



OPENRULES[®]

Release 6.2.2

TUTORIAL
**“Cloud Application
Development”**

OpenRules, Inc.

www.openrules.com

October-2012

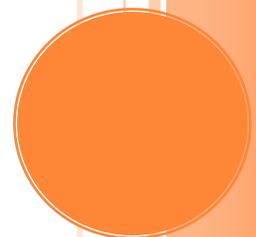


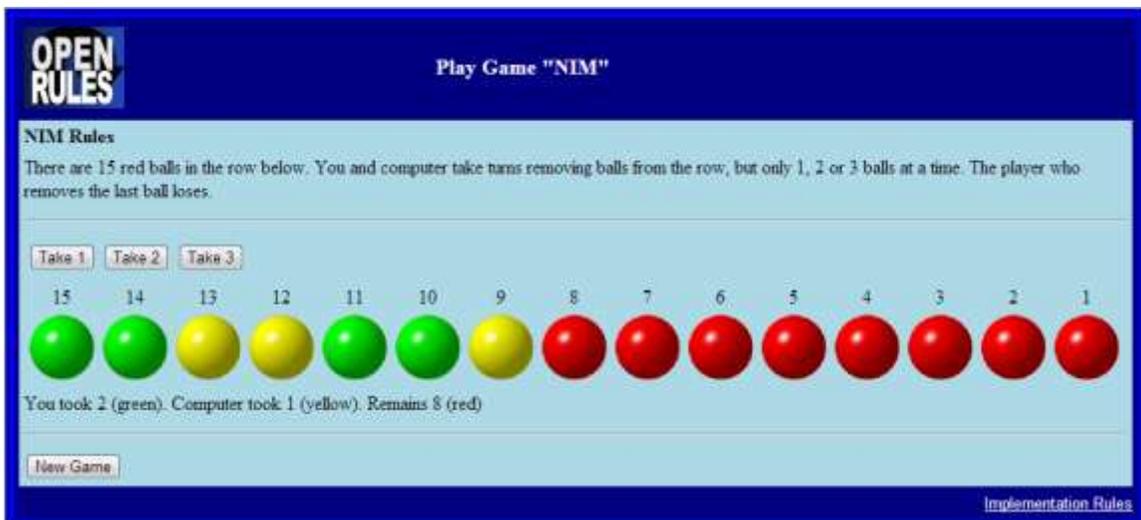
Table of Contents

<i>Introduction</i>	3
<i>Creating Web Application</i>	4
Game Rules	4
Business Logic	4
Implementation	6
<i>Graphical Interface</i>	7
Game Layout	7
External Layout	9
Processing Flow Rules	9
<i>Project Structure</i>	10
<i>Project Deployment</i>	12
Deploying on Local Tomcat	12
Deploying on Jelastic Cloud	12
Setting Cloud Environment	12
Uploading Wars.....	13
Deploying.....	14
Running	15
<i>Technical Support</i>	15

INTRODUCTION

OpenRules® is as an open source Business Decision Management System (BDMS) with proven records of delivering and maintaining reliable decision support software. The detailed description of OpenRules® can be found in the User Manual.

This tutorial describes how to create a rules-based web application using OpenRules Forms or Dialog and to deploy it on the cloud using www.jelastic.com. As an example, we selected a popular game “Nim” that you may play from http://openrules.jelastic.servint.net/Nim. Here is the graphical interface for the game:



The complete OpenRules® installation includes a workspace “openrules.cloud” that has several projects to be deployed on the cloud. The project described in this tutorial is called “Nim”.

CREATING WEB APPLICATION

Game Rules

We We want to implement the popular game known as “Nim”. The rules of the game are simple:

There is a row with 15 red balls. You and computer take turns removing balls from the row, but only 1, 2, Or 3 balls at a time. The player who removes the last ball loses the game.

We will create a graphical user interface, implement the logic according to which computer will play the game, and will deploy the game on the Jelastic cloud.

Business Logic

We want use rules tables to define the logic according to which a computer will play the game (respond to the user’s moves). Here is the strategy for the computer:

Rules String nimRules(Nim nim, int opponentTook)			
IF Number of Remaining Objects	THEN Take Objects with Numbers		
15	15	14	
14	14		
13	13	12	
12	12	11	10
11	11	10	
10	10		
9	9		
8	8	7	6
7	7	6	
6	6		
5	5		
4	4	3	2
3	3	2	
2	2		
1	1		

These rules specify which balls should be taken by a computer based on the number of remaining balls. For example, if there are 11 balls left, it tells computer to take two balls with numbers “11” and “10”.

