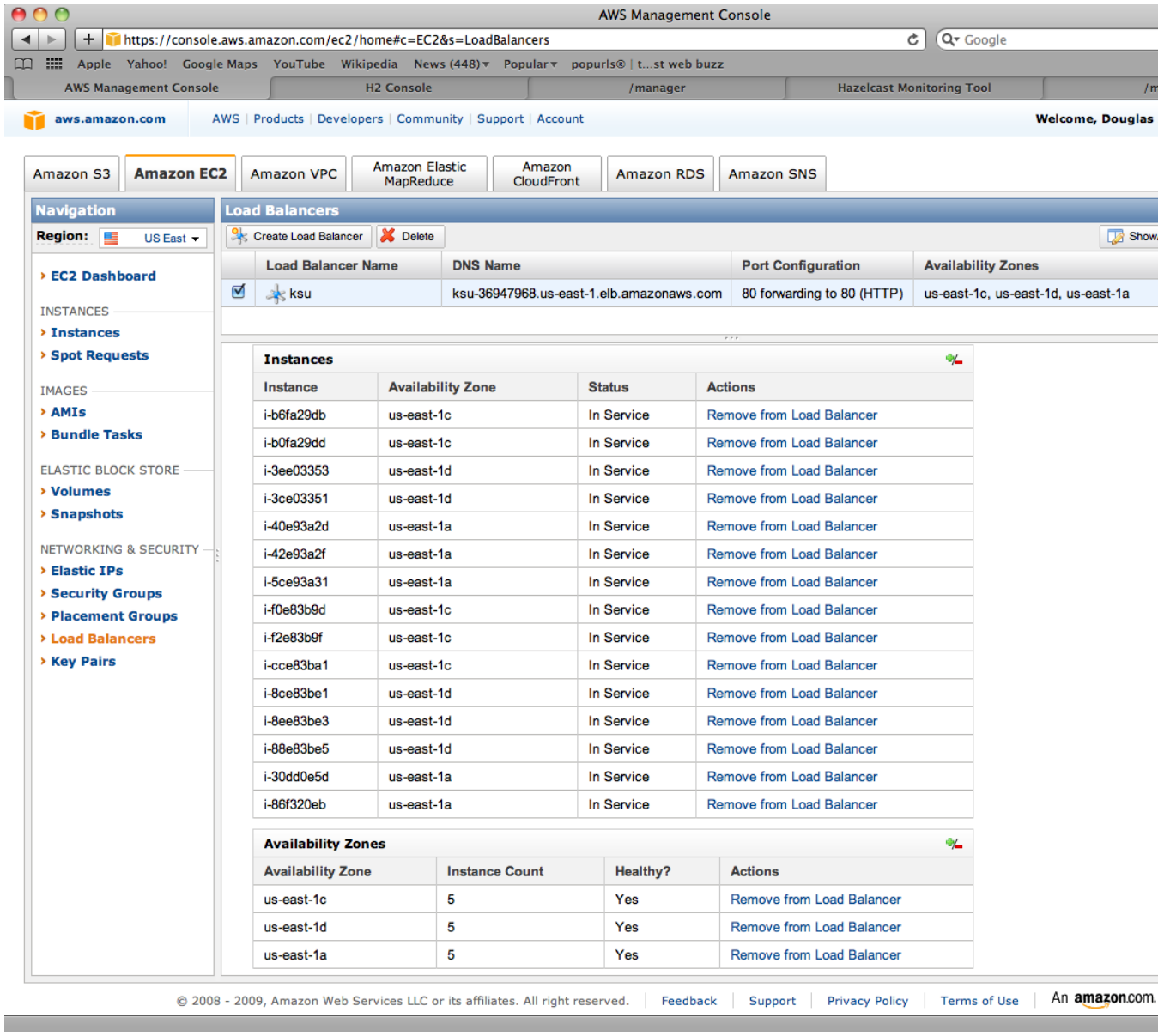


Next, configure the health check. Note when using tomcat server on the instance prepared as detailed in this document, the health check port is 80.

