# **Red Hat JBoss BRMS**

# **Weight Watcher Demo**

**Use case: stateless CEP decision service**

Stefano Picozzi

ANZ OpenShift SME

Sydney, Australia

spicozzi@redhat.com

**Table of Contents**

1 Introduction 3

2 Setup 3

2.1 Project Download 3

2.2 Software Downloads 4

3 Deployment 5

3.1 Traditional Deployment 5

3.2 Container Deployment with Rebuild 5

3.3 Container Deployment Prebuilt 6

4 Configuration 7

4.1 Complete Configuration 7

4.2 Minimal Configuration 7

4.3 SoapUI/REST Configuration 8

4.4 Workbench Configuration 9

4.5 Terminal Configuration 10

5 Running the Demo 11

5.1 Health Check 11

5.2 Basic Decision Server 12

5.3 Advanced Workbench 13

5.4 High Availability 14

# Introduction

Interested in a demo that showcases the JBoss BRMS 6.1 Real Time Decision Server? Then look here. The application is a stateless Decision Server with complex event processing (CEP) support based on a pseudo clock.

An example use case demonstrated includes a (REST) client sending a time series of *facts* in the form of weight observations to the Decision Server. The Decision Server then reasons over the inputs to derive CEP insights such as average weight, least weight and weight change of a rolling time window. These insights are returned to the calling client as *facts*.

This is a facts-in-facts-out (FIFO) pattern using a standardized fact interface representation. This technique makes it easier for a simple thin client application such as SoapUI to send request/response payloads to the Decision Server without knowledge of the underlying rules data model.

<http://blog.emergitect.com/2014/12/08/really-simple-rules-service/>

# Setup

## Project Download

You first need to get the project by cloning it from the central location:

|  |
| --- |
| $ git clone git://github.com/StefanoPicozzi/weightwatcher.git |

Once downloaded, you will have the following folder structure:

* \weightwatcher
  + \installs – Initially empty, but will contain the EAP, BRMS platform downloads.
  + \traditional – Artefacts to assist in a traditional workstation installation
  + \container – Artefacts to assist in a container based installation
  + \docs – Contains quickstart guide you are reading and architectural overview slides.
  + \test – SoapUI project file with test invocations
  + \src – source code including data model, rules and project settings

## Software Downloads

Some test cases and configuration steps make use of the SoapUI functional testing tool. If you do not have it, download and install SoapUI from <http://www.soapui.org/> .

Download JBoss BRMS from the Red Hat Customer Portal ([https://access.redhat.com](https://access.redhat.com/)).

1. Under JBoss Enterprise Platforms, select the BRMS product.
2. Select version *6.1.0* in the *Version* field.
3. Download Red Hat JBoss BMRS 6.1.0 installer

Then copy jboss-brms-6.1.0.GA-installer.jar, to the projects *installs* folder. Ensure that this file is executable by running:

|  |
| --- |
| $ chmod +x <path-to-project>/installs/jboss-brms-6.1.0.GA-installer.jar |

Download JBoss EAP 6.4:

1. Under JBoss Enterprise Platforms, select the EAP product.
2. Select version *6.4* in the *Version* field.
3. Download Red Hat JBoss eap 6.4 installer

Now copy jboss-EAP-6.4.0-installer.jar, to the projects *installs* folder. Ensure that this file is executable by running:

|  |
| --- |
| $ chmod +x <path-to-project>/installs/jboss-EAP-6.4.0.-installer.jar |

# Deployment

Various deployments models are supported as described below. In all cases, once the application is started, you can access the browser based workbench console via:

http://localhost:8080/business-central (u:spicozzi / p:jbossbrms1! )

## Traditional Deployment

This will deploy the application to \target directory. To do this, run the *init.sh* script and then configure the application as described in the next section.

|  |
| --- |
| $ cd <path-to-project>  $ cd traditional  $ ./init.sh |

When the script completes you will have a new folder named *jboss-eap-6.4*, in the \target folder. The folder is a ready to run EAP 6 server with JBoss BRMS. Launch an instance of your new BRMS application and then complete the configuration steps detailed in the next chapter.

