

Build a hotel booking application using the Business Rules service in Bluemix

Yiquan Zhou (<https://www.ibm.com/developerworks/community/profiles/html/profileView.do?key=71173616-18c0-42bc-b587-b1e2cb6ffd03&lang=en>)

Software Engineer, Operational Decision Manager, IBM
IBM

08 July 2014

Benjamin Ratiarisolo (<https://www.ibm.com/developerworks/community/profiles/html/profileView.do?key=6ca15118-d726-4e42-b197-078353a63e91>)

ODM Software Developer, Enterprise Integration, IBM
IBM

The Business Rules service hosted in Bluemix and powered by Operational Decision Manager allows you to define, deploy, and maintain business rules and policies separately from your application code. To demonstrate the benefits of decoupling the lifecycle of the calling application from the business logic (rules) running in a cloud service, we build a simple hotel booking application in a Node.js runtime bound to a Business Rules service that calculates the booking rates. This article will show you how to deploy and execute business rules in the cloud environment. You will then be able to extend this application with more complex rules or build your own application very easily.

Updated 28 October 2015

The [Business Rules service](#) hosted in [IBM Bluemix™](#) and powered by Operational Decision Manager allows you to define, deploy, and maintain business rules and policies in a RuleApp separate from your application code, providing greater application agility. You can update the business logic in the RuleApp and redeploy it without any change to the booking application, spending less time recoding and testing business policy changes such as pricing calculation, eligibility determination or credit approvals.

To demonstrate the benefits of decoupling the business logic from the lifecycle of a calling application running in the cloud, we'll walk you through the steps to build a sample application that takes advantage of the Business Rules service. You will then be able to extend this application with more complex rules or build your own application very easily.

“ This sample booking application shows the possibilities of integrating the Business Rules service with a Node.js application in the cloud, while also taking advantage of the ease of deployment and the scalability of the Bluemix platform. ”

Our sample application is for a hotel chain that is building its reservation system and wants to provide its clients with an application to search for and book rooms. The owners of the hotel need to define various business policies to calculate booking rates, such as early booking discounts or last-minute offers. They may need to modify these policies to adapt to different travel seasons and exceptional events, for example. They may also want to add more policies in the future on special offers or loyalty programs.

To accomplish all this, we'll build an application with Node.js and use the Business Rules service to easily manage and execute the business rules that define these policies. Meanwhile, we also take advantage of the ease of deployment and the scalability of the Bluemix platform.

[Run the app](#)
[Get the code](#)

What you will need for your application

1. Familiarity with [Node.js](#).
2. Some Node.js modules: [Express framework](#), [EJS](#), [async](#).
3. Basic knowledge of HTML and [Bootstrap 3](#) CSS.
4. Basic knowledge of [IBM Operational Decision Manager](#) (ODM) (recommended).
5. The Eclipse Juno 4.2.2 IDE for Java EE Developers. Install the [Rule Designer plugins for the Business Rules service](#).
6. Cloud Foundry Command-Line Interface V6. Download from [GitHub](#) and launch the installer.

Build the hotel booking app

Step 1. Create a Business Rules service instance in Bluemix

1. From the Bluemix catalog, click the **Business Rules** service. In the **Add Service** section, name your service BlueBooking-BusinessRules, for example, and select **[Leave**

unbound]. (You will bind the service to your application in the next step.) Click **CREATE**.

Add Service

Space:

dev

App:

Leave unbound

Service name:


BlueBooking-BusinessRules

Selected Plan:

Standard

CREATE

2. In the BlueBooking-BusinessRules service details page, note the following information:
 - Console URL and credentials (for ruleset deployment)
 - REST Execution API endpoint URL and credentials (for ruleset execution)

 **BlueBooking-BusinessRules**

[Home](#) [Get Started](#)

- Use this information to [create](#) a Rule Execution Server Configuration in your Eclipse IDE.

URL:

[Open Console](#)

Username:

*

Password :

*

Show Password

- Use the following information to authenticate calls to the service API.

REST Execution API:

SOAP Execution API:

Username:

*

Password (RFC2617):

*

Show Password

Pre-encoded for Basic Authentication

if you have bound this service to an application, this information can be retrieved directly from the VCAP_SERVICES.

Step 2. Define your application business logic with Rule Designer

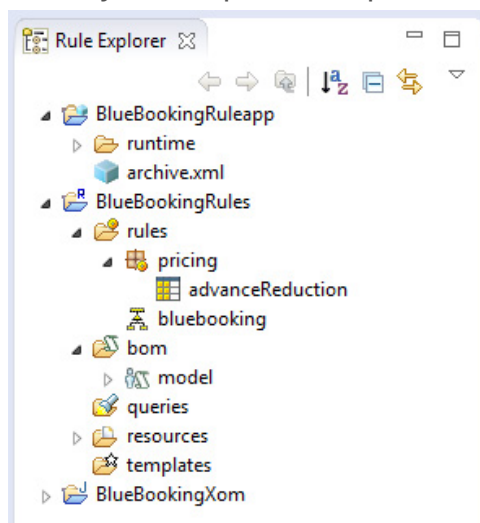
In this step, you will go through the core concepts and activities for authoring business rules in Rule Designer. However, the focus of this article is not to provide detailed explanations about this; for more information, see below.