To support a user communication on each turn we will use the following rules after the first table defines “computerTook” number:

Rules String nimAfterMyMove(Nim nim, int opponentTook, int computerTook)		
IF Number of Remaining Objects	THEN Set Initial Size	AND Display Message
0	15	You WON. Congratulations!
1	15	:= "You took " + opponentTook + " (green). Computer took " + computerTook + " (yellow). You LOST. Try again!"
		:= "You took " + opponentTook + " (green). Computer took " + computerTook + " (yellow). Remains " + nim.size + " (red)"

So, if no balls remain, it means a computer took the last ball and we will display “You WON. Congratulations!”.

If only 1 balls left, we will inform a user about his/her loss.

In all other cases, we will inform a user how many balls we took.

These rules table have parameters:

- Nim nim – a placeholder for the current state of the game implemented as a Java class Nim
- int opponentTook – a number of balls taken by opponent
- int computerTook – a number of balls taken by computer.

Implementation

We created one Java class in the package “openrules.nim” within the folder “src”:

```
public class Nim {  
  
    OpenRulesEngine engine;  
    String problem;  
    int size;  
    String message;  
    Object[] objects;  
  
    public Nim(OpenRulesEngine engine) {  
        this.engine = engine;  
        problem = "Nim";  
    }  
  
    public void init(int size) {  
        this.size = size;  
        objects = new Object[size];  
        for(int i=0; i<size; i++)  
            objects[i] = null;  
    }  
    // We omitted public getters and setters  
}
```

An instance of this class should be created for each user’s session.

Implementation of the rules table “nimRules” is described in the technical rows that were hidden on the previous view:

Rules String nimRules(Nim nim, int opponentTook)	
C1	A1
nim.size - opponentTook == n	for(int i=0; i<numbers.length;i++) nim.objects[numbers[i]-1] = "my"; // taken by computers nim.size = nim.size - opponentTook - numbers.length; return nimAfterMyMove(nim,opponentTook,numbers.length);
int n	int[] numbers
IF Number of Remaining Objects	THEN Take Objects with Numbers

Knowing “opponentTook” number, we mark all balls taken by us as “my” and subtract “opponentTook” from “nim.size”. Then we simply call rules table “nimAfterMyMove” that has the following technical view:

Rules String nimAfterMyMove(Nim nim, int opponentTook, int computerTook)		
C1	A1	A2
nim.size == n	nim.init(newSize);	nim.setMessage(message); return message;
int n	int newSize	String message
IF Number of Remaining Objects	THEN Set Initial Size	AND Display Message

GRAPHICAL INTERFACE

We use OpenRules Forms to create the graphical interface described [above](#).

Game Layout

The main layout of the GUI is described at the following Excel tables:

Layout TableLayout nimLayout(Nim nim)
NIM Rules
"There are 15 red balls in the row below. You and computer take turns removing balls from the row, but only 1, 2 or 3 balls at a time. The player who removes the last ball loses."
<hr/>
nimActions()
nimRow15(nim);
nim.message
<hr/>
actionButton("New Game")

The layout “nimRow15” display 15 balls:

Layout TableLayout nimRow15(Nim nim)	"15"	"14"	"13"	"12"	"11"	"10"	"9"	"8"	"7"	"6"	"5"	"4"	"3"	"2"	"1"
	obj(nim,15)	obj(nim,14)	obj(nim,13)	obj(nim,12)	obj(nim,11)	obj(nim,10)	obj(nim,9)	obj(nim,8)	obj(nim,7)	obj(nim,6)	obj(nim,5)	obj(nim,4)	obj(nim,3)	obj(nim,2)	obj(nim,1)

Here the method “obj” is defined as:

Method TableLayout obj(Nim nim,int n)
String object = "";
if (n > nim.size) {
object = "";
if (nim.objects[n-1] != null)
object = "";
}
object;

It simply selects red (not taken yet), green (taken by a user), or yellow (taken by a computer) image for every ball to be displayed.