|  |
| --- |
| $ cd <path-to-project>  $ ./traditional/target/jboss-eap-6.4.0/bin/standalone.sh |

## Container Deployment with Rebuild

This will deploy the application as a docker image. First build the image as per below. Mac OS/X users may want to install boot2docker first. Once completed, then configure the application as described in the next section.

|  |
| --- |
| $ cd <path-to-project>  $ cd container/build  $ cp -r ../../installs .  $ docker build -t spicozzi/weightwatcher . |

When the build is complete, launch a container then proceed to the next section for Configuration.

|  |
| --- |
| $ docker run -it -p 8080:8080 -p 9990:9990 spicozzi/weightwatcher |

## Container Deployment Prebuilt

This will deploy the application as a docker image of a fully configured application. This is the fast and simplest way to implement this application, as no further configuration is required but requires access to the weightwatcher.zip file. First build the image as per below. Mac OS/X users may want to install boot2docker first.

$ cd <path-to-project>

$ cd container/prebuilt

# Contact author for location of weightwatcher.zip

# Copy this .zip to the current location

$ docker build -t spicozzi/weightwatcher .

When the build is complete, launch a container, then skip Configuration and proceed directly to the Demonstration section.

|  |
| --- |
| $ docker run -it -p 8080:8080 -p 9990:9990 spicozzi/weightwatcher |

# Configuration

Users who have chosen the Container Prebuilt installation option can ignore this section as the application is already configured. For all other options, configuration consists of executing a few pre-supplied REST API calls from the SoapUI client, creating a repository, registering a new decision server and updating the git repository with the latest project source code.

The steps can be summarised as follows, which are executed either within the SoapUI tool, BRMS Workbench or via command line in a Terminal as annotated. Perform each in the order specified. Instructions for the three configuration tools are documented in the next sub-sections. Two configuration options exists for a minimal or complete configuration. Minimal is all that is required to show the Decision Server in action. The complete method allows the user to inspect and customise the rules and associated data model via the workbench.

Note that the intention is to fully automate this procedure via a script once APIs are located to replace the manual Workbench and Terminal steps.

## Complete Configuration

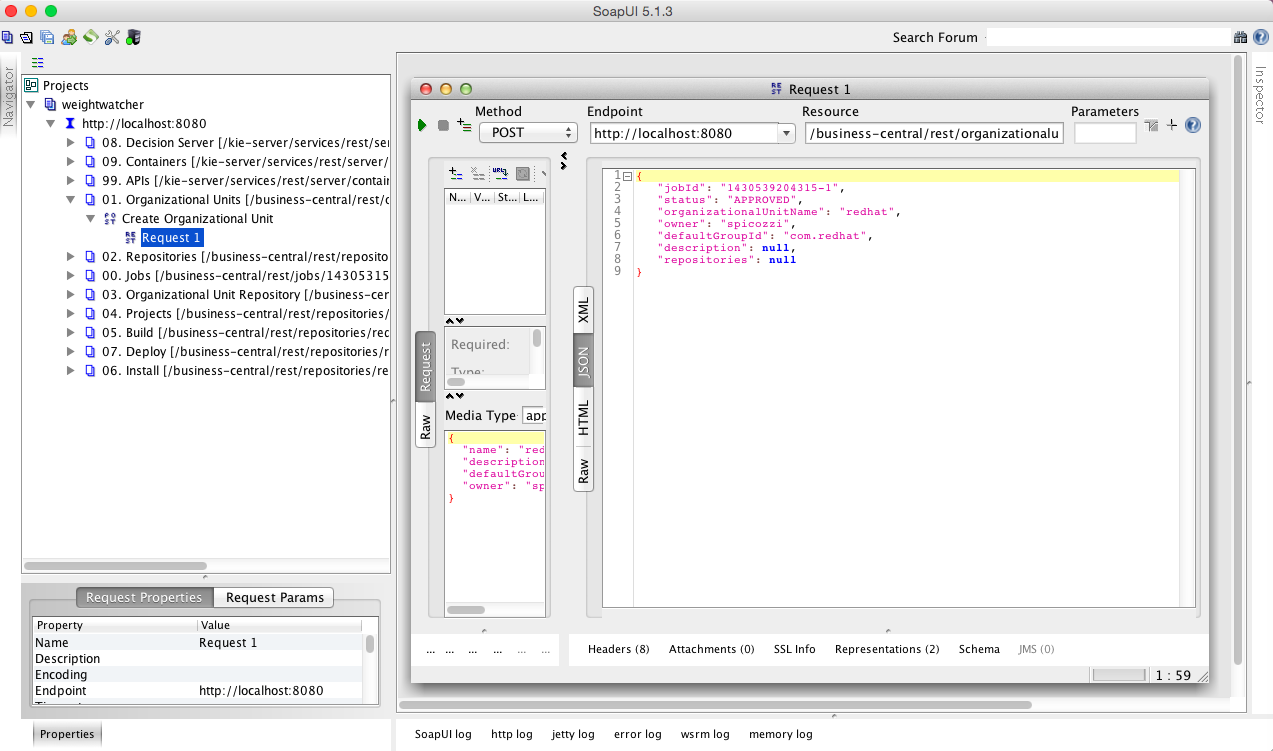
1. (SoapUI) 01. Create an organizational unit - POST
2. (Workbench) Create a repository
3. (SoapUI) 03. Associate the repository to the organizational unit - POST
4. (SoapUI) 04. Create a project - POST
5. (Terminal) Populate the project source code
6. (SoapUI) 05. Build the project - POST
7. (SoapUI) 06. Install the project - POST
8. (SoapUI) 07. Deploy the project - POST
9. (Workbench) Register a decision server
10. (SoapUI) 09. Create a container - PUT