READ: [Getting started with business rules](#)

Before proceeding with this step:

- Make sure you have installed the [Rule Designer plugins for the Business Rules service](#) in your Eclipse Juno 4.2.2 IDE.
- Check out the BlueBookingXom and BlueBookingRules projects from the Git repository:
`git clone https://hub.jazz.net/git/ruleservicesample/BlueBookingV1`

- Import them into your Eclipse workspace, then switch to the **Rule** perspective to open **Rule**



Explorer.

1. Define your execution object model (XOM)

The execution object model (XOM) is the runtime model against which the rules are executed. IBM Operational Decision Manager (ODM) supports execution object models built from Java™ or XML sources. For the purpose of this article, you will be working with a simple Java-based XOM that is defined in the `BlueBookingXom` Java project and contains two classes:

1. The `Hotel` class represents a hotel with the following attributes:
 - Hotel name
 - Hotel location (city)
 - Base rate for a room
2. The `Result` class defines the result of a client request with the following information:
 - Hotel
 - Check-in date
 - Check-out date
 - Booking rate as computed by the business rules execution

2. Specify your business object model

The business object model (BOM) is the model upon which rules are authored. You usually create a BOM entry from an existing XOM source. In our case, the `BlueBookingRules` Rule project already contains a BOM entry created from the `BlueBookingXom` Java XOM.

3. Author your business logic

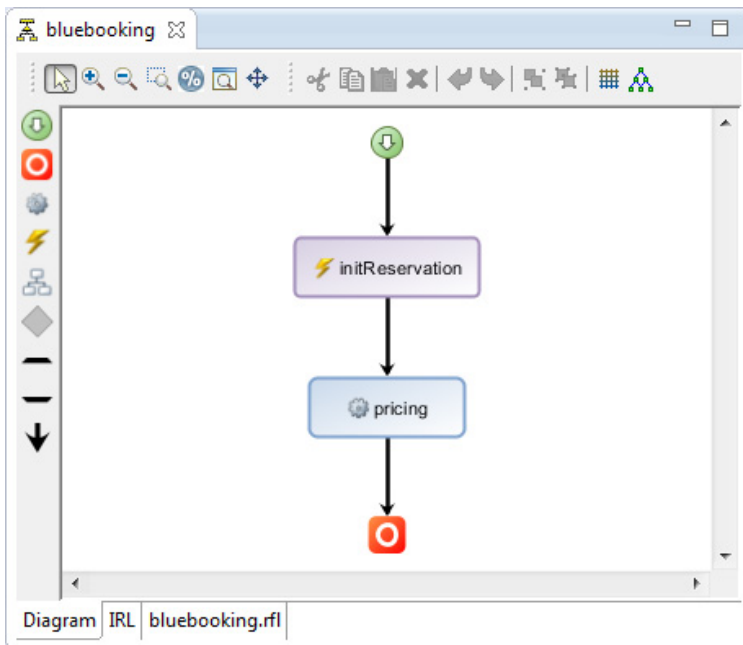
Now you can author your application business logic. The sample `BlueBookingRules` Rule project contains the `advanceReduction` decision table and the `bluebooking` rule flow.

The `advanceReduction` decision table computes discounts depending on how early the reservation is made.

	the number of days ahead of arrival		the reduction	the message
	min	max		
1		> 7	30	You benefit a -30% reduction from our early booking offer!
2	2	7	0	Best rates guarantee in our hotels!
3	0	1	50	You benefit a -50% reduction from our LAST MINUTE OFFER!
4				
5				

The bluebooking rule flow defines the execution as two sequential tasks —`initReservation` and `pricing`:

- `initReservation` initializes the `Result` instance that will be returned after rules are executed.
- `pricing` triggers the computation of the discount to be applied.



4. Define the contract between the business logic and your client application

Ruleset parameters specify the contract between your business logic and your client application. In this case, you have:

- Three input parameters: hotel, check-in date, check-out date
- One output parameter that is passed to your client application after execution of the business logic: an instance of the class `Result`

Ruleset Parameters

Define ruleset parameters.