Next, add EC2 instances:



After that, review settings and create the load balancer if everything looks correct. Once the load balancer has been created the AWS console can be used to add and remove servers, review the health of load balancer members, and so on:



Web Services

While the services have been implemented using JAX-WS annotations on Java objects, the Apache CXF framework makes WSDL descriptions of the services available at runtime. To obtain the WSDL for a web service, put the service endpoint URL into the browser with "?wsdl" appended, e.g.

For convenience, I have make the WSDL available via the code repository – see https://code.google.com/p/ds-ksu-mse/source/browse/#svn/service-wsdl
http://localhost:8080/kstate-mse-ds/services/PropertyDefinition?wsdl

HibernateStats

This service provides a way to obtain Hibernate statistics from a server instance (e.g. one JVM) via a web service interface.

This service provides a single operation: getStats. This returns all Hibernate statistics capture since system start time.

Sample input:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
  <soapenv:Header/>
  <soapenv:Body>
    <doug:getStats/>
  </soapenv:Body>
</soapenv:Envelope>
```

Sample output:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:getStatsResponse xmlns:ns2="http://people.cis.ksu.edu/dougs">
      <return>
        <name>QueryCacheHitCount</name>
        <value>0</value>
      </return>
      <return>
        <name>QueryCacheMissCount</name>
        <value>0</value>
      </return>
      <return>
        <name>QueryCachePutCount</name>
        <value>0</value>
      </return>
      <return>
        <name>SecondLevelCacheHitCount</name>
        <value>0</value>
      </return>
      <return>
        <name>SecondLevelCacheMissCount</name>
        <value>0</value>
      </return>
      <return>
        <name>SecondLevelCachePutCount</name>
        <value>0</value>
      </return>
      <return>
        <name>domain.Swimlane fetch count</name>
        <value>7</value>
      </return>
      <return>
        <name>domain.Swimlane insert count</name>
```

```
        <value>0</value>
      </return>
        <name>domain.PropertyDefinition fetch count</name>
        <value>21</value>
      </return>
      <return>
        <name>domain.PropertyDefinition insert count</name>
        <value>0</value>
      </return>
      <return>
        <name>domain.PropertyDefinition load count</name>
        <value>66</value>
      </return>
      <return>
        <name>domain.PropertyDefinition update count</name>
        <value>1</value>
      </return>
    </ns2:getStatsResponse>
  </soap:Body>
</soap:Envelope>
```


Performance Stats

The PerformanceStats web service provides operations related to retrieving performance counters.

getStats

This methods retrieves performance statistics from the JVM hosting the service.

Sample request:


```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
  <soapenv:Header/>
  <soapenv:Body>
    <doug:getStats/>
  </soapenv:Body>
</soapenv:Envelope>
```