## Minimal Configuration

1. (Workbench) Register a decision server
2. (SoapUI) 09. Create a container - PUT

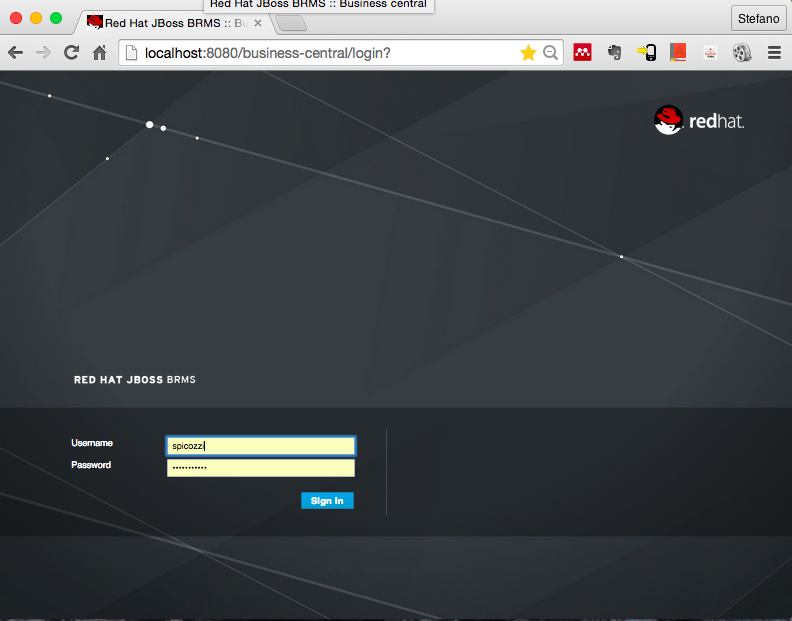
## SoapUI/REST Configuration

For steps marked SoapUI, launch the client and import in the project located under \tests. Then execute the numbered REST API call matching the appropriate step.

