

# **Red Hat JBoss BRMS Weight Watcher Demo**

Use case: stateless CEP decision service



Stefano Picozzi  
Sydney, Australia

[spicozzi@emergitect.com](mailto:spicozzi@emergitect.com)  
<http://blog.emergitect.com>

# Table of Contents

Table of Contents .....	2
1 Introduction .....	3
2 Setup .....	4
2.1 Project Download.....	4
2.2 Software Downloads .....	5
2.2.1 Sample Client Tools.....	5
2.2.2 JBoss BRMS.....	5
2.2.3 JBoss EAP .....	5
3 Traditional Deployment .....	6
3.1 Installation Script .....	6
4 Configuration.....	7
4.1 Register Decision Server.....	8
4.2 Build/Deploy the Project.....	9
4.3 Create/Start a KIE Server Container.....	9
5 Running the Demo .....	10
5.1 Companion Website Samples .....	10
5.2 cURL Samples .....	11
5.3 SoapUI Samples.....	12
5.4 R using RStudio Samples .....	13
5.5 Changing Rules .....	14
5.6 High Availability.....	15
5.6.1 Registry Pull Option.....	15
5.4.1 Local Build Option .....	15
5.4.2 Tests .....	16
5.7 Rule Changes with Docker Container.....	17
6 Docker Containers .....	18
6.1 Registry Pull Option.....	18
6.2 Local Build Option .....	18
7 OpenShift V3.0 GA.....	20
7.1 OpenShift Environment Setup .....	20
7.2 Create weightwatcher OpenShift Application.....	21
7.3 Create RStudio OpenShift Application .....	22
7.4 Running the RStudio Sample.....	23
7.5 OpenShift Console.....	23
7.6 Create Website OpenShift Application .....	23
A Appendix .....	24
A.1 Useful boot2docker Docker Commands .....	24

# 1 Introduction

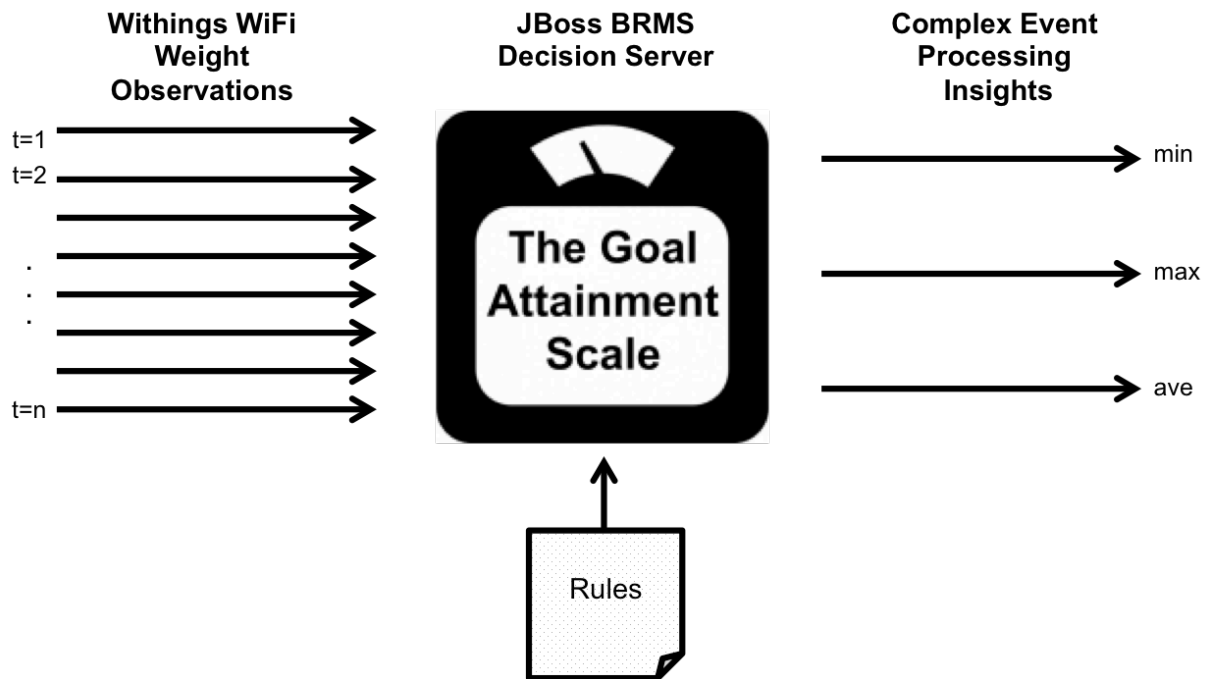
Interested in a demo that showcases the JBoss BRMS 6.1 Real Time Decision Server? Then look here. The application is a stateless Decision Server with complex event processing (CEP) support based on a pseudo clock.

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 or RStudio 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/>

Both a traditional and Docker container and OpenShift V3 based deployment models are supported. If choosing to use the Container approach you can skip sections 3 and 4. Docker container installation is described in section 6 and OpenShift V3 in section 7.



