BPM Voices: Using business rules in the cloud to solve Sudoku, Part 2: Deploying the rules application to the cloud

Rajesh Rao
Decision Management Solution Architect
IBM

Skill Level: Introductory

Date: 26 Sep 2012

In Part 1, I presented a hybrid approach to solving Sudoku puzzles - an approach that combines business rules built using WebSphere Operational Decision Management (ODM) with a heuristic depth-first search algorithm. In this column, I'll focus on executing the rule application on the cloud. Can a web application leveraging a rule engine be deployed to a public cloud? If so, which rule integration pattern would we use? Which ODM components would we deploy? What level of execution performance can we expect? What challenges await us? These are the questions that this column seeks to answer for two widely available cloud offerings: Google App Engine™ and Amazon Web Services™.

View more content in this series

Introduction

This column describes a personal journey of discovery with integrating two leading edge technologies – cloud computing and business rules management systems. Both of these technologies cater to the inexorable forces of change: changes to infrastructure needs and changes to decision logic. As a long time practitioner of business rules management systems (BRMS), I was very curious to see what challenges need to be addressed when developing and deploying a rule-based application on the cloud. The Sudoku ruleset described in Part 1 is deployed to the cloud with the intent of proving the viability of using rules on the cloud, and with a particular eye towards understanding the deployment and execution constraints imposed by the cloud. Other areas such as rule maintenance and licensing are not covered in this article.

Cloud computing - the deployment of network-based applications in a scalable, shared IT environment - is all the rage in today's IT landscape because it offers a number of advantages over more traditional application deployment models. These

advantages include massive scalability, near-immediate availability and provisioning, and improved cost management controls.

Cloud computing represents a fundamental shift not only for IT administrators, but also architects and developers. From a development perspective, particularly interesting is Platform as a Service (PaaS), which delivers development environments as a service. In the PaaS model, cloud providers deliver a computing platform, which typically includes an operating system, programming language execution environment, database, and web server. These services are free or offered on a pay-per-use basis. Application developers can develop and run their software solutions on a cloud platform without the cost and complexity of buying and managing the underlying hardware and software layers. Additionally, in many PaaS offerings the underlying resources scale automatically to meet application demand. Examples of PaaS include: Amazon™ Elastic Beanstalk, Google App Engine, Microsoft® Azure and IBM SmartCloud® Application Services.

Applications built using the PaaS model are then run on the provider's infrastructure and are delivered to the general public via the internet from the provider's servers. This deployment model is termed the public cloud. Typically, the services available on the public cloud are constrained by the vendor's design and capabilities. Considering these restrictions, I was curious to see whether rule applications can be developed and deployed at all using them. What components of WebSphere ODM would we use? How should the rule applications be packaged? These are the questions I set out to answer when building the application and writing this column. Here, we explore deployment of a web application on two popular public cloud services with a free daily quota: Google App Engine and Amazon Web Services (AWS).

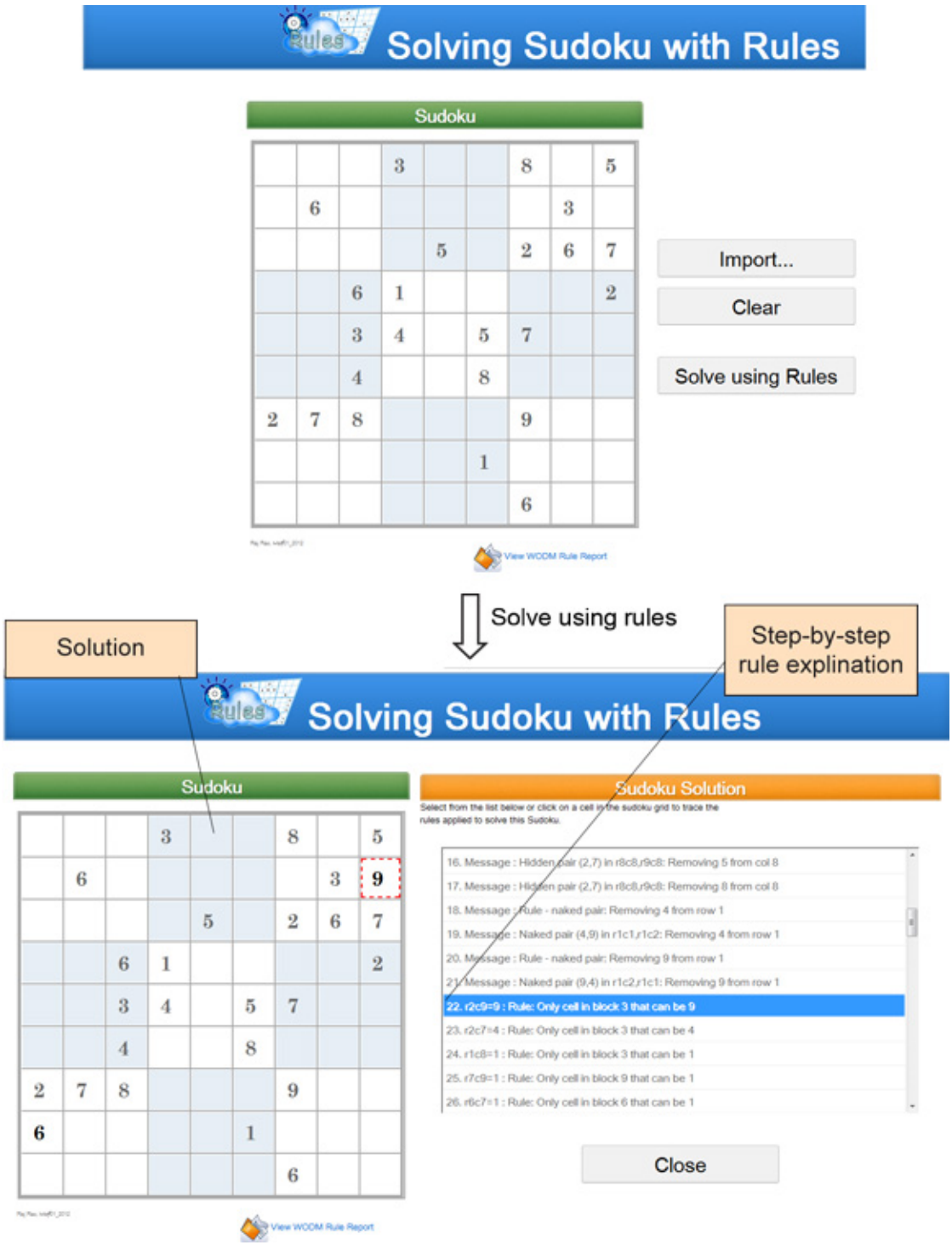
The Sudoku web application

The Sudoku web application provides an interface for users to enter Sudoku puzzles and invoke the rule engine on the server. The goals of the Sudoku web application are straightforward:

- The application should be available to the general public over the internet.
- The application should employ a rule engine (using the ruleset described in Part 1) in the back end to solve Sudoku.
- The application should be user-friendly. It should use an interactive AJAX-based front end to capture the Sudoku configuration and to display the solution along with step-by-step explanations.