And finally action buttons are created using this layout:

Layout TableLayout nimActions()		
actionButton ("Take 1")	actionButton ("Take 2")	actionButton ("Take 3")

External Layout

We use the following Excel layouts to describe external view of the game:

Layout TableLayout mainLayout()	
titleBarLayout()	
mainPageLayout()	
Implementation Rules	
Layout TableLayout titleBarLayout()	
 	Play Game "NIM"
Layout TableLayout mainPageLayout()	
nimLayout(dialog().get("nim"))	

These layouts have properties that defined selected colors and described in the hidden rows similar to these ones:

Layout TableLayout mainLayout()		
properties	width	100%
	cellspacing	2
	style	background-color: Navy; border: 8px groove
	cellpadding	2

Processing Flow Rules

We define interaction logic using the following rules:

Rules void nimFlowRules(Nim nim)		
C1	A1	A2
dialog().isAction(action);	out(msg);	dialog().nextLayout = layout;
String action	String msg	TableLayout layout
IF Action is	THEN Execute Rules	AND Set Next Layout
Init	{ newGame(nim); }	{ nimLayout(nim); }
Take 1	{ nimRules(nim,1); }	
Take 2	{ nimRules(nim,2); }	
Take 3	{ nimRules(nim,3); }	
New Game	{ newGame(nim); }	

And finally on each client-server interaction the following main-method will be called:

```
Method TableLayout main(Dialog d)
Nim nim = (Nim) dialog().get("nim");
nimFlowRules(nim);
if (dialog().errors == 0)
    dialog().status = "";
return mainLayout();

Method void newGame(Nim nim)
out("NEW GAME");
if (nim != null) {
    nim.init(15);
    nim.setMessage("Take 1, 2, or 3 objects");
}
```

The method “newGame” will be called when a dialog is initialized and when a user clicks on “New Game”.

PROJECT STRUCTURE

Project Nim has a typical structure for all OpenRules web applications:

- src
 - openrules.nim
 - Nim.java
- war
 - css – stylesheets
 - images – used images
 - rules
 - Main.xls
 - WEB-INF
 - classes
 - log4j.properties
 - web.xml

- build.properties
- build.xml – used by Ant to compile the project and to build a war file
- build.bat – creates a war-file Nim.war
- index.jsp

The file "index.jsp" is the main entry point for this JSP application. It is the standard "index.jsp" file with one difference – when a new session starts we create a new instance on the class Nim and put it into the dialog under the name "nim". Here is the complete text of "index.jsp":

```

<%@ page import="com.openrules.forms.gui.jsp.*" %>
<%@ page import="com.openrules.forms.*" %>
<%@ page import="com.openrules.ruleengine.*" %>
<%@ page contentType="text/html; charset=UTF-8" %>
<%@ page import="openrules.nim.Nim" %>

<%
    volatile OpenRulesEngine engine = null;
%>

<%
    String name = request.getParameter("name");
    if (name==null)
        name = "Nim";
    String xlsMain = "file:../webapps/" + name + "/rules/Main.xls";
    String sessionAttribute = "openrulesSession";
    OpenRulesSession openrulesSession = (OpenRulesSession) session.getAttribute(sessionAttribute);
    if (openrulesSession == null )
    {
        System.out.println("NEW SESSION");
        engine = (OpenRulesEngine) application.getAttribute("OpenRulesEngine");
        OpenRulesEngine sessionEngine = engine; // this local variable seems unnecessary, but for some JVMs, it makes the code faster
        if (sessionEngine == null) {
            synchronized(this) {
                sessionEngine = engine;
                if (sessionEngine == null) {
                    openrulesSession = new OpenRulesSession(xlsMain); // create a new engine
                    System.out.println("NEW ENGINE");
                    engine = openrulesSession.getOpenRulesEngine();
                    application.setAttribute("OpenRulesEngine",engine);
                }
            }
        }
        else {
            openrulesSession = new OpenRulesSession(engine); // reuse previously created "engine"
        }
        session.setAttribute( sessionAttribute, openrulesSession);
        Dialog dialog = openrulesSession.getDialog();
        dialog.setName(name);
        System.out.println("NEW NIM");
        Nim nim = new Nim(engine);
        dialog.put("nim",nim);
    }
%>

<HTML>
<HEAD><TITLE>OpenRules</TITLE></HEAD>
<body>
<%
    Dialog dialog = openrulesSession.getDialog();
    System.out.println("***** PROCESS REQUEST for " + dialog.getName());
    openrulesSession.processRequest(session, request, out);
%>
</body>
</HTML>

```

PROJECT DEPLOYMENT

The same project can be deployed on the local Tomcat or on the cloud. Our application depends on the standard OpenRules library called “openrulesdialog”. This library should always be deployed first.