## 2 Setup

### 2.1 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
  - \container – Artefacts to assist in nginx container installation
  - \docs – Contains quickstart guide you are reading and architectural overview slides.
  - \installs – Initially empty, but will contain the EAP, BRMS platform downloads.
  - \m2 – JBoss BRMS readable rules and data model artefacts
  - \OpenShiftV3 – json configuration for OpenShift V3 application
  - \projects – Original source code including data model, rules and project settings
  - \support – Artefacts to assist in a traditional workstation installation
  - \tools – SoapUI and R example project files with configuration, samples and test invocations

## 2.2 Software Downloads

### 2.2.1 Sample Client Tools

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/>.

Similarly, test samples for R have also been provided which make use of the RStudio tool which can be installed as per instructions at <http://blog.emergitect.com/2015/06/15/4castr-on-rstudio-server-as-a-docker-container/>.

### 2.2.2 JBoss BRMS

Download JBoss BRMS from the Red Hat Customer Portal (<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 BRMS 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
```

### 2.2.3 JBoss EAP

Download JBoss EAP 6.4:

4. Under JBoss Enterprise Platforms, select the EAP product.
5. Select version *6.4* in the *Version* field.
6. 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
```

## 3 Traditional Deployment

Various traditional 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:erics / p:jbossbrms1!)`