Figure 1 shows the web application, which you can find here. The top part of the illustration shows the screen that allows a user to enter Sudoku data or import it as a string. The bottom part illustrates the solution screen that results from clicking **Solve using Rules**. The list on the right depicts the step-by-step reasoning used by the rule engine. Clicking on any of the list items highlights the cell corresponding to the reasoning step and displays the Sudoku solution derived thus far.

Figure 1. Sudoku web application



The web application was built using Google Web Toolkit™ (GWT), which is a development toolkit that is used to build complex browser-based applications. The GWT SDK provides a set of core Java APIs and widgets. These allow a developer to write AJAX applications in Java and then compile the source to highly optimized JavaScript that runs across all browsers, including mobile browsers for

Android™ and the iPhone®. You can find an introduction to the GWT SDK, along with instructions for its installation and usage within Eclipse, here. I found that using the Google Plugin for Eclipse makes development of GWT apps very enjoyable.

Using GWT, all coding is done using Java, HTML and CSS. The application code is compartmentalized into server code, client code and shared code. GWT compiles the client-side Java code and shared Java code into JavaScript, which can run on any web browser. The client-side JavaScript communicates with the server using the GWT Remote Procedure Calls (RPC) framework, which allows the client and server components of the web application to exchange Java objects over HTTP. The server-side code that gets invoked from the client is referred to as a service, and is implemented using the Java servlet architecture. Within the client code, GWT automatically generates a proxy class to make calls to the service and handles serialization of the Java objects passing back and forth between the client and the service.

Deploying an application built using GWT to a web server is straightforward. All you need to do is copy the client-side HTML, CSS and JavaScript files generated by GWT to the web server. The server-side code is also easily deployed to a servlet container since the GWT compiler creates output in a standard WAR directory structure.

Rule integration

The client code is typical GWT fare, but of more interest is the service definition that invokes the rule engine on the server side. This service definition takes a string representation of the Sudoku grid as an argument and returns a Sudoku object that contains the solution grid along with a step-by-step explanation of the heuristics used to arrive at the solution. The following listing shows the service definition.

```
/**
 * The client side stub for the RPC service.
 */
@RemoteServiceRelativePath("sudoku")
public interface SudokuService extends RemoteService {
 Sudoku solveSudokuUsingRules(String sudokuDataString) throws
IllegalArgumentException;
}
```

The server-side implementation of this service invokes the SudokuSolver, which, as described in Part 1, uses a hybrid approach combining WebSphere ODM and depth-first search to arrive at the solution. The server-side implementation is show below. The Sudoku solver and the rule engine are invoked in line 16.

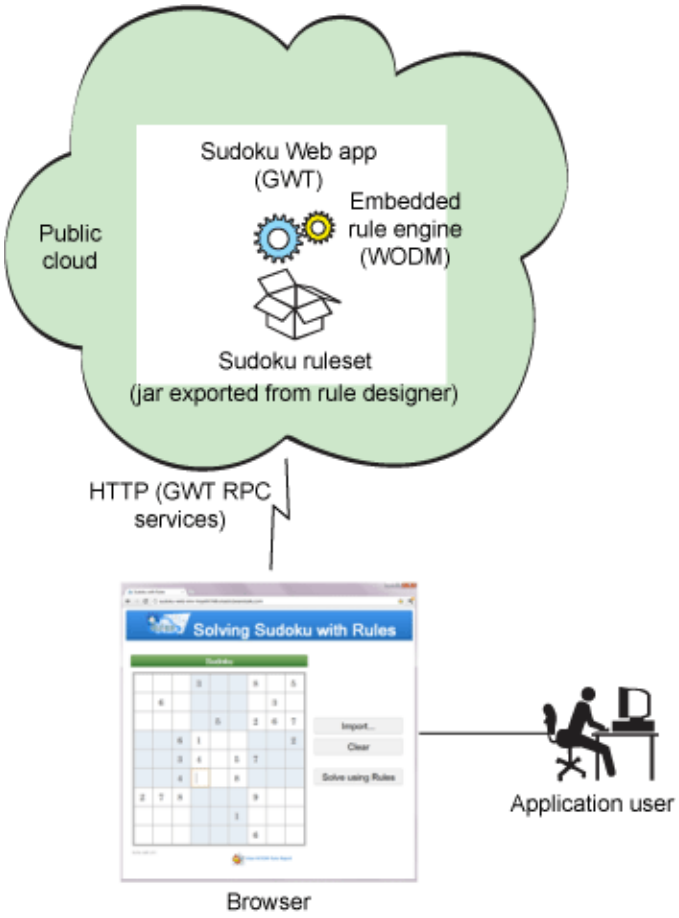
```
1./**
2. * The server side implementation of the RPC service.
3. */
4.public class SudokuServiceImpl extends RemoteServiceServlet implements
5.  SudokuService {
6.
7. public Sudoku solveSudokuUsingRules(String sudokuDataString) throws
```

```
   IllegalArgumentException {
8.   // Verify that the input is valid.
9.   String errorMsg = FieldVerifier.isValidData(sudokuDataString);
10.   if (errorMsg != null) {
11.       throw new IllegalArgumentException(errorMsg);
12.   }
13.
14.   SudokuWrapperNode solution = null;
15.   try {
16.       solution = (new
     SudokuSolver(getServletContext())).solve(sudokuDataString);
17.   } catch (Exception e) {
18.   throw new IllegalArgumentException(e);
19.   }
20.
21.   Sudoku sudoku = null;
22.   if (solution != null) {
23.      sudoku = solution.getSudoku();
24.   }
25.   return sudoku;
26.  }
```

Note that the `Sudoku` class and all its associated classes, including all the Java classes comprising the execution object model (XOM), must be placed in the shared code compartment in the GWT application because they are used on the client side as well as on the server side. Additionally, all these classes are tagged to be serializable to enable them to be transported between the server and the client.