Sample response:


```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:getStatsResponse xmlns:ns2="http://people.cis.ksu.edu/dougs">
      <return>
        <avgTime>47.0</avgTime>
        <call>service.ProcessExecution.retrieveProcessTaskList</call>
        <maxTime>63.0</maxTime>
        <minTime>31.0</minTime>
        <numberOfCalls>3.0</numberOfCalls>
      </return>
```

```
<return>
  <avgTime>51.53061224489796</avgTime>
  <call>com.jamonapi.allPages</call>
  <maxTime>483.0</maxTime>
  <minTime>0.0</minTime>
  <numberOfCalls>49.0</numberOfCalls>
</return>
<return>
  <avgTime>47.0</avgTime>
  <call>service.ProcessExecution.retrieveTask</call>
  <maxTime>47.0</maxTime>
  <minTime>47.0</minTime>
  <numberOfCalls>1.0</numberOfCalls>
</return>
<return>
  <avgTime>242.0</avgTime>
  <call>service.ProcessExecution.instantiateProcess</call>
  <maxTime>296.0</maxTime>
  <minTime>188.0</minTime>
  <numberOfCalls>2.0</numberOfCalls>
</return>
<return>
  <avgTime>234.0</avgTime>
  <call>service.PropertyDefinition.retrieveList</call>
  <maxTime>234.0</maxTime>
  <minTime>234.0</minTime>
  <numberOfCalls>1.0</numberOfCalls>
</return>
<return>
  <avgTime>3.8333333333333335</avgTime>
  <call>service.ProcessExecution.releaseClaim</call>
  <maxTime>31.0</maxTime>
  <minTime>0.0</minTime>
  <numberOfCalls>12.0</numberOfCalls>
</return>
<return>
  <avgTime>59.80555555555556</avgTime>
  <call>/kstate-mse-ds/services/ProcessExecution</call>
  <maxTime>483.0</maxTime>
  <minTime>0.0</minTime>
  <numberOfCalls>36.0</numberOfCalls>
</return>
<return>
  <avgTime>59.0</avgTime>
  <call>service.ProcessExecution.executeTask</call>
  <maxTime>141.0</maxTime>
  <minTime>16.0</minTime>
  <numberOfCalls>4.0</numberOfCalls>
</return>
<return>
  <avgTime>8.0</avgTime>
  <call>service.PropertyDefinition.updateDescription</call>
  <maxTime>16.0</maxTime>
```

```
        <minTime>0.0</minTime>
        <numberOfCalls>2.0</numberOfCalls>
      </return>
      <return>
        <avgTime>10.333333333333334</avgTime>
        <call>/kstate-mse-ds/services/PerformanceStats</call>
        <maxTime>31.0</maxTime>
        <minTime>0.0</minTime>
        <numberOfCalls>3.0</numberOfCalls>
      </return>
      <return>
        <avgTime>11.25</avgTime>
        <call>/kstate-mse-ds/services/HibernateStats</call>
        <maxTime>15.0</maxTime>
        <minTime>0.0</minTime>
        <numberOfCalls>4.0</numberOfCalls>
      </return>
      <return>
        <avgTime>47.0</avgTime>
        <call>service.ProcessExecution.findInstances</call>
        <maxTime>47.0</maxTime>
        <minTime>47.0</minTime>
        <numberOfCalls>1.0</numberOfCalls>
      </return>
      <return>
        <avgTime>12.833333333333334</avgTime>
        <call>service.ProcessExecution.claimActivity</call>
        <maxTime>31.0</maxTime>
        <minTime>0.0</minTime>
        <numberOfCalls>6.0</numberOfCalls>
      </return>
      <return>
        <avgTime>49.333333333333336</avgTime>
        <call>/kstate-mse-ds/services/PropertyDefinition</call>
        <maxTime>234.0</maxTime>
        <minTime>0.0</minTime>
        <numberOfCalls>6.0</numberOfCalls>
      </return>
      <return>
        <avgTime>70.0</avgTime>
        <call>service.ProcessExecution.retrieveTaskList</call>
        <maxTime>156.0</maxTime>
        <minTime>15.0</minTime>
        <numberOfCalls>4.0</numberOfCalls>
      </return>
    </ns2:getStatsResponse>
  </soap:Body>
</soap:Envelope>
```


reset

Reset performance statistics and counters in the JVM hosting the service.

Sample input:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
  <soapenv:Header/>
  <soapenv:Body>
    <doug:reset/>
  </soapenv:Body>
</soapenv:Envelope>
```

Sample output:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:resetResponse xmlns:ns2="http://people.cis.ksu.edu/dougs"/>
  </soap:Body>
</soap:Envelope>
```

Process Execution

Claim Activity

This operation pulls an activity from the system, if one is available. The input is the swimlane name from which the activity should be pulled from. The response is the id of the claimed activity, if an activity was available and the claim granted, or -1 if no activity was available or an invalid swimlane name was provided.