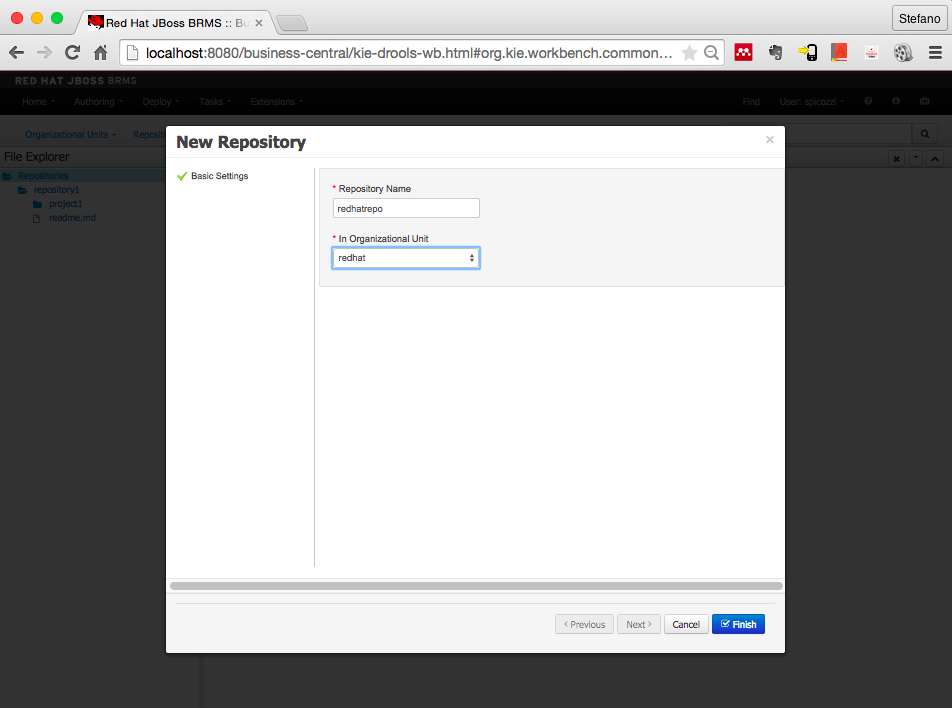


## Workbench Configuration

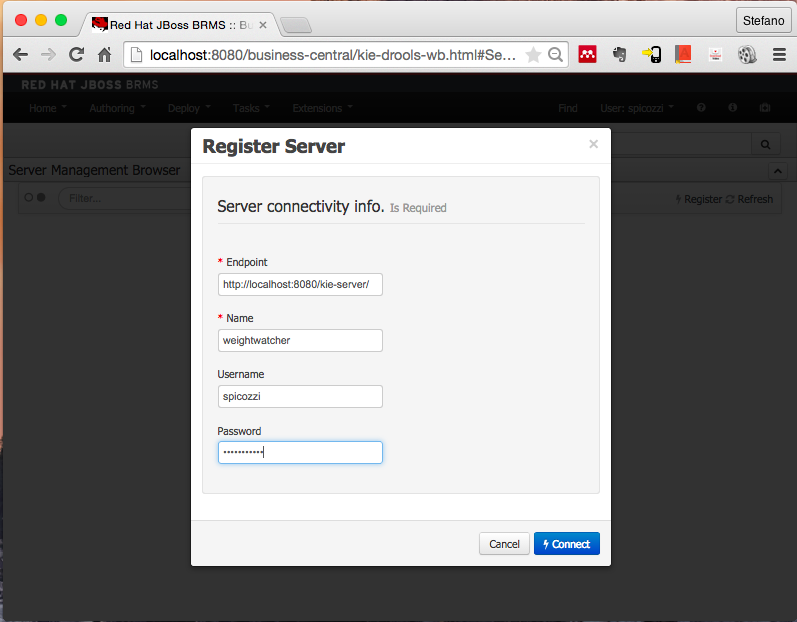
Workbench steps require you login first using u:spicozzi/p:jbossbrms1!.



The repository creation step is accessible from the Author/Administration menu. Create a repository named redhatrepo as follows:



The register new decision server step is accessible from the Deploy menu. Complete the registration as follows:



## Terminal Configuration

The source code git step requires you launch a terminal window. From the command line issue return to your demo installation location and replace the cloned empty src tree as follows:

$ cd <path-to-project>

$ git clone ssh:[//spicozzi@localhost](mailto://spicozzi@localhost):8001/redhatrepo

$ cd redhatrepo

$ cd weighwatcher

$ cp -r ../../src .