WebSphere ODM provides many different options for integration rules into an application. The most versatile deployment of rules involves usage of an application server. However, this is not a suitable deployment option on the public cloud because we are constrained by the vendor's platform and capabilities. Instead we use an embedded rule engine, as shown in Figure 2, where the rule engine loads a ruleset from a JAR file and executes in the same JVM as the invoking application. This ruleset is exported from the rule project that was described in Part 1, using either WebSphere ODM Rule Designer or Decision Center console. Note that the ruleset should be thoroughly unit tested in the Rule Designer or Decision Center before it is exported.

Figure 2. Application architecture



`SudokuSolver`, the key server-side class, instantiates a `RuleEngine` class, which in turn uses an embedded WebSphere ODM rule engine (`IlrContext`) for its execution. The ruleset is loaded from a JAR file that is packaged with the application. During execution, the `RuleEngine` makes use of the `ServletContext` to load the ruleset JAR as a resource, as shown in line 17 of the code listing below. The `RuleEngine` makes use of the WebSphere ODM API (lines 31-36) to create the `ruleset` parameter and execute the embedded rule engine.

```
1. public class RuleEngine implements IRuleEngine {
2.  private static final Logger LOG =
   Logger.getLogger(RuleEngine.class.getName());
3.  private static String rulesetPath = "/WEB-INF/sudoku-rules.jar";
4.  private static IlrRuleset ruleset = null;
5.  private IlrContext context = null;
6.
7.  public RuleEngine(ServletContext servletContext) {
8. super();
9. try {
10.  init(servletContext);
11. } catch (Exception e) {
12.  …
13. }
14.  }
15.
16.  private void init(ServletContext servletContext) throws
    FileNotFoundException, IOException {
```

```
17. InputStream in =
    servletContext.getResourceAsStream(rulesetPath);
18. JarInputStream is = new JarInputStream(in);
19. IlrRulesetArchiveLoader rulesetloader = new
    IlrJarArchiveLoader(is);
20. IlrRulesetArchiveParser rulesetparser = new
    IlrRulesetArchiveParser();
21.
22. ruleset = new IlrRuleset();
23. rulesetparser.setRuleset(ruleset);
24. boolean parsed =
     rulesetparser.parseArchive(rulesetloader);
25. // Create the engine
26. context = new IlrContext(ruleset);
27.  }
28.
29.  public void run(Sudoku sudoku) {
30. // Initialize the inputs
31. IlrParameterMap inputs = new IlrParameterMap();
32. inputs.setParameter("sudoku", sudoku);
33. context.setParameters(inputs);
34.
35. // Execute the ruleset
36. IlrParameterMap outputs = context.execute();
37. context.reset();
38.  }
39. …
40. };
```

One downside of using an embedded engine is that it's not possible to perform hot deployments of rulesets. In other words, when rules change, the ruleset has to be regenerated from the Rule Designer or Decision Center and the Sudoku web application has to be rebuilt and redeployed in order for the new rules to take effect. Therefore, this deployment option is not viable when rules change often. A far superior option in such cases would be to use Rule Execution Server provided as a component of WebSphere ODM Decision Server. Rule Execution Server is an environment for executing rules that provides management, performance, logging and security capabilities. It delegates rule execution to an Execution Unit (XU), which runs as a resource adapter for Java EE Connector Architecture (JCA 1.5) on a web server or application server and enables hot deployment of rulesets. We'll explore which of the cloud offerings permit usage of this deployment option.

With the basic web application in place, let's turn our attention to deployment on the selected public cloud infrastructure.

Deploying the Sudoku application using Google App Engine

Google App Engine for Java enables web applications built using standard Java technologies to be run on Google's scalable infrastructure. An Eclipse plugin, bundled with the App Engine and GWT SDKs, makes it easy to develop web applications using GWT and deploy them to App Engine.

App Engine uses the Java 6 JVM to run Java applications in a secured sandbox environment which isolates the application for security and service. The sandbox

imposes some restrictions on the application; for example, an app can't spawn threads or write data to a local file system. App Engine uses the Java Servlet standard for web applications. The application's servlet classes, JavaServer Pages (JSPs), static files and data files, along with the deployment descriptor (the web.xml file) and other configuration files are packaged in a standard WAR directory structure. App Engine serves requests by invoking servlets as configured in the deployment descriptor.

Up to a certain limit, resources consumed by App Engine applications are free of cost. Billing for resource above these free limits can be controlled by setting a daily resource budget, as described here.