Sample request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
  <soapenv:Header>
    <doug:userid>ds</doug:userid>
  </soapenv:Header>
  <soapenv:Body>
    <doug:claimActivity>
      <swimlaneName>Scan dept</swimlaneName>
    </doug:claimActivity>
  </soapenv:Body>
</soapenv:Envelope>
```

Sample response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:claimActivityResponse xmlns:ns2="http://people.cis.ksu.edu/dougs">
      <return>1290640986740</return>
    </ns2:claimActivityResponse>
  </soap:Body>
</soap:Envelope>
```

Execute Task

This operation is used to execute a task. The task id is supplied, along with the set of task data as specified by the task definition metadata. A fault is generated is the caller has not claimed the task, or if the full set of data is not supplied.

Sample request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
  <soapenv:Header>
    <doug:userid>ds</doug:userid>
  </soapenv:Header>
  <soapenv:Body>
    <doug:executeTask>
      <taskId>1290640986740</taskId>
      <fieldData>
        <name>p1</name>
        <value>foo</value>
      </fieldData>
<fieldData>
        <name>p2</name>
        <value>foo</value>
      </fieldData>
<fieldData>
        <name>p3</name>
        <value>foo</value>
      </fieldData>
<fieldData>
        <name>p4</name>
        <value>foo</value>
      </fieldData>
<fieldData>
        <name>p5</name>
        <value>foo</value>
      </fieldData>
<fieldData>
        <name>p6</name>
        <value>foo</value>
      </fieldData>
<fieldData>
        <name>p7</name>
        <value>foo</value>
      </fieldData>
<fieldData>
        <name>p8</name>
        <value>foo</value>
      </fieldData>
<fieldData>
        <name>p9</name>
        <value>foo</value>
      </fieldData>
```

```
<fieldData>
          <name>p10</name>
         <value>foo</value>
       </fieldData>
     </doug:executeTask>
   </soapenv:Body>
</soapenv:Envelope>
```

Sample response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:executeTaskResponse xmlns:ns2="http://people.cis.ksu.edu/dougs"/>
   </soap:Body>
</soap:Envelope>
```

Find Instances

This service is used to search for process instances based on property values associated with any of the process's activities. Inputs are the property values to search for.

Sample request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
   <soapenv:Header>
      <doug:userid>ds</doug:userid>
   </soapenv:Header>
   <soapenv:Body>
      <doug:findInstances>
         <fields>
            <name>p1</name>
            <value>foo</value>
         </fields>
      </doug:findInstances>
   </soapenv:Body>
</soapenv:Envelope>
```

Sample response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:findInstancesResponse xmlns:ns2="http://people.cis.ksu.edu/dougs">
         <return>1290640986691</return>
      </ns2:findInstancesResponse>
   </soap:Body>
</soap:Envelope>
```

Instantiate Process

This operation is used to instantiate a process. It's input argument is the name of the process to instantiate, and it returns a process instance if the specified process exists, and a fault if not.

Sample request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
   <soapenv:Header>
      <doug:userid>ds</doug:userid>
   </soapenv:Header>
   <soapenv:Body>
      <doug:instantiateProcess>
         <!--Optional:-->
         <processName>update beneficiary v2</processName>
      </doug:instantiateProcess>
   </soapenv:Body>
</soapenv:Envelope>
```

Sample response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:instantiateProcessResponse xmlns:ns2="http://people.cis.ksu.edu/dougs">
         <return>1290640986691</return>
      </ns2:instantiateProcessResponse>
   </soap:Body>
</soap:Envelope>
```

Release Claim

Release a claimed activity. Input is the activity a user has previous claimed. Output is a SOAP acknowledgement. Note the acknowledgement is returned

Sample request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
   <soapenv:Header>
      <doug:userid>ds</doug:userid>
   </soapenv:Header>
   <soapenv:Body>
      <doug:releaseClaim>
         <taskId>1290640986740</taskId>
      </doug:releaseClaim>
   </soapenv:Body>
</soapenv:Envelope>
```

Sample response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
   <soap:Body>
      <ns2:releaseClaimResponse xmlns:ns2="http://people.cis.ksu.edu/dougs"/>
   </soap:Body>
</soap:Envelope>
```

Retrieve Process Task List

This service retrieves all the tasks associated with a specific process at a specified swimlane.