$ git add src

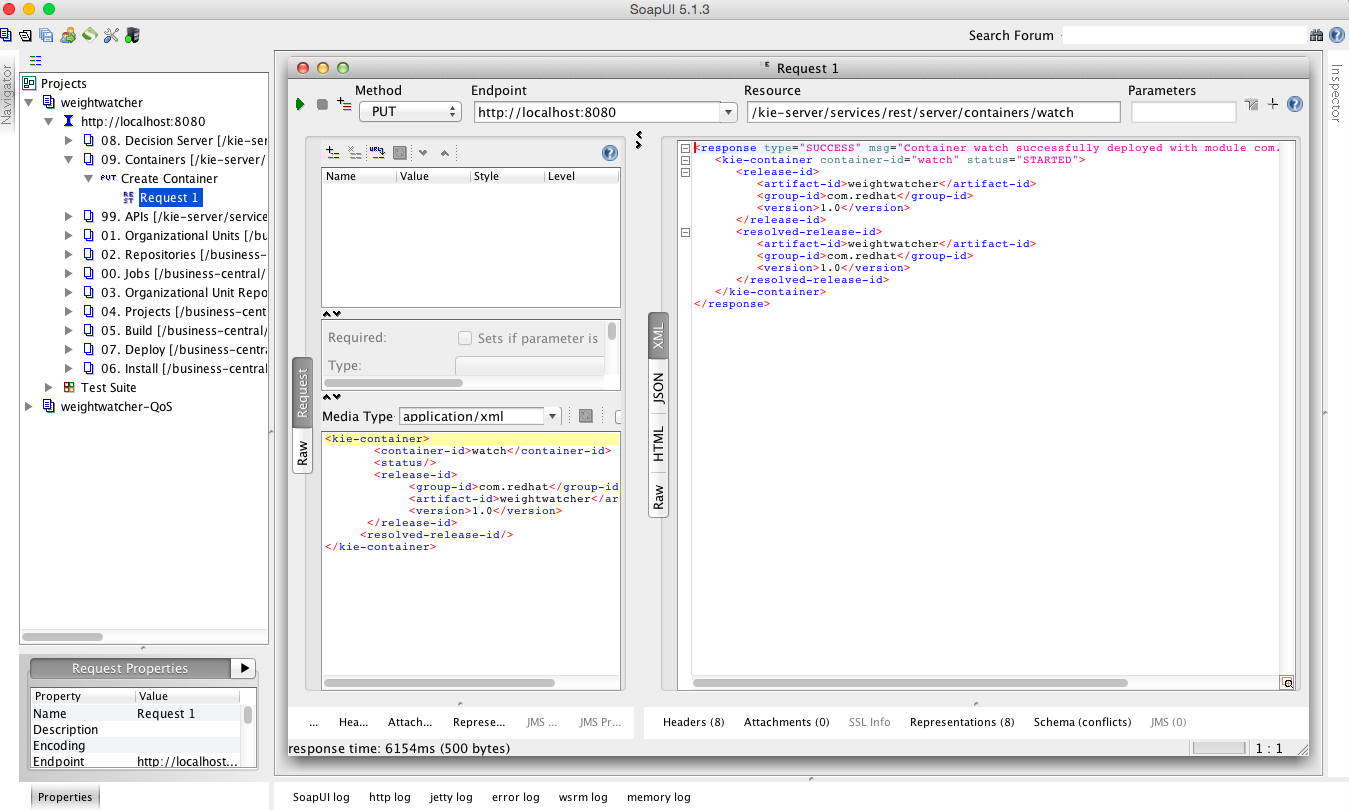
$ git commit -am "Replaced src"

$ git push

# Running the Demo

## Health Check

Before attempting any of the following demonstrations, ensure that the Decision Server is ready to accept requests. You can check this by repeating the Create Container step 10 in the Configuration section. This requires you invoke the 09. REST/PUT request using SoapUI. If this invocation fails, then check the earlier steps to ensure your application is ready.