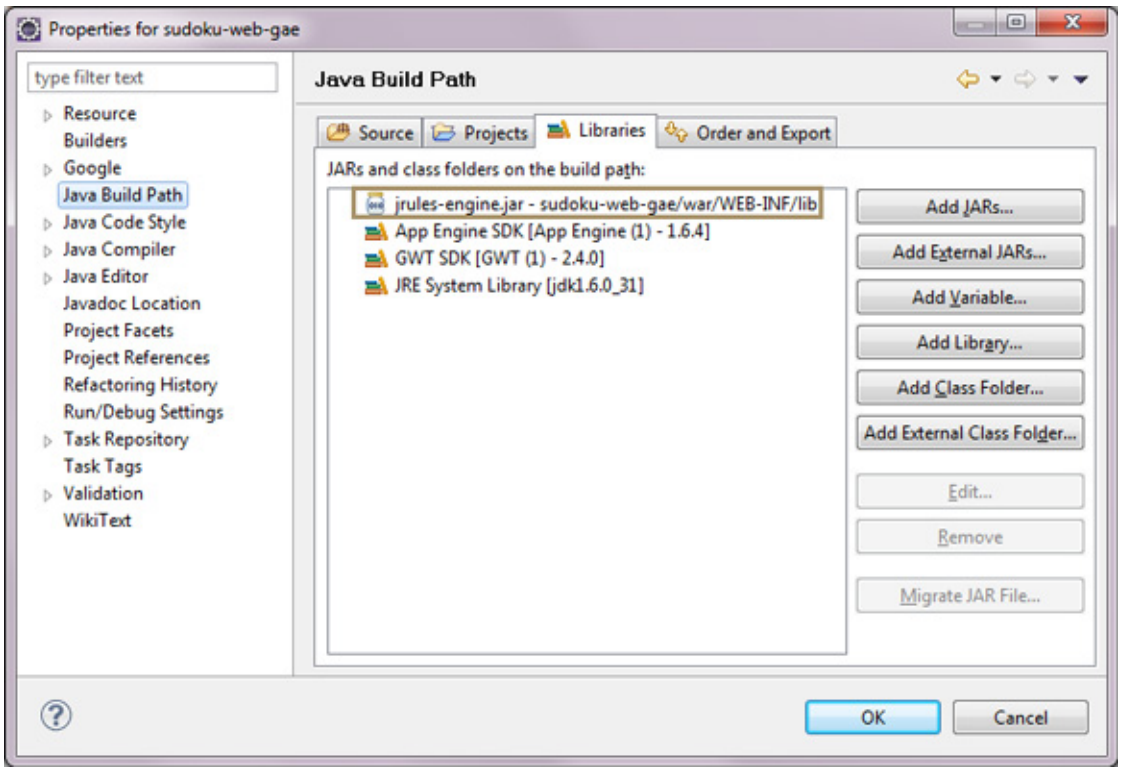
Installing Google App Engine

You can download the Google App Engine SDK for Java here and install it as an Eclipse plugin. The plugin adds new project wizards and debug configurations to the Eclipse workspace containing App Engine projects. Using this plugin, you can test and debug App Engine projects locally using a development server that simulates the App Engine sandbox restrictions. You can find more information on running the development server here.

Building and packaging the application on Google App Engine

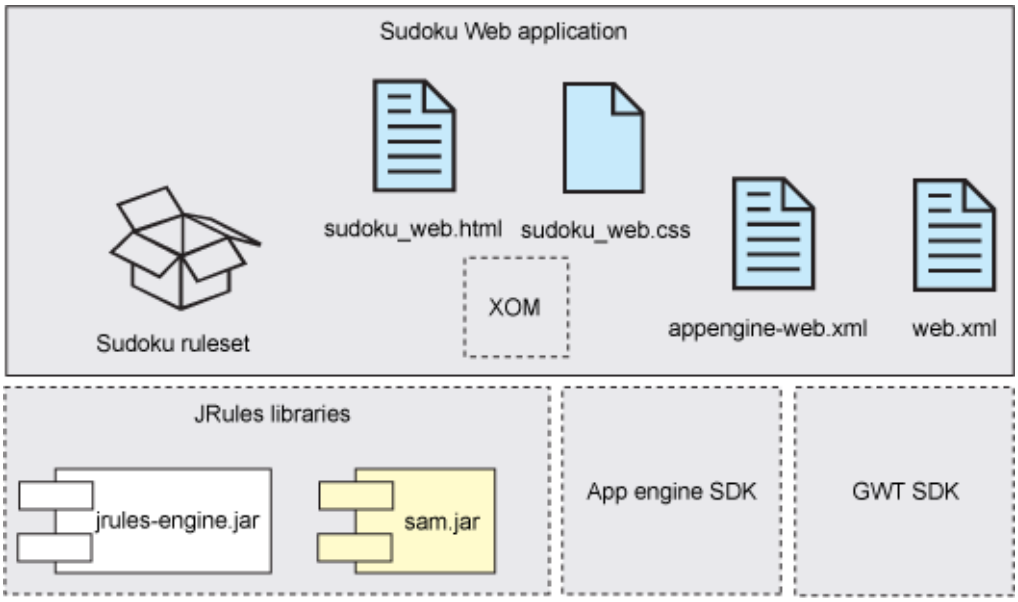
For building the web application, you need to supplement the default libraries of an App Engine application build path with the jrules-engine.jar, as shown in Figure 3.

Figure 3. Adding the rule engine JAR to the build path



However, when packaging for execution, some additional JARs are necessary. To pass the licensing checks, the sam.jar that is bundled with your installation of WebSphere ODM must be packaged with the application. In addition, the ruleset is bundled as a JAR file and placed in the WEB-INF directory so that the rules can be accessed and loaded by the rule engine. Also in the WEB-INF directory is the web.xml, which has the servlet mappings, and the appengine-web.xml, which contains App Engine settings. Figure 4 shows the packaging for deployment.