### 3.1 Installation Script

This step 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>
$ ./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>
$ ./target/jboss-eap-6.4.0/bin/standalone.sh
```

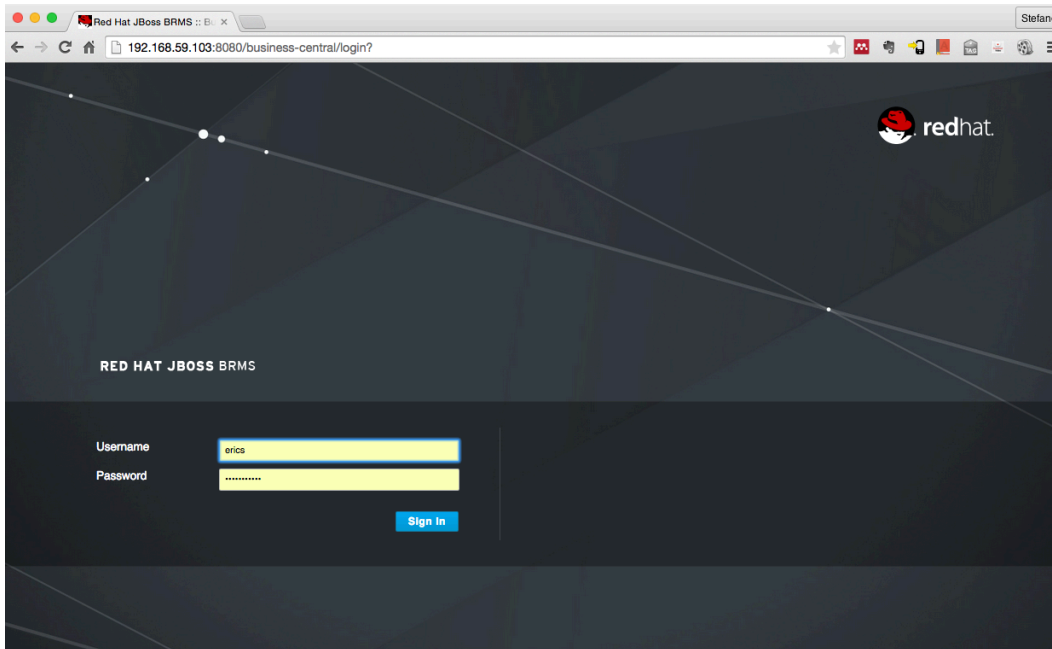
## 4 Configuration

Configuration consists of 1) building/deploying the project jar file and 2) registering a new Decision Server and 3) starting a Container for our new server. All steps aside from server registration can be completed using an API. The intention is to fully automate the procedure via a script/maven once APIs are located to replace the manual Workbench. Configuration is summarised as follows:

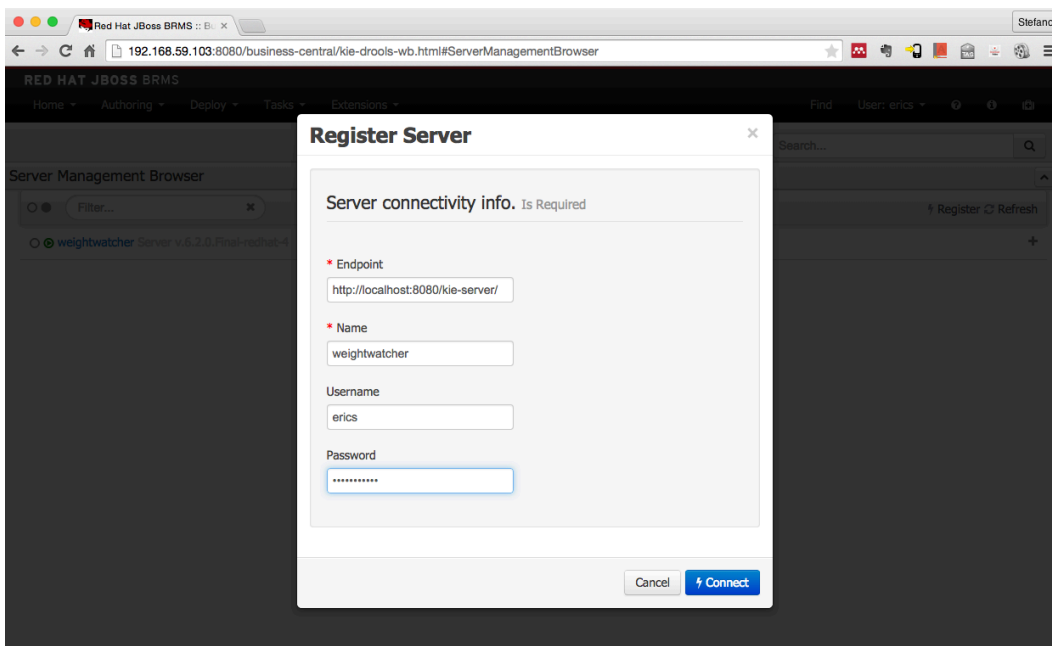
1. (Workbench) Login to the workbench and then register a new Decision Server
2. (SoapUI) Build/deploy the project with the supplied REST/POST command
3. (SoapUI) Create/start a container for our new server using the supplied REST/PUT command

## 4.1 Register Decision Server

Workbench related steps require you login first ( u:erics/p:jbossbrms1! )