Name	Type	Direction	Default Value	Verbalization
hotel	com.ibm.bluebooking.Hotel	IN		the hotel
checkin	java.util.Date	IN		the date of checkin
checkout	java.util.Date	IN		the date of checkout
result	com.ibm.bluebooking.Result	OUT		the result

Step 3. Deploy your business logic to the Business Rules service in Bluemix

Before proceeding with this step, you may want to familiarize yourself with the deployment architecture of a business rule application:

READ: [Deployment architecture](#)

Create a RuleApp project in Rule Designer

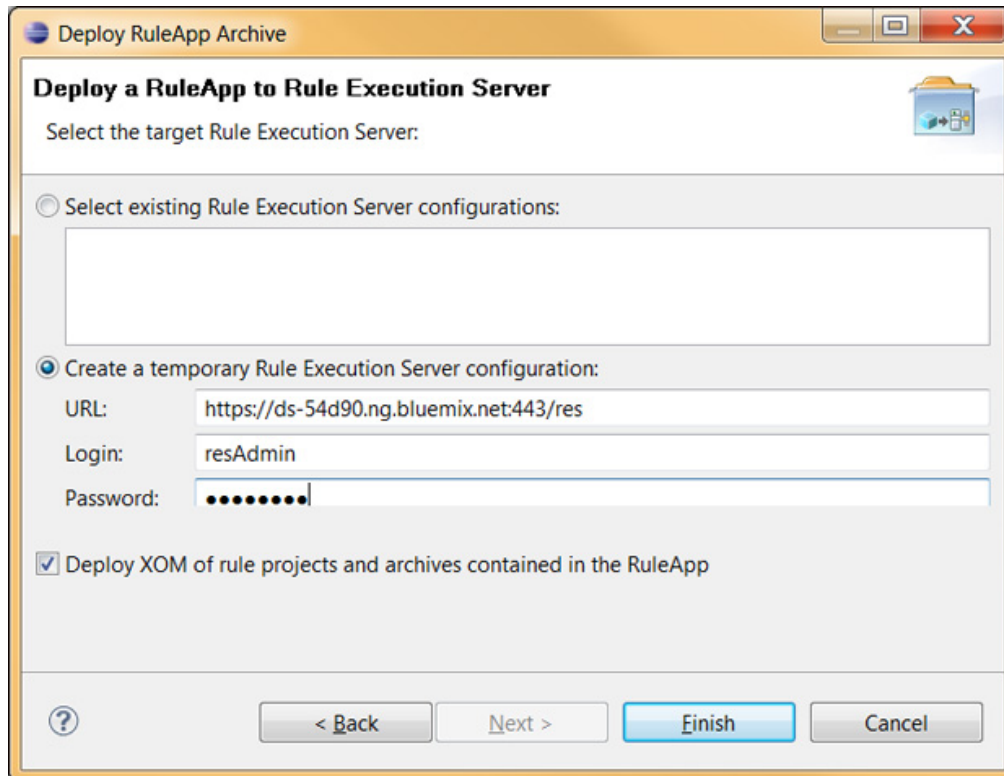
You will deploy the business rules through a RuleApp project:

1. In the **Rule Explorer** view, select the `BlueBookingRules` Rule project.
2. Click the **Create RuleApp project** link in the **Deploy and Integrate** section of the **Rule Project Map** view.
3. In the **New RuleApp Project** wizard, set the project name as `BlueBookingRuleApp`, then click **Finish**. A new RuleApp project containing the `BlueBookingRules` Rule project (as ruleset) will be created in your workspace.

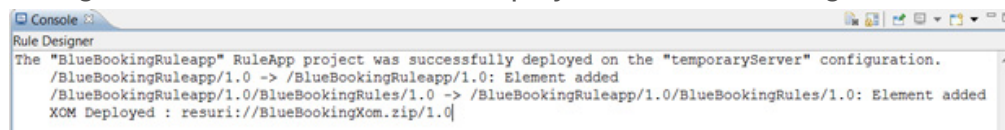
Deploy the RuleApp to the Business Rules service in Bluemix

Now you are ready to deploy your RuleApp to the Rules service:

1. In the **Rule Explorer** view, select the `BlueBookingRuleApp` project.
2. Right-click the RuleApp project and select **RuleApp > Deploy**.
3. In the **Deploy RuleApp Archive** wizard, click **Next** to keep the default versioning policy for RuleApp and ruleset deployment. A warning dialog opens if you are using a JDK 7. Click **OK** to close it.
4. On the next wizard page, select **Create a temporary Rule Execution Server configuration**. Fill in the URL of the Administration Console, the login, and the password with the corresponding information for your [Business Rules service](#) instance. Click

**Finish.**

If all goes well, the **Console** will display a success message:



Step 4. Build a Node.js Express application in Bluemix

1. Create a Node.js starter application and bind it to the Business Rules service

- In the Bluemix catalog, click the **SDK for Node.js** runtime. In the **Start with a runtime:** section, give your application a name and a host URL that will be its access point

on the Internet (mybookingapp.mybluemix.net, for example). Then click **CREATE**.