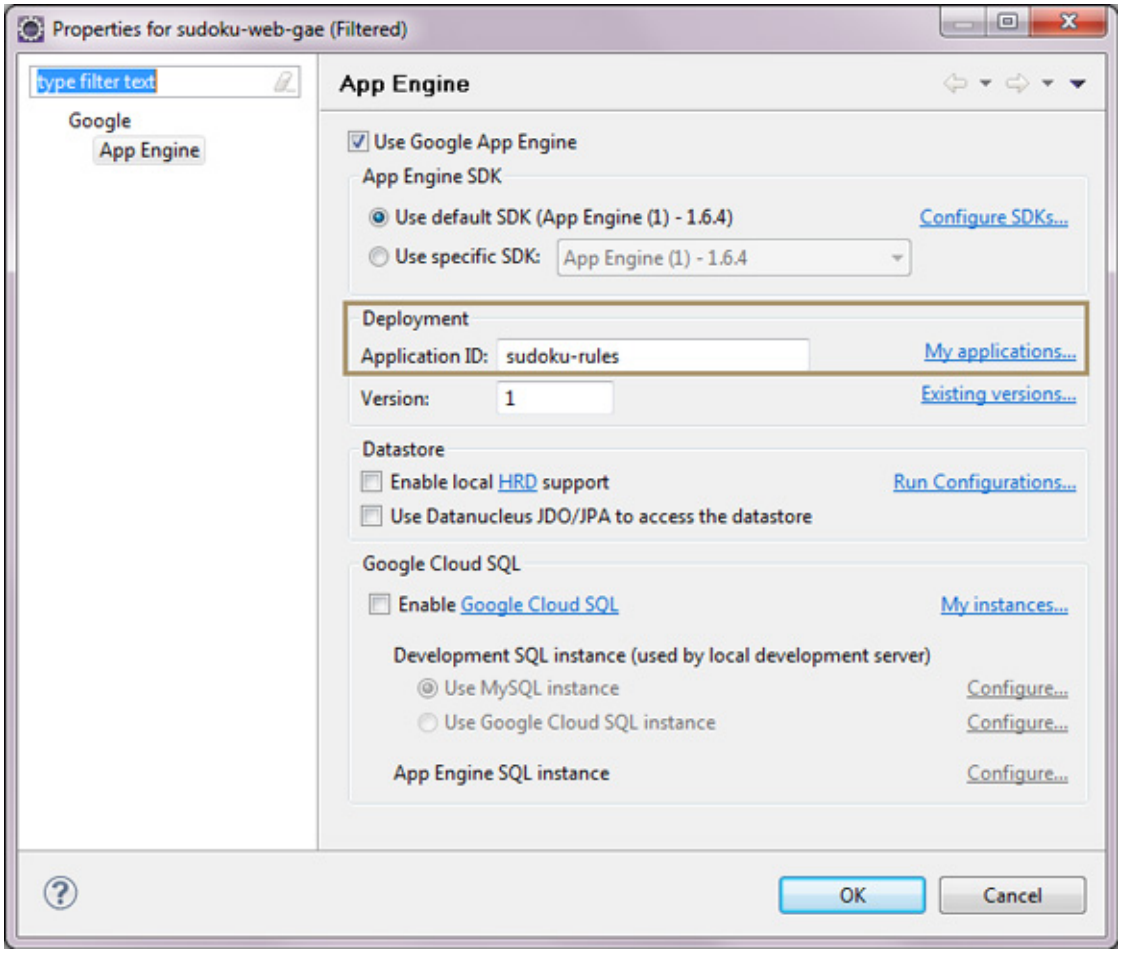
Figure 4. Application packaging for execution



Deploying the application on Google App Engine

Before deploying an application, you need to obtain an application ID from the App Engine Administration Console. This is where the application identifier is chosen: sudoku-rules.appspot.com in our case. This identifier is used in the Google App Engine settings for the project as shown in Figure 5, thereby linking the development workspace with the cloud application.

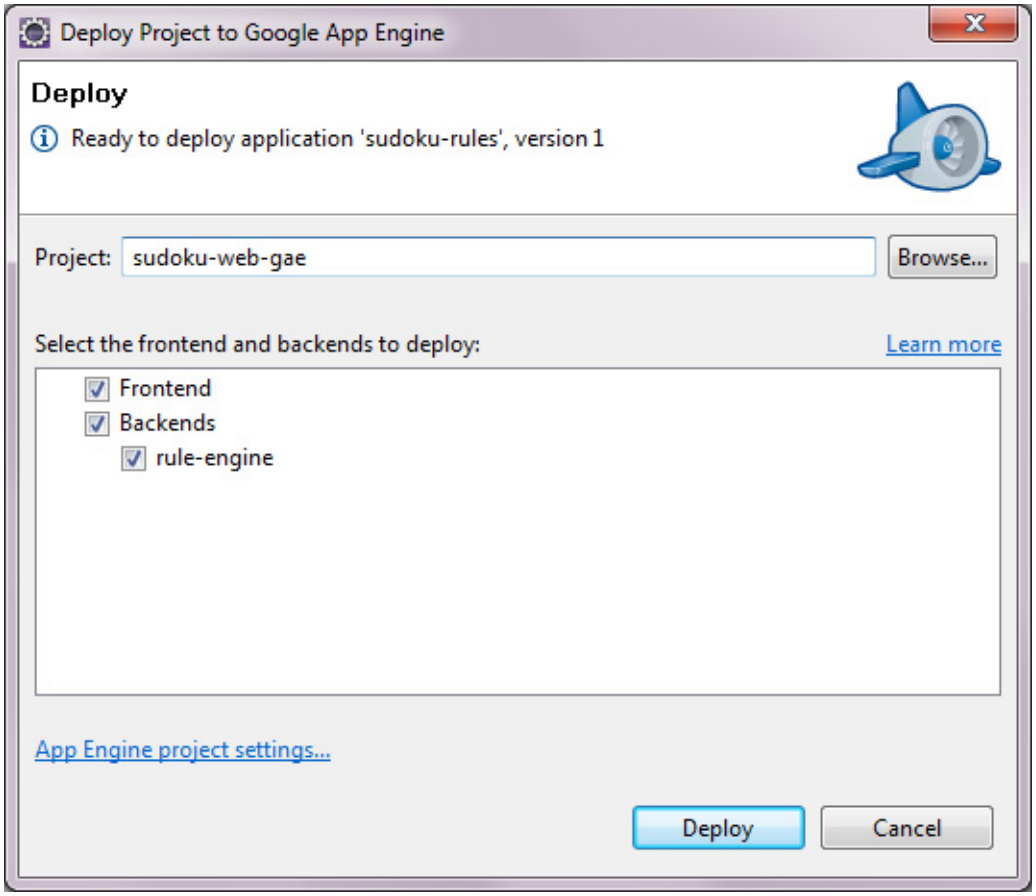
Figure 5. App Engine settings



Deploying the application to the App Engine involves two steps:

1. Compiling the application using the GWT compiler to generate the compiled application artifacts in a standard WAR directory structure.
2. Uploading the application to the cloud, which is done using the deployment wizard provided with the plugin, as shown in Figure 6.