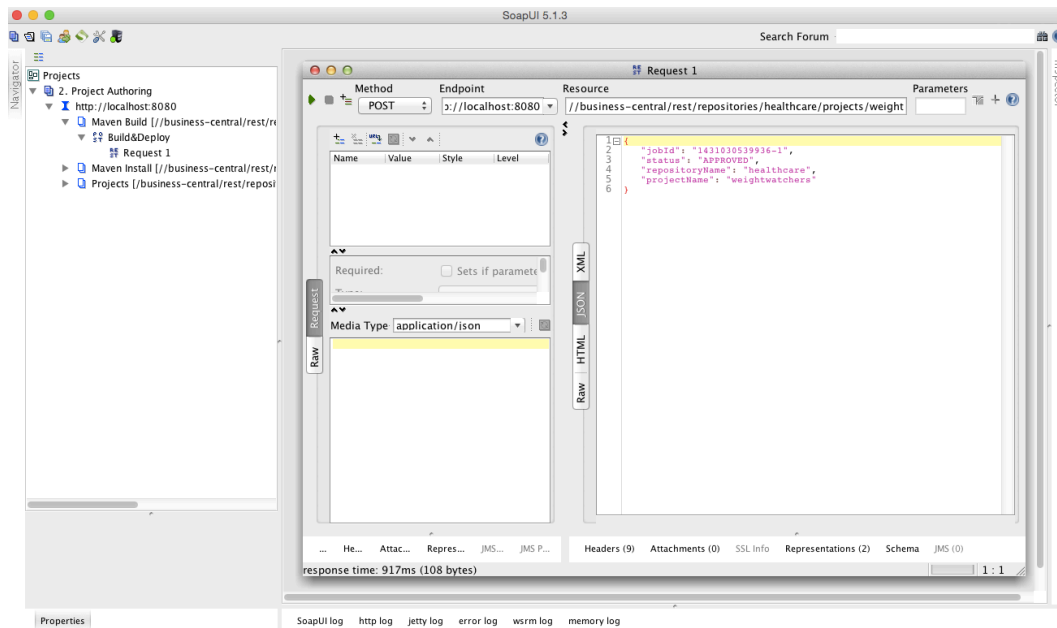
The register new decision server step is accessible from the Deploy menu. Complete the registration as follows:





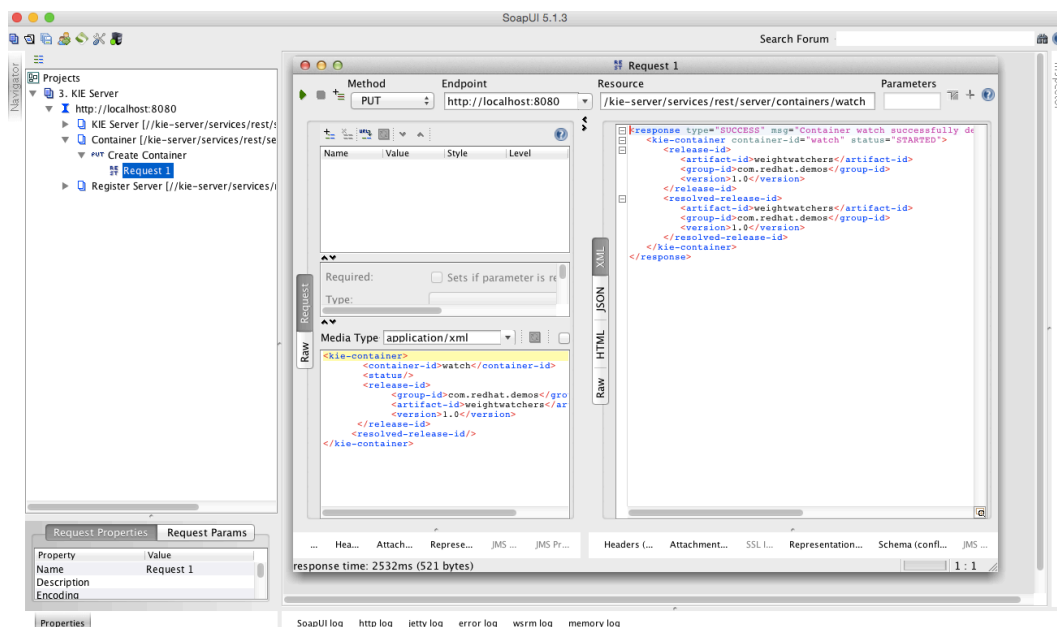
## 4.2 Build/Deploy the Project

Launch the SoapUI tools and then import the project with a name that includes the label "Project Management". Locate the "Maven Build" resource and then execute the REST/POST Build&Deploy request.



## 4.3 Create/Start a KIE Server Container

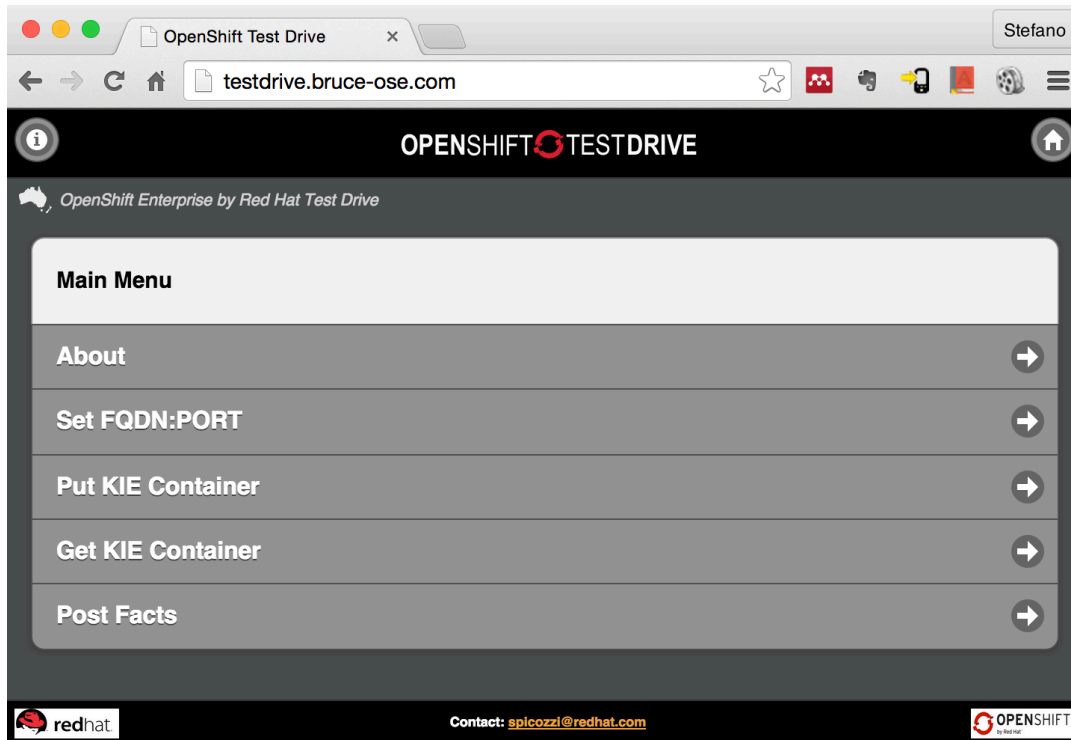
Launch the SoapUI tools and then import the project with a name that includes the label "KIE Server". Locate the "Container" resource and then execute the REST/PUT Create Container request.



## 5 Running the Demo

### 5.1 Companion Website Samples

A companion website is available that hosts some test use case invocations. Point your browser to <http://testdrive.bruce-ose.com/> and follow the instructions in the About page.



This sample application is also available as a docker image which you can access and run locally using:

```
$ docker pull spicozzi/testdrive
$ docker run -it -p 8081:80 spicozzi/testdrive
```

## 5.2 cURL Samples

Some simple cURL scripts have also been supplied to test the health of your configuration. These are similar to the test cases available at the companion website. These are located under the /tools/cURL subdirectory. Edit each script to change the http:// FQDN end point to reflect your deployment as follows:

```
$ cd <path-to-project>
$ cd tools/cURL

# Edit the PUT-kiecontainer.sh script and change the FQDN as necessary
$ ./put-KIEcontainer.sh
# Response payload will show SUCCESS or FAILURE depending on whether
# KIEcontainer "watch" already exists or not

# Edit the get-KIEcontainer.sh script and change the FQDN as necessary
$ ./get-KIEcontainer.sh
# Check for 200 http response code
# Response payload should show SUCCESS as KIEcontainer status is STARTED

# Edit the post-facts.sh script and change the FQDN as necessary
$ ./post-facts.sh
# Check for 200 http response code
# Response payload should show list of JBoss BRMS CEP notifications
```

More sophisticated examples and use cases follow. Note that before attempting any of the following demonstrations, ensure that the Decision Server's Container is ready to accept requests. Check this by running the get- and/or put-KIEcontainer.sh scripts described above.

Note that if you edit/run the supplied container/heartbeat.sh script in a separate window, then there is no need to restart the KIE server container each time you restart JBoss.

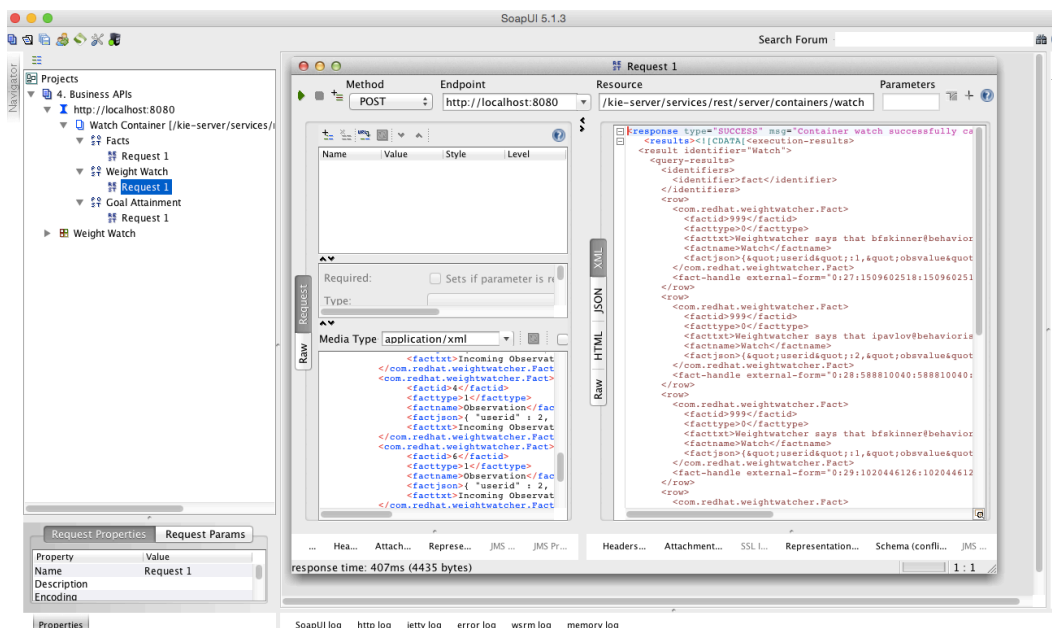
## 5.3 SoapUI Samples

This demo shows a stateless request/response interaction with the Real Time Decision Server. Launch the SoapUI tools and then import the project with a name that includes the label "Business APs". The 3 supplied resources and REST POST requests are samples representing the following.

"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.

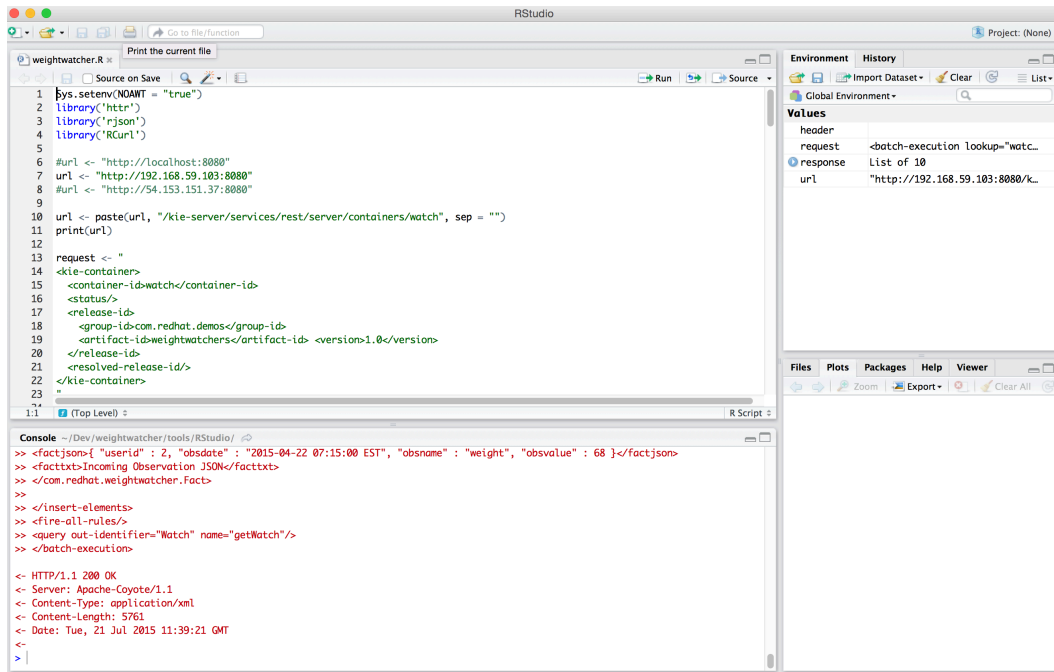
The "Goal Attainment" sample demonstrates a use case in which the Participant has elected to enter into a period of intermittent fasting, known as the Fast Diet <http://thefastdiet.co.uk/>. The GAS fact represents the Participant's number of fasting day goals over the week, described in ranges of worst through to best outcomes, refer [http://en.wikipedia.org/wiki/Goal\\_Attainment\\_Scaling](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.



Also look in the test subdirectory for SoapUI projects that showcases some of the BRMS REST knowledge base management APIs.

## 5.4 R using RStudio Samples

R script based samples showing how to interact with the Weight Watcher Services are also provided under the /tools/RStudio directory. Inspect the supplied .R script and edit the end point according to your environment.



The screenshot shows the RStudio interface with a script file named 'weightwatcher.R'. The script defines a URL, constructs an XML request, and sends it via a curl command. The console output shows the XML request being sent and the HTTP response received.