Sample request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
  <soapenv:Header>
    <doug:userid>ds</doug:userid>
  </soapenv:Header>
  <soapenv:Body>
    <doug:retrieveProcessTaskList>
      <processId>1290640986691</processId>
      <swimlaneId>3</swimlaneId>
    </doug:retrieveProcessTaskList>
  </soapenv:Body>
</soapenv:Envelope>
```

Sample response:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:retrieveProcessTaskListResponse xmlns:ns2="http://people.cis.ksu.edu/dougs">
      <return>
        <activityId>1290640986739</activityId>
        <processId>1290640986691</processId>
        <state>Pending</state>
        <swimlaneId>3</swimlaneId>
      </return>
    </ns2:retrieveProcessTaskListResponse>
  </soap:Body>
</soap:Envelope>
```

Retrieve Task

Retrieve the data associated with a test. Input is the activity id, which can be obtained via retrieveTaskList or claimActivity.

Sample request:

```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
  <soapenv:Header>
    <doug:userid>ds</doug:userid>
  </soapenv:Header>
  <soapenv:Body>
    <doug:retrieveTask>
      <taskId>1290640986740</taskId>
    </doug:retrieveTask>
  </soapenv:Body>
</soapenv:Envelope>
```

Sample response:


```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:retrieveTaskResponse xmlns:ns2="http://people.cis.ksu.edu/dougs">
      <return>
        <activityId>1290640986740</activityId>
        <processId>1290640986691</processId>
        <state>Active</state>
        <swimlaneId>1</swimlaneId>
      </return>
    </ns2:retrieveTaskResponse>
  </soap:Body>
</soap:Envelope>
```


Retrieve Task List

Retrieve a list of tasks at a specific swimlane. Input is a swimlane id, output is the activities at the swimlane, regardless of the state. If the swimlane id is invalid, or there are no activities at the swimlane, an empty list is returned.

Sample request:


```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
  <soapenv:Header>
    <doug:userid>ds</doug:userid>
  </soapenv:Header>
  <soapenv:Body>
    <doug:retrieveTaskList>
      <swimlaneId>1</swimlaneId>
    </doug:retrieveTaskList>
  </soapenv:Body>
</soapenv:Envelope>
```


Sample response:


```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:retrieveTaskListResponse xmlns:ns2="http://people.cis.ksu.edu/dougs">
      <return>
        <activityId>1290442142418</activityId>
        <processId>0</processId>
        <state>Active</state>
        <swimlaneId>1</swimlaneId>
      </return>

      <return>
        <activityId>1290442359549</activityId>
        <processId>1290442359521</processId>
        <properties>
          <name>p1</name>
```

```
            <value>p1</value>
          </properties>
          <properties>
            <name>p4</name>
            <value>p4</value>
          </properties>
          <properties>
            <name>p6</name>
            <value>p6</value>
          </properties>
          <properties>
            <name>p3</name>
            <value>p3</value>
          </properties>
          <properties>
            <name>p9</name>
            <value>p9</value>
          </properties>
          <properties>
            <name>p2</name>
            <value>p2</value>
          </properties>
          <properties>
            <name>p10</name>
            <value>p10</value>
          </properties>
          <properties>
            <name>p8</name>
            <value>p8</value>
          </properties>
          <properties>
            <name>p5</name>
            <value>p5</value>
          </properties>
          <properties>
            <name>p7</name>
            <value>p7</value>
          </properties>
          <state>Complete</state>
          <swimlaneId>1</swimlaneId>
        </return>


      </ns2:retrieveTaskListResponse>
    </soap:Body>
  </soap:Envelope>
```


Property Definition

The PropertyDefinition web service is an example of a service interface for accessing and defining metadata used at runtime in the execution of a process.

retrieveList

This operation retrieves a list of the property definitions in the system.