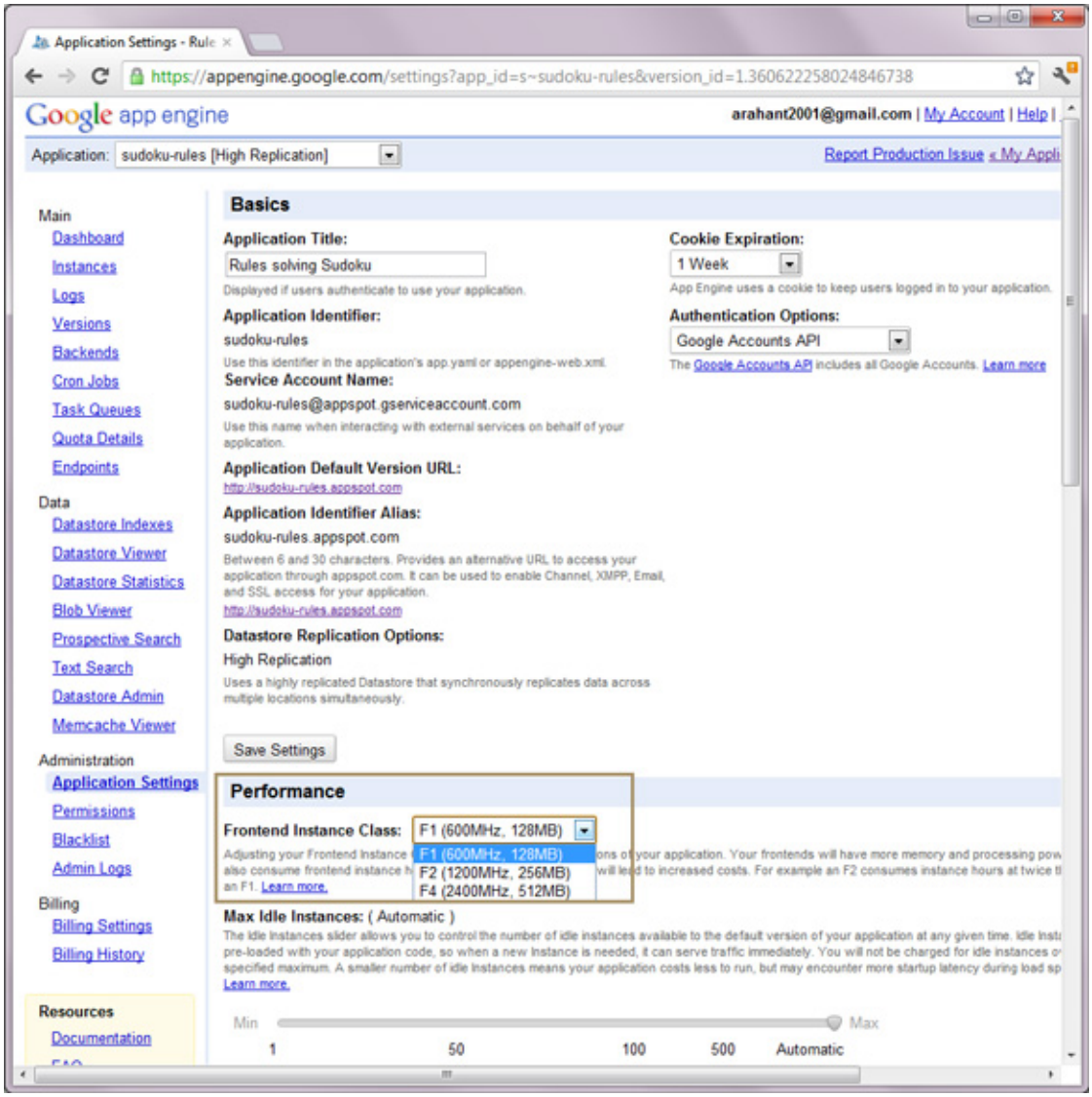
Figure 6. Deploying the application to Google App Engine



Executing the application on Google App Engine

Once deployed, the web application can be accessed on the internet by pointing your browser to http://sudoku-rules.appspot.com/. However, executing of the Sudoku application on App Engine has some serious challenges. Rule parsing and execution is extremely slow, to the point that solving most of the Sudoku configurations exceeds the 60 seconds that is the limit for an App Engine application to respond. To overcome this, first an attempt is made to upgrade the instance class that the application is run on. This is configured in the application settings of the App Engine administration console, as shown in Figure 7. However, even with the most powerful front-end server instance class of F4, which is a 2400 MHz CPU with 512 MB of RAM, the performance is grossly inadequate and most requests fail to reach a solution in the allotted 60 seconds.

Figure 7. Application settings in App Engine



(See a larger version of Figure 7.)

Google App Engine provides another execution pattern called *backends*, which are special App Engine instances that have no request deadlines. Each backend instance has a unique URL formed by adding a backend prefix to the URL: http://rule-engine.sudoku-rules.appspot.com/ for the Sudoku Web application. There are four backend instance classes, as shown in Table 1. Using a configuration file (backends.xml in WEB-INF directory), we use an App Engine instance of the B2 class. With this backend, the 60 second interruption is avoided and the Sudoku solution is obtained after the application runs for multiple minutes. However, even bumping it up to a B8 class does not appreciably improve performance. This suggests that it is neither CPU nor memory that is the performance bottleneck. There appears to be something inherent to the App Engine JVM that makes the rule execution extremely slow.

Backend Class configuration	Memory limit	CPU limit
B1	128MB	600MHz
B2 (default)	256MB	1.2GHz
B4	512MB	2.4GHz
B8	1024MB	4.8GHz

Deploying the Sudoku application using Amazon Web Services

Given the inadequate performance I saw when running the application on App Engine, I turned my attention to Amazon Elastic Beanstalk, which is offered as part of Amazon Web Services (AWS). Amazon Elastic Beanstalk may be considered a PaaS offering that handles the deployment details of load balancing, auto-scaling and application monitoring. It leverages other AWS services such as Amazon Elastic Cloud Compute (Amazon EC2), Elastic Load Balancing, and Auto Scaling. Amazon Elastic Beanstalk is built on commonly used technology stacks and allows the use of Apache Tomcat on Linux™ to run Java applications.

AWS also offers a free usage tier for new users for one year, which allows 750 hours of Amazon EC2 Linux†Micro Instance usage (613 MB of memory) each month.