```
1 Sys.setenv("NOANT" = "true")
2 library("httr")
3 library("rjson")
4 library("Rcurl")
5
6 #url <- "http://localhost:8080"
7 url <- "http://192.168.59.103:8080"
8 #url <- "http://54.153.151.37:8080"
9
10 url <- paste(url, "/kie-server/services/rest/server/containers/watch", sep = "")
11 print(url)
12
13 request <- "
14 <kie-container>
15 <container-id>watch</container-id>
16 <status>
17 <release-id>
18 <group-id>com.redhat.demos</group-id>
19 <artifact-id>weightwatchers</artifact-id> <version>1.0</version>
20 </release-id>
21 <resolved-release-id>
22 </kie-container>
23 "
```

```
>> <factjson>{ "userid" : 2, "obsdate" : "2015-04-22 07:15:00 EST", "obsname" : "weight", "obsvalue" : 68 }</factjson>
>> <facttxt>Incoming Observation JSON</facttxt>
>> </com.redhat.weightwatcher.fact>
>>
>> </insert-elements>
>> </fire-all-rules>
>> <query out-identifier="Watch" name="getWatch">
>> </batch-execution>

<- HTTP/1.1 200 OK
<- Server: Apache-Coyote/1.1
<- Content-Type: application/xml
<- Content-Length: 5761
<- Date: Tue, 21 Jul 2015 11:39:21 GMT
<-
>
```

The Environment pane on the right shows the following values:

header	
request	<batch-execution lookup="wetc...
response	List of 10
url	"http://192.168.59.103:8080/k...

## 5.5 Changing Rules

You can also experiment with changing and creating rules and observing their impact. This is not available in the Docker container version as the business-central application is not included in that deployment in the interests of a lean profile. 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
```

## 5.6 High Availability

A high-availability and load balancing demonstration is also available. 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. There are two nginx build options as described below.

### 5.6.1 Registry Pull Option

Note that availability of this container is not guaranteed so if the pull fails follow the alternate build option that follows. To use this container we need to make a small configuration change first, commit the change then run the docker container. Once completed, move on to the steps to launch multiple weightwatchers containers.

```
$ docker pull spicozzi/nginx

# Find the <IP> addresses of your weighwatcher containers
$ boot2docker ip

# Now check whether we need to update the nginx configuration file
# Run the container in foreground
$ docker run -it -p 80:80 --name nginx spicozzi/nginx /bin/bash

$ root@123456789:/# sudo vi /etc/nginx/sites-enabled/default
# Change the IP address of the upstream entries
# to reflect your environment

# From another terminal get the <CONTAINER_ID> and commit the changes
$ docker ps -l
$ docker commit <CONTAINER_ID> spicozzi/nginx

# Now go back to the interactive nginx container and kill it
$ root@123456789:/# Ctrl-D
$ docker rm <CONTAINER_ID>

# Now run the nginx container in the background
$ docker run -d -p 80:80 --name mynginx spicozzi/nginx \
nginx -g "daemon off;"

# Test the nginx container
$ curl http://<IP>
```

### 5.4.1 Local Build Option

To build your own nginx container, first 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 nginx/default # Change hostnames as appropriate
$ docker build -t spicozzi/nginx .
$ docker run -d -p 80:80 --name nginx spicozzi/nginx \
nginx -g "daemon off;"
$ boot2docker ip
$ curl http://<IP>

```

## 5.4.2 Tests

Once you have built the nginx container run the tests as follow. The nginx instance will proxy URIs with /kie-server/ to an upstream pair of docker BRMS server containers listening on 8080 and 8081 respectively. Note that this will require a workstation with more than 8 GBytes RAM. These instances could be traditional or container based deployments. For container based images, you would launch a pair of servers like below. Container deployments are described in section 6.

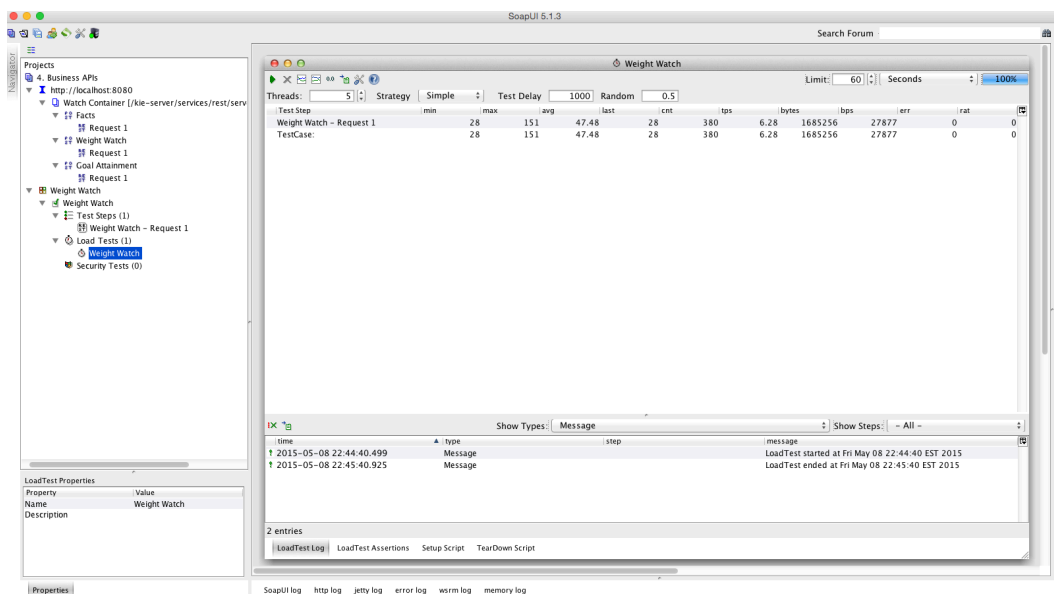
```