Deploying on Local Tomcat

The project Nim can be deployed and tested on your local Tomcat by simply moving the generated file “Nim.war” to your Tomcat’s webapps directory. Make sure that you also moved the file “openrules.dialog/openrulesdialog.war” to your Tomcat’s webapps directory. After starting Tomcat you may use your browser with the URL <http://localhost:800/Nim>. If you receive an error “File ...Main.xls” cannot be found, please change the line

```
String xlsMain = "file:../webapps/" + name + "/rules/Main.xls";
```

in your “index.jsp” to

```
String xlsMain = "file:../webapps/" + name + "/rules/Main.xls";
```

When you think you have your web application working fine, switch to the cloud deployment.

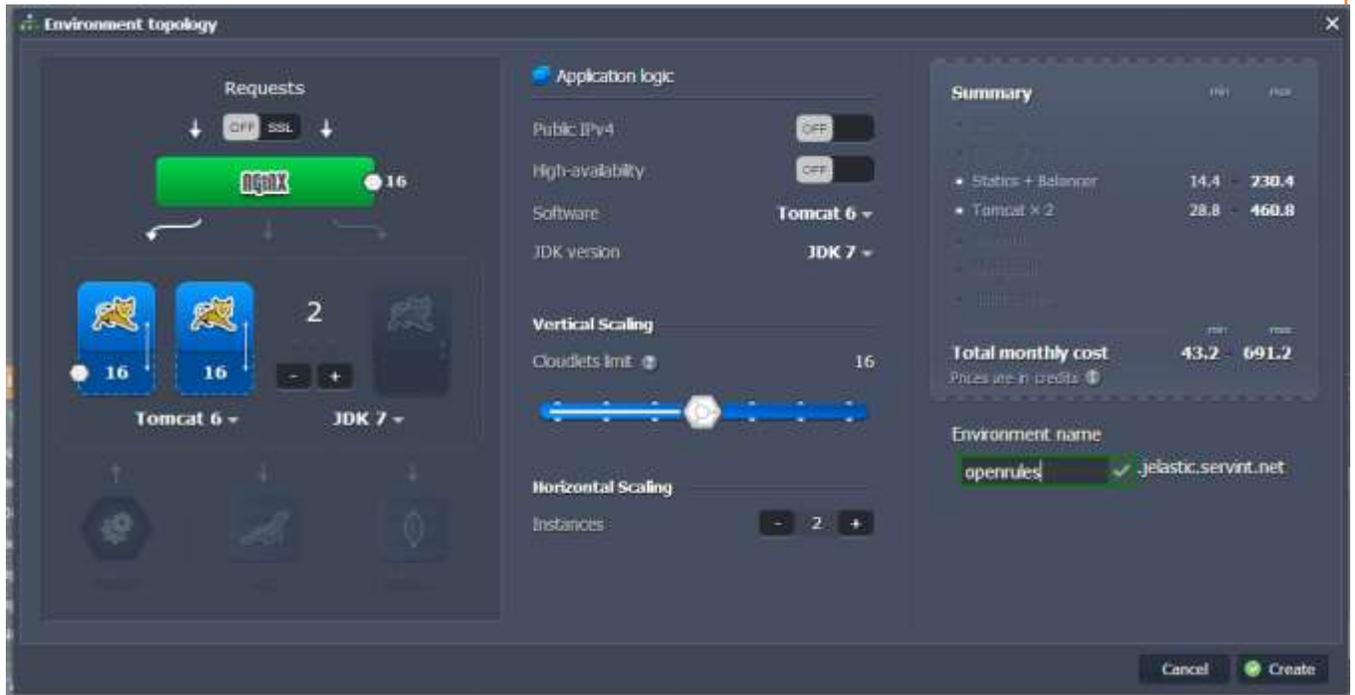
Deploying on Jelastic Cloud

We will use the Jelastic cloud (www.jelastic.com) that provides a very simple interface for Java web application that does not require any changes in when you move from local Tomcat to a cloud-based Tomcat. The only real difference if that currently Jelastic does not allow special characters like dots inside names of the Tomcat’s projects. We selected Jelastic as a winner of the latest Open World [Duke’s Choice Technology Award](#).