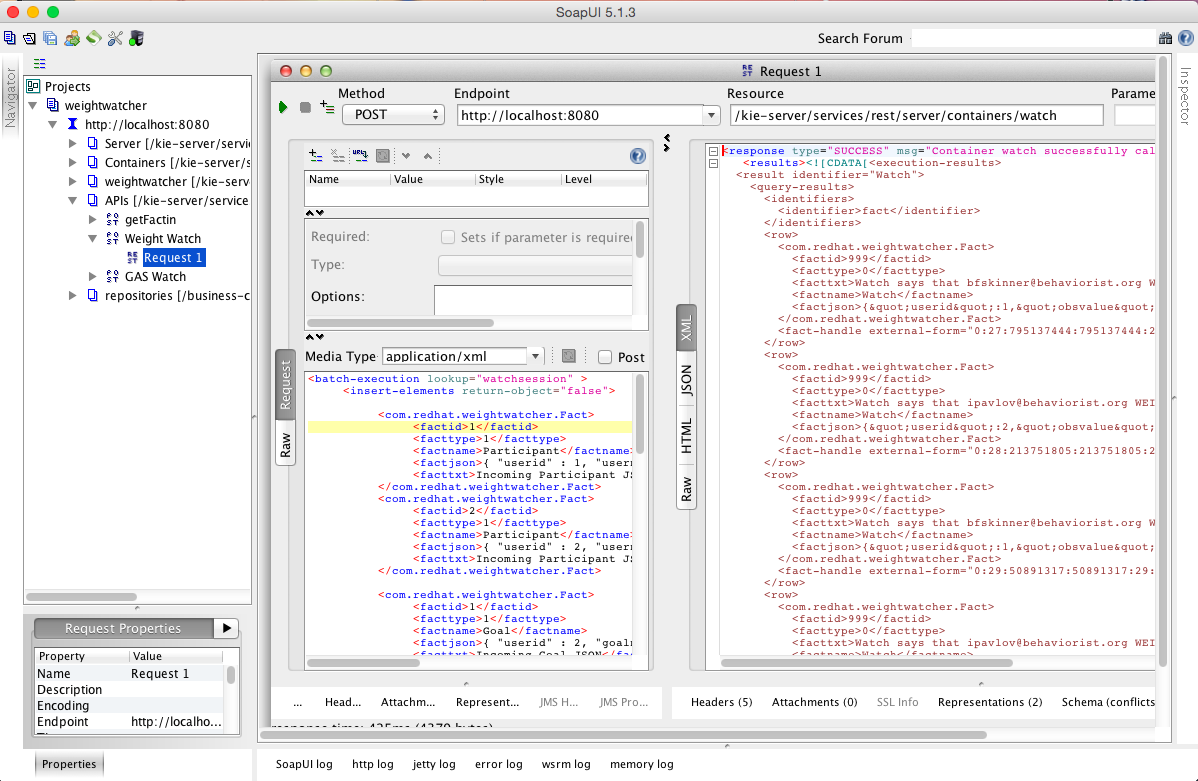
## Basic Decision Server

This demo shows a stateless request/response interaction with the Real Time Decision Server. Three samples are located under "APIs" Resource. If you wish to edit or create your own invocations simply follow the examples provided and edit the XML payload content appropriately.

"Facts" shows a simple request in which a request payload of facts are *inserted* into the Decision Server knowledge and then a *query* is issued to verify this action has been successful.

The "Weight Watch" sample shows an invocation in which a set of facts containing weight measurements is sent to the Decision Server. CEP rules are then applied to derive insights as per the response payload. The request consists of facts representing Participant, Goal and Observation data records. The Participant records capture details of the user, Goal captures the Participant's target weight objectives and Observation records a time series of weight measurements. The response payload then returns a set of facts reporting minimum, maximum and weight change statistics over a sliding time window.