Sample request:


```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
  <soapenv:Header/>
  <soapenv:Body>
    <doug:retrieveList/>
  </soapenv:Body>
</soapenv:Envelope>
```


Sample response:


```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:retrieveListResponse xmlns:ns2="http://people.cis.ksu.edu/dougs">
      <return>
        <description>description</description>
        <name>1290442142962</name>
        <propertydefinitionid>1</propertydefinitionid>
        <type>1</type>
      </return>
      <return>
        <description>description</description>
        <name>1290442143008</name>
        <propertydefinitionid>2</propertydefinitionid>
        <type>1</type>
      </return>
      <return>
        <description>description</description>
        <name>1290442359288</name>
        <propertydefinitionid>13</propertydefinitionid>
        <type>1</type>
      </return>
      <return>
        <description>description</description>
        <name>1290442359303</name>
        <propertydefinitionid>14</propertydefinitionid>
        <type>1</type>
      </return>
      <return>
        <description>description</description>
        <name>1290443919724</name>
        <propertydefinitionid>25</propertydefinitionid>
        <type>1</type>
      </return>
      <return>
        <description>description</description>
        <name>1290443919739</name>
        <propertydefinitionid>26</propertydefinitionid>
```

```
    <type>1</type>
  </return>
  <return>
    <description>description</description>
    <name>1290526831730</name>
    <propertydefinitionid>37</propertydefinitionid>
    <type>1</type>
  </return>
  <return>
    <description>description</description>
    <name>1290526831761</name>
    <propertydefinitionid>38</propertydefinitionid>
    <type>1</type>
  </return>
  <return>
    <description>update</description>
    <name>1290527977616</name>
    <propertydefinitionid>49</propertydefinitionid>
    <type>1</type>
  </return>
  <return>
    <description>update</description>
    <name>1290527977647</name>
    <propertydefinitionid>50</propertydefinitionid>
    <type>1</type>
  </return>
  <return>
    <description>description</description>
    <name>state</name>
    <propertydefinitionid>29000</propertydefinitionid>
    <type>2</type>
  </return>
  <return>
    <description>description</description>
    <name>status</name>
    <propertydefinitionid>29001</propertydefinitionid>
    <type>2</type>
  </return>
  <return>
    <description>description</description>
    <name>policy_no</name>
    <propertydefinitionid>30000</propertydefinitionid>
    <type>1</type>
  </return>
  <return>
    <description>description</description>
    <name>beneficiary</name>
    <propertydefinitionid>30001</propertydefinitionid>
    <type>2</type>
  </return>
  <return>
    <description>description</description>
    <name>p1</name>
```

```
        <propertydefinitionid>40000</propertydefinitionid>
        <type>2</type>
      </return>
      <return>
        <description>description</description>
        <name>p2</name>
        <propertydefinitionid>40001</propertydefinitionid>
        <type>2</type>
      </return>
    </ns2:retrieveListResponse>
  </soap:Body>
</soap:Envelope>
```


updateDescription

The updateDescription service takes a property name and a value for the description associated with the property. The property definition is updated with the supplied description. A fault is generated if a non-existent property name is supplied.

Sample input:


```
<soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:doug="http://people.cis.ksu.edu/dougs">
  <soapenv:Header/>
  <soapenv:Body>
    <doug:updateDescription>
      <name>p29</name>
      <description>new description</description>
    </doug:updateDescription>
  </soapenv:Body>
</soapenv:Envelope>
```


Sample output:


```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    <ns2:updateDescriptionResponse xmlns:ns2="http://people.cis.ksu.edu/dougs"/>
  </soap:Body>
</soap:Envelope>
```