$ cd <path-to-project>
$ docker run -it -p 8080:8080 -p 9090:9090 --name weightwatcher1 \
  spicozzi/weightwatcher
$ docker run -it -p 8081:8080 -p 9091:9090 --name weightwatcher2 \
  spicozzi/weightwatcher

```

Note that if you edit/run the container/heartbeat.sh script in a separate window, then there is no need to restart the KIE server container each time you restart these JBoss containers.

Once the docker containers are running, 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 /tools. Remember to ensure you have started the Container in each BRMS instance using the supplied REST API.





## 5.7 Rule Changes with Docker Container

This example shows how to change BRMS artefacts used by a Docker container based JBoss BRMS instance. Details for how to build a traditional instance are described in Section 5 and for Docker in Section 8. The steps essentially consist of:

1. Launch a Docker container JBoss BRMS instance using the supplied `docker-run-with-volume-attached.sh` script
2. Launch a traditional JBoss BRMS deployment and then connect to the business-central console
3. Make whatever changes you require to your rule and/or data model artefacts. For examples refer to the Changing Rules section
4. Build and deploy those changes from the business-central GUI
5. Copy across the updated contents in `$M2_HOME` to the `<path-to-project>/m2` directory
6. Use the supplied test scripts against the Docker container instance to verify the changes

How does this work? Look inside the referenced Docker run script and you will note that the `$M2_HOME` directory used by the Docker container now references a local file system location. This allows you to change BRMS rule artefacts outside the Container as per the instructions above.

## 6 Docker Containers

This section describes instructions for setting up the demonstration using a Docker based container. Two options are provided 1) a direct pull from the registry or 2) a manual build. The Registry option is not guaranteed as from time to time, the image may be removed for storage/capacity reasons. Mac OS/X users may want to install boot2docker first.

### 6.1 Registry Pull Option

To pull down and run a Docker container-based image, do as follows. Once the image is running, be sure to create a KIE Server Container, as described in Section 4.3. Once this is done, you may execute the business case tests.

```
$ docker pull spicozzi/weightwatcher
$ docker run -it -p 8080:8080 -p 9090:9090 spicozzi/weightwatcher
# Take note of your <IP> address using e.g. boot2docker ip
# Now create a KIE Server Container as described in Section 4.3
```

### 6.2 Local Build Option

To create your own image from scratch, first complete the traditional installation described earlier. The steps required are summarised as:

1. Stop the jboss-eap-6.4 you started earlier
2. Copy across the contents of \$M2\_HOME to /m2
3. Remove the business-central war file for a lighter weight server profile
4. Zip the jboss-eap-6.4 installation located under \target you built earlier
5. Move the zip file to the \installs directory
6. Run the docker build script
7. Run the docker container
8. Locate the IP address used by this docker container
9. Make the necessary configuration changes using workbench and SoapUI
10. Bake the changes into the docker image
11. Restart the container

```
$ cd <path-to-project>
$ cd target
# Stop the running jboss server
$ rm -rf jboss-eap-6.4/standalone/deployments/business-central*
# Find the location of your M2_HOME
$ cd $M2_HOME
$ cp -r .m2 <path-to-project>/m2
$ zip -r jboss-eap-6.4.zip jboss-eap-6.4/
$ mv jboss-eap-6.4.zip ../installs/
$ cd ../
$ docker build -t spicozzi/weightwatcher .
# For Mac OS/X find IP address using
$ boot2docker ip
$ docker run -it -p 8080:8080 -p 9090:9090 spicozzi/weightwatcher
```

Now repeat the step to create a Container as described in section 4.3. And then finally commit these changes.