The "Goal Attainment" sample demonstrates a use case in which the Participant has elected to enter into a period of intermittent fasting, known as the Fast Diet http://thefastdiet.co.uk/. The GAS fact represents the Participant's number of fasting day goals over the week, described in ranges of worst through to best outcomes, refer http://en.wikipedia.org/wiki/Goal\_Attainment\_Scaling for details on the method. The Observation records then report back actual days of fasting in the previous weeks. The Decision Server then responds back with performance against goals. The GAS fact table is a candidate for remodelling using, e.g. a Guided Decision Tables.



## Advanced Workbench

You can also experiment with changing and creating rules and observing their impact. To do this, try the following from within the JBoss BRMS workbench:

* Select the Deploy menu and delete the watch container created previously
* Visit the weightwatcher project and edit the DRL named GASScore under the weightwatcher project
* Change the rule “ruleExpectedCount” so that it only counts when the Participant meets exactly his expected result (obsvalue == 0) and change the message accordingly by removing the “or better” text
* Rebuild the workbench project jar file
* Select the Deploy option again and recreate the watch container, you can also do this using the REST API instruction as in section 2.1
* Start the watch container
* Now return to the SoapUI client and invoke the GAS Watch API and confirm the changed rule behavior

The modified rule will look like:

|  |
| --- |
| rule "ruleExpectedCount"  salience -100  no-loop true  when  $participant : Participant( )  $gas : GAS( userid == $participant.userid )  $obscounttotal : Number( intValue > 0) from accumulate(  Observation( $obscount : obsvalue == 0, $participant.userid == userid, obsname == $gas.goalname ) over window:time( 60d ),  count( $obscount ) )  then  String rulename= new String( drools.getRule().getName( ) );  Integer userid = $participant.getUserid( );  String factname = new String( "Watch" );  String username = new String( $participant.getUsername( ) );  String facttxt = new String( "Weightwatcher says that for " + $gas.getGoalname( ) + ", " + username + " attained expected outcome " +  $obscounttotal + " times over the past 60 days" );  // Rest removed |

## High Availability

A high-availability and load balancing demonstration is available for a Docker container based deployment. This will work on a single host if required. To do this, build an nginx container to act as a load balancing reverse proxy and then launch it with the docker run command. If your weightwatcher application(s) are deployed to localhost on ports 8080 and/or 8081 then simply pull down the prebuilt nginx container:

|  |
| --- |
| $ docker pull spicozzi/nginx |

If not using localhost, then you need to build your own nginx container. Before you initiate the nginx build command check the container/nginx/default file and edit the upstream weightwatcher target to reflect the hostname of your workstation as per the fragment shown below.

upstream weightwatcher {

server localhost:8080;

server localhost:8081;

}

|  |
| --- |
| $ cd <path-to-project>  $ cd container/nginx  $ vi nginix/default # Change hostnames as appropriate  $ docker build -t spicozzi/nginx .  $ docker run -d -p 80:80 --name nginx spicozzi/nginx \  nginx -g "daemon off;" |

The nginx instance will proxy URIs with /kie-server/ to an upstream pair of docker BRMS server containers listening on 8080 and 8081 respectively. Run each docker as per below. Note that this will require a workstation with more than 8 GBytes RAM.

|  |
| --- |
| $ cd <path-to-project>  $ docker build -t spicozzi/weightwatcher .  $ docker run -d -p 8080:8080 -p 9090:9090 --name weightwatcher1 \  spicozzi/weightwatcher  $ docker run -d -p 8081:8080 -p 9091:9090 --name weightwatcher2 \  spicozzi/weightwatcher \ |

Other useful docker commands include, using weightwatcher1 named instance as an example:

|  |
| --- |
| $ docker ps -l  $ docker rm -f weightwatcher1  $ docker logs -f weightwatcher1 |

Once the docker containers are tunning, you can then experiment with unit and load testing as per tests located in the SoapUI projects as per screen shot below. To do this import the weightwatcher-QoS project under /tests. Remember to ensure you have started the Container in each BRMS instance using the supplied REST API.