Setting Cloud Environment

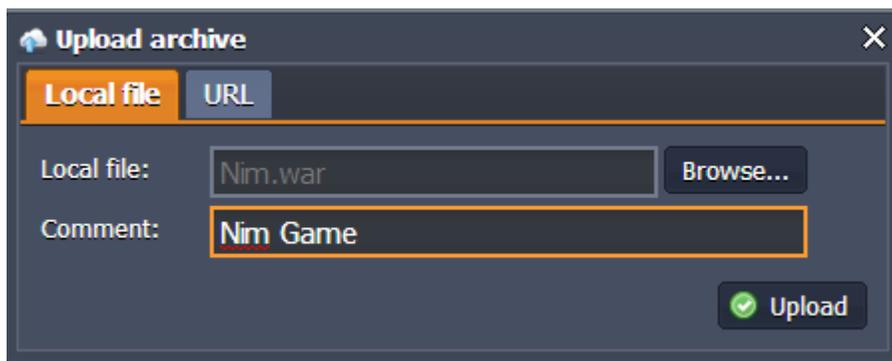
The website www.jelastic.com hat provides a very simple interface. First you may create a free evaluation account using one of many cloud hosts (e.g. US-based host “ServInt”). It gives you a web access to a quite intuitive Jelastic’s GUI from which you may create an instance of the Cloud Environment by choosing

from many predefined components. For example, we selected two Tomcat containers and one JDK 7 and call this environment “openrules”. You always may modify the environment later and Jelastic will take about load balancing and other cloud administration issues (if any). Here is a sample view:



Uploading Wars

Use the Deployment Manager to upload war-files from your machine to the cloud. From the Jelastic’s GUI click on the button “Upload” and “Browse...” to select and upload your local war-files “openrulesdialog.war” and “Nim.war”.



Deploying

To deploy these wars to my “openrules” cloud environment click on the “Deploy” button and select “openrules”.

Name		Comment
Nim.war	 	Nim Game
openrulesdialog.war	 guess	ules Dialog Support
Dialog1040EZCloud.war	 openrules	1040EZCloud

Deploy to openrules

Context

Select one of the existing contexts or type new

Deploy to openrules

Context

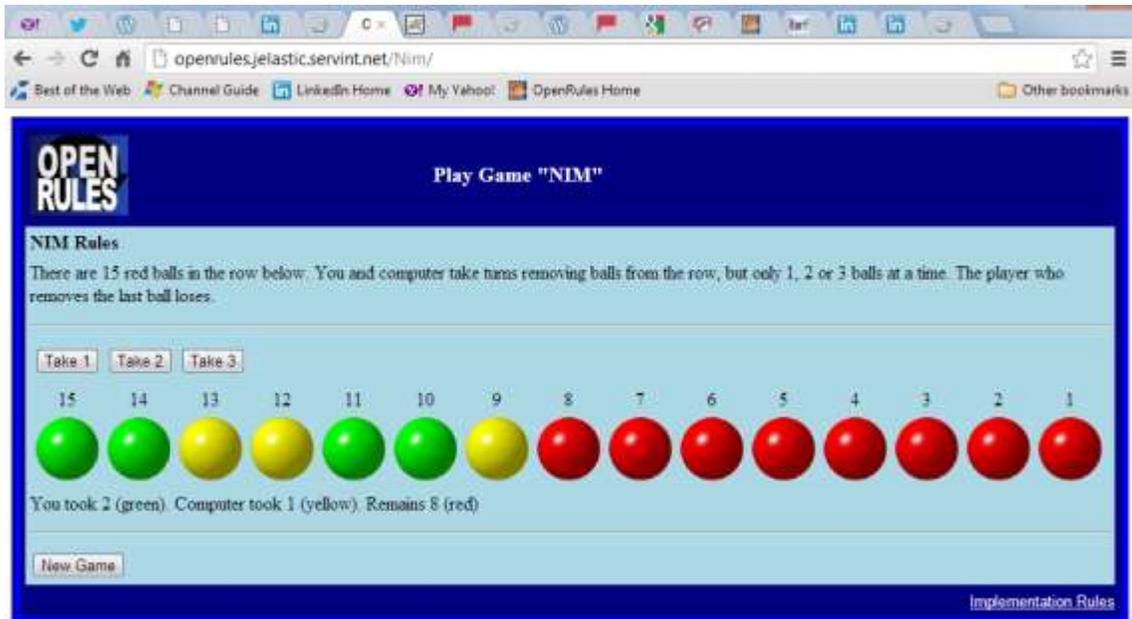
Select one of the existing contexts or type new

After the deployment, you will a button that opens a browser with this URL <http://openrules.jelastic.servint.net/Nim/>.



Running

Then you will see your application running on the cloud:



You may start it at the same time from another browser at the same time using the same URL <http://openrules.jelastic.servint.net/Nim/>. All instances of the application will run in parallel without interference. A nice thing about this particular implementation is the fact that the same instance of OpenRuleEngine serves different client sessions minimizing memory requirements on the server.

TECHNICAL SUPPORT

Direct all your technical questions to support@openrules.com or to this [Discussion Group](#).