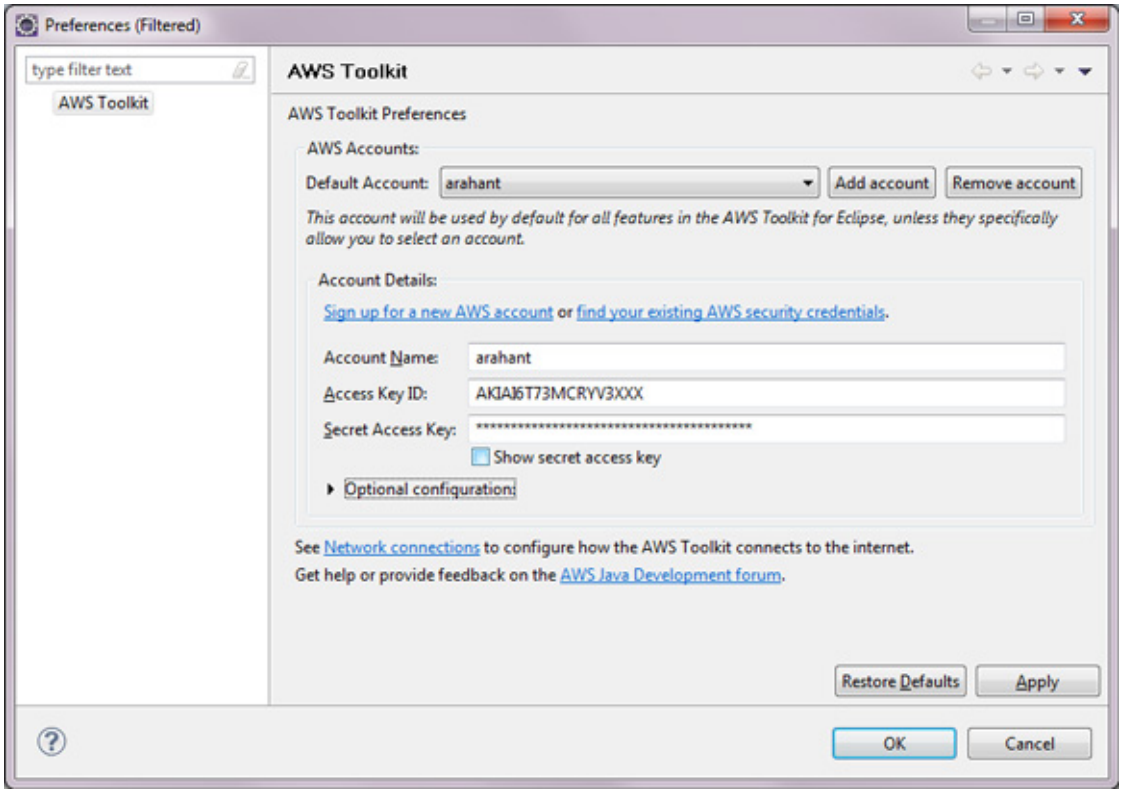
Installing and configuring AWS

The AWS Toolkit for Eclipse is an open source plugin for Eclipse that contains AWS libraries and project templates that aid a developer in developing and deploying Java applications using AWS. It integrates the AWS Elastic Beanstalk with the Eclipse IDE.

Before creating an AWS application, you need to sign up for an AWS account and also for any of the individual services used. The Sudoku application only makes use of the Elastic Compute Cloud (EC2) service, which is a virtual computing environment that can be configured with a variety of operating systems and Amazon Machine Images (AMI). I chose a 64-bit Amazon Linux running Tomcat 6.

The AWS toolkit is used to create a new AWS Java web project, which creates a web application project configured with the AWS SDK and AWS credentials as specified in the preferences panel, shown in Figure 8.

Figure 8. AWS configuration



(See a larger version of Figure 8.)

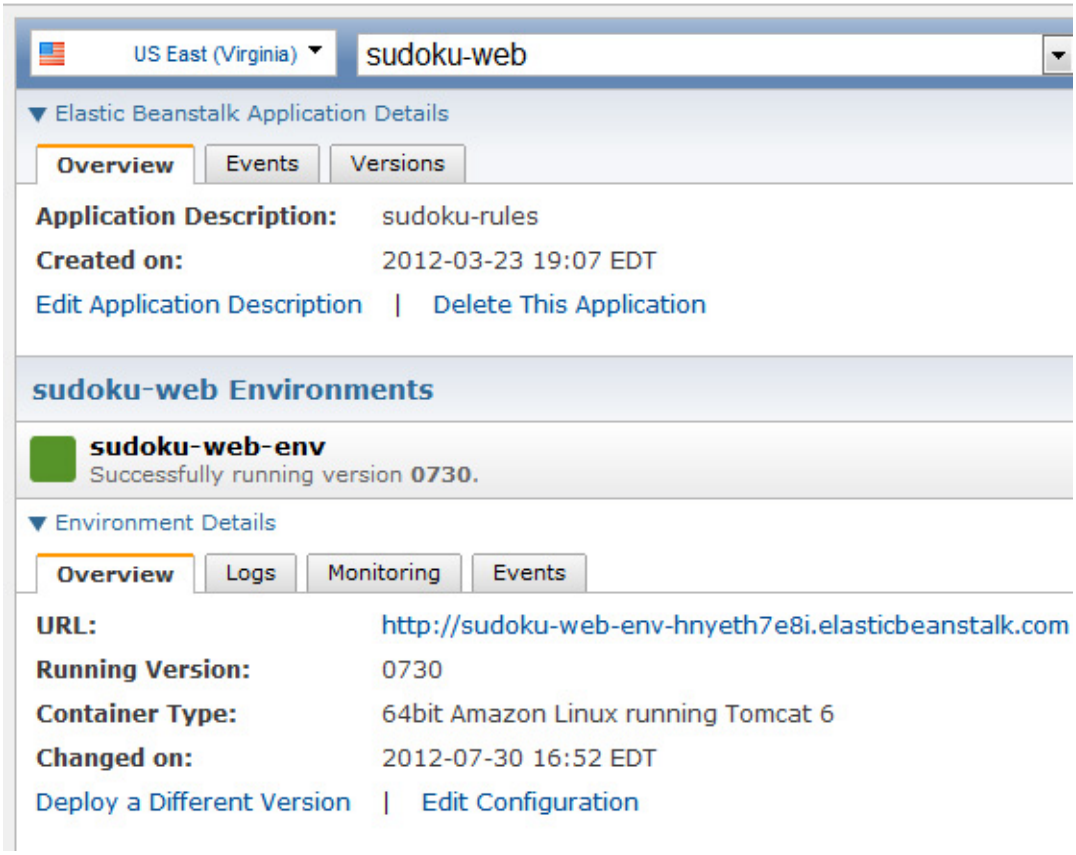
Building and packaging the application on AWS

No additional coding is necessary to deploy the Sudoku application on AWS. The compiled application artifacts from the App Engine WAR directory are simply copied over to the WebContents directory of the AWS project. The packaging of the deployed application is quite similar to that for Google App Engine, except that the App Engine SDK is swapped out for the AWS SDK. These application artifacts are packaged into a WAR file that can then be deployed to any Tomcat server, including the one provided by the Amazon Elastic Compute Cloud.