```
$ docker ps -l
# Look for <PID> of your running weightwatcher container
$ docker commit <PID> spicozzi/weightwatcher
# Stop the weightwatcher container running and then restart
$ docker run -it -p 8080:8080 -p 9090:9090 spicozzi/weightwatcher
```

You are then ready to explore the demo as described in section 5. The only thing you will need to do is restart the Container as described in section 5.1.

Note that if you edit/run the container/heartbeat.sh script in a separate window, then there is no need to restart the KIE server container (section 4.3) each time you restart JBoss. This script and its function will be replaced by a container instance at some later stage.

## 7 OpenShift V3.0 GA

OpenShift V3 can load and run an arbitrary Docker image. The following instructions assume you have a working OpenShift V3 installation. We are going to create two OpenShift applications, one for the weightwatcher itself and another to host RStudio to act as a client. Deploy. Enjoy!

### 7.1 OpenShift Environment Setup

```
# Pull down the image
$ su - root

# Set appropriate OpenShift security context constraints
$ oc edit scc restricted
> change runAsUser.Type to RunAsAny

# Create project that ensure pods land in certain nodes
$ oadm new-project weightwatcher --node-selector='region=primary'
$ oadm policy add-role-to-user admin <DEVUSER> -n weightwatcher

# Pull down my docker images
$ docker pull spicozzi/weightwatcher
$ docker pull spicozzi/rstudio
```

## 7.2 Create weightwatcher OpenShift Application

```
# For instructions below replace <PARAMETER> to reflect your environment

# Login as an OpenShift V3.0 user <DEVUSER>
$ su - <DEVUSER>

# Authenticate to the OpenShift master at <MASTER-FQDN>:<PORT>
$ oc login -u <DEVUSER> --insecure-skip-tls-verify --
server=https://<MASTER-FQDN>:<PORT>

# Switch to the weightwatcher project
$ oc get projects
$ oc project weightwatcher

# Create an OpenShift application
$ oc new-app spicozzi/weightwatcher
$ oc expose service weightwatcher --name=weightwatcher-route --
hostname=weightwatcher.<CLOUDAPPS-FQDN>
$ oc describe pod weightwatcher

# To scale up replicas
$ oc scale dc weightwatcher --replicas=2

# Tail the log file for the created pod
$ oc get pods
$ oc logs -f <PODNAME>

# If you make any errors just delete the <PROJECT> and repeat
$ oc delete project <PROJECT>
```

## 7.3 Create RStudio OpenShift Application

```
# For instructions below replace <PARAMETER> to reflect your environment

# Login as an OpenShift V3.0 user <DEVUSER>
$ su - <DEVUSER>

# Authenticate to the OpenShift master at <MASTER-FQDN>:<PORT>
$ oc login -u <DEVUSER> --insecure-skip-tls-verify --
server=https://<MASTER-FQDN>:<PORT>

$ oc project <PROJECT>

# Create an OpenShift application
$ oc new-app spicozzi/rstudio
$ oc expose service rstudio --name=rstudio-route --
hostname=rstudio.<CLOUDAPPS-FQDN>
```

## 7.4 Running the RStudio Sample

Point your browser to `rstudio.<CLOUDAPPS-FQDN>` to open the RStudio application and then login using `guest/guest` as credentials. Using the RStudio File Menu, navigate down to the `weightwatcher.R` sample R. Change the `url` variable in the script to reflect your `weightwatcher.<CLOUDAPPS-FQDN>`. Click the Source command to run the script.

## 7.5 OpenShift Console

Note that once you have created the OpenShift application using the CLI as described above, you can also login to the OpenShift Console to inspect. To do this point your browser to `<MASTER-FQN>:<PORT>`, login as the `<DEVUSER>` and then select the `<PROJECT>` that hosts the `weightwatcher` application.

## 7.6 Create Website OpenShift Application

Alternatively, you can pull down the companion website known as `testdrive` site instead.

```
# For instructions below replace <PARAMETER> to reflect your environment

# Login as an OpenShift V3.0 user <DEVUSER>
$ su - <DEVUSER>

# Authenticate to the OpenShift master at <MASTER-FQDN>:<PORT>
$ oc login -u <DEVUSER> --insecure-skip-tls-verify --
server=https://<MASTER-FQDN>:<PORT>

$ oc project <PROJECT>

# Create an OpenShift application
$ oc new-app spicozzi/testdrive
$ oc expose service testdrive --name=testdrive-route --
hostname=testdrive.<CLOUDAPPS-FQDN>

# To scale up replicas
$ oc scale dc testdrive --replicas=2
```

## A Appendix

### A.1 Useful boot2docker Docker Commands

```
# Find <IP> of boot2docker virtual machine
$ boot2docker ip

# If you encounter strange problems while pull/push of images
$ boot2docker stop
$ boot2docker start

# List of docker images
$ docker images

# List of running docker containers showing <CONTAINER_ID>
$ docker ps -l

$ docker attach <CONTAINER_ID>

# Kill a running docker container with <CONTAINER_ID>
$ docker rm -f <CONTAINER_ID>

# Pull down a docker image
$ docker pull spicozzi/nginx

# Assume you have a running container named weightwatcher1
$ docker logs -f weightwatcher1
$ docker rm -f weightwatcher1

# Remove all running containers
$ docker rm -f $(docker ps -aq)

# Remove all untagged images
$ docker rm -f $(docker images | grep "^<none>" | awk "{print $3}")
```