Deploying the application on AWS

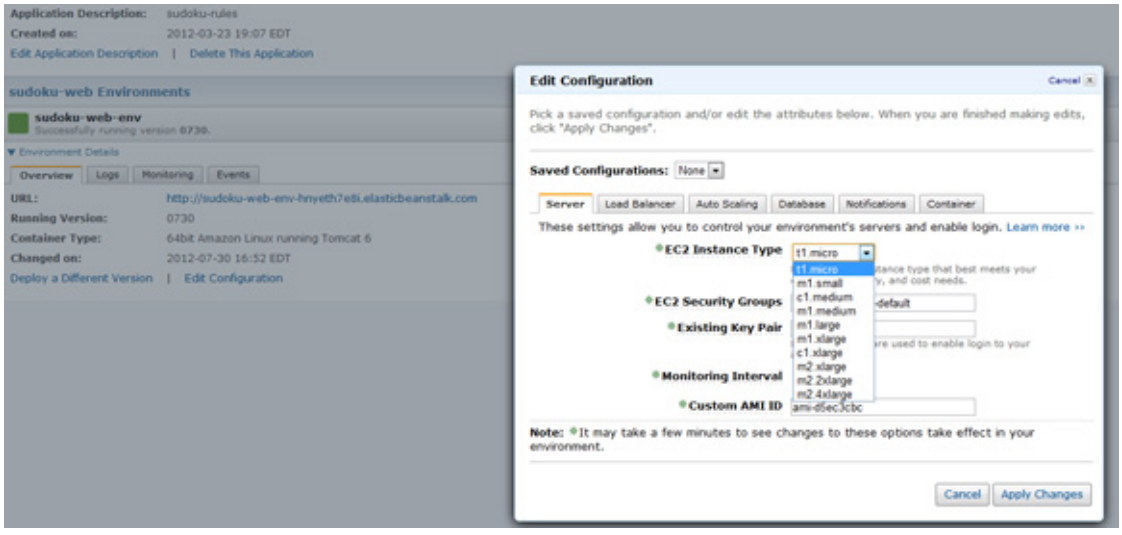
The AWS Management Console, which is a web-based interface for managing cloud resources, is used to upload and deploy the WAR file as shown in Figure 9.

Figure 9. AWS Management Console



As part of the application deployment configuration, you specify the instance type used, which may range from micro (613 MB of memory, up to 2 compute units) to High-Memory Quadruple Extra Large Instance (68.4 GB of memory, 26 EC2 Compute Units). For the Sudoku application, I used the micro instance (as shown in Figure 10, which I found to be sufficient for testing purposes.

Figure 10. AWS deployment configuration



As before, the WAR file deployed to EC2 uses an embedded rule engine. However, you do have the more flexible option of using the Rule Execution Server Java SE deployment. On the Java SE platform, the Execution Unit (XU) is deployed as a simple JAR and WebSphere ODM rule engine pooling is enabled. Full management and monitoring support of the Rule Execution Server can be harnessed by packaging the Java SE execution stack with the web application. This makes use of external persistence to store the ruleset, which is not an option for Google App Engine, but is an option with AWS. To maintain parity with the App Engine deployment, I didn't exercise this option.

Executing the application on AWS

You can invoke the Sudoku application by pointing your browser to http://sudoku-web-env-hnyeth7e8i.elasticbeanstalk.com/. As expected, the look and feel of the application is identical to the application deployed on App Engine. However, performance is a different matter: the application's performance is a couple of orders of magnitude better on AWS than on App Engine. Even fairly tough puzzles are solved in a matter of a few seconds!

Conclusion

The free usage tier offered by Google App Engine and Amazon Web Services makes it easy for developers and architects to get hands-on experience building cloud applications. As the demands on these cloud applications for smart decision-making increase, so will the need to integrate with rule engines. This article showed you how a web application built using GWT can leverage a rule engine during execution. It also illustrated how the Sudoku web application can easily be deployed to multiple cloud platforms. You saw that using an embedded rule engine as the rule integration pattern worked in both App Engine and AWS. You also saw which components of WebSphere ODM should be packaged with the web application.

Cloud offerings from different vendors vary widely in their capabilities and restrictions. With Google App Engine, you're restricted to an embedded engine. However, with AWS you have more rule deployment options, such as Java SE deployment on Tomcat. In general, as compared with AWS the Google App Engine sacrifices flexibility for ease of use.

Running rule applications on the Cloud is not without its challenges. For example, I had to resort to using a "backend" in App Engine to execute the rules. Unexpectedly, there was a marked difference between AWS and App Engine in the performance of rule applications. Google App Engine had very poor performance on the Sudoku application even with the largest application instance class, while AWS provided snappy performance with its smallest instance class.

Resources

- Cloud Computing
- IBM SmartCloud Application Services
- Get Started with the GWT SDK
- Google Web Toolkit: Making Remote Procedure Calls
- Google Web Toolkit: Developer's Guide
- Rule Execution Server overview
- Google App Engine
- AWS Elastic Beanstalk
- Using business rules in the cloud to solve Sudoku, Part 1: Implementing the rules application
- IBM WebSphere Operational Decision Management V7.5 Information Center: The information center contains information describing the IBM WebSphere Operational Decision Management product line and features.
- WebSphere Operational Decision Management V7.5: Try the WebSphere Operational Decision Management samples online.
- developerWorks BPM zone: Get the latest technical resources on IBM BPM solutions, including downloads, demos, articles, tutorials, events, webcasts, and more.
- IBM BPM Journal: Get the latest articles and columns on BPM solutions in this quarterly journal, also available in both Kindle and PDF versions.

About the author

Rajesh Rao

Rajesh (Raj) Rao has been working in the area of expert systems and business rule management systems for over 20 years, during which time he has applied business rules technology to build diagnostic, scheduling, qualification, and configuration applications across various domains such as manufacturing, transportation, and finance. He has been with IBM since2009. With a background in Artificial Intelligence, his interests include Natural Language Processing and Semantic Web.