Start with a runtime:

Space:

dev

Name:

mybookingapp

Host:

mybookingapp . mybluemix.net

Selected Plan:


Default

CREATE

- b. Select the app, then click **BIND A SERVICE**. Select the Business Rules service instance you created in the previous step, and click **ADD**. The Business Rules service is now bound to your

Add Service to Application 'mybookingapp'

Select the previously used service instance that you want to add to your application.

	NAME	SERVICE	VERSION
	BlueBooking-BusinessRules	businessrules	

application.

- c. You can now download the sample starter application generated by Bluemix and do some modifications. Click **VIEW > GUIDE**, then click **Download the starter application package** to download the ZIP file. Extract it to your file system.

2. Add and download dependencies

Let's start by adding the required Node.js modules to the dependencies. Modify the application name, description, and dependencies in the `package.json` file.

```
{
  "name": "mybookingapp",
  "version": "0.0.1",
  "description": "A simple hotel booking app using Rules Service in BlueMix",
  "dependencies": {
    "express": "3.4.7",
    "ejs": "0.8.5",
    "async": "0.7.0"
  },
  "engines": {
    "node": "0.10.0"
  },
  "repository": {}
}
```

Run an `npm install` command from the root directory of your application to download the dependencies to the `node_modules` directory.

3. Craft a page to search hotels

In the sample, we chose EJS as the template engine because it uses the standard HTML syntax. But you can use any other template engine like Jade, Hogan, or Underscore.

- a. Modify the `'view engine'` configuration parameter to make it the default for your application:

```
app.set('view engine', 'ejs');
```

- b. Create an `index.ejs` file in the `views` folder of the project and create an HTML form with three inputs: the city, the check-in date, and the check-out date to search for hotels. You may want to give your page a better look and make it responsive to different screen sizes with the help of Bootstrap, but we will not go into details on HTML markup and CSS style in this article. Alternatively, you can copy and paste the code from the `BlueBookingServer` project in Git repository.
- c. When the `index.ejs` is completed, you simply route the default requests to this main page.

```
// Main app page
app.get('/', function(req, res){
  res.render('index');
});
```

4. Find hotels in a city

When the search form is submitted, the `GET` request is routed to `/hotels`. We want to extract the parameters from the request and find the list of available hotels in the requested city. Note that the dates are parsed in Coordinated Universal Time (UTC) time zone.

```
// Hotel search results
app.get('/hotels', function(req, res){
  // get the request parameters
  var city = req.query.city;
  // the date string is in mm/dd/yyyy format
  var fromDateStr = req.query.from;
  var toDateStr = req.query.to;
  // parse the date string in UTC timezone
  var fromDate = Date.UTC(fromDateStr.split('/')[2], fromDateStr.split('/')[0]-1, fromDateStr.split('/')[1]);
  var toDate = Date.UTC(toDateStr.split('/')[2], toDateStr.split('/')[0]-1, toDateStr.split('/')[1]);
  ... //render the page
});
```

For simplification, we will not query a database for hotels in this sample application. Instead, we load a data file (data/hotels.json) and return an array of hotels for the queried city.

```
function findHotels(city) {
  var hotels = require(__dirname + '/data/hotels.json');
  return hotels[city];
}
```

5. Invoke the Business Rules service for results

When we bind the [Business Rules service](#) to our application, the service configuration is added to `VCAP_SERVICES`, a read-only environment variable of your application. You can see it in your dashboard.

```
with:
{
  "businessrules": [
    {
      "name": "BlueBooking-BusinessRules",
      "label": "businessrules",
      "plan": "standard",
      "credentials": {
        "executionAdminUrl": "https://brsv2-XXXXXXX.ng.bluemix.net/res",
        "executionSoapUrl": "https://brsv2-XXXXXXX.ng.bluemix.net/DecisionService/ws",
        "executionAdminRestUrl": "https://brsv2-XXXXXXX.ng.bluemix.net/res/apiauth",
        "executionRestUrl": "https://brsv2-XXXXXXX.ng.bluemix.net/DecisionService/rest",
        "password": "password",
        "user": "username"
      }
    }
  ]
}
```

- a. The ruleset deployed in the Business Rules service can be executed by invoking the execution REST API. You need to parse the `VCAP_SERVICES` variable and use it in your application to get the REST execution endpoint.

```
// Retrieve the Rules Service parameters
if (process.env.VCAP_SERVICES) {
  var env = JSON.parse(process.env.VCAP_SERVICES);
  var rules = env['businessrules'][0].credentials;
} else {
  // for local testing
  var rules = {
    "executionRestUrl": "http://{your_execution_url}.mybluemix.net/DecisionService/rest",
    "user": "{username}",
    "password": "{password}"
  };
}
```

- b. Then in the Node.js application, you can load the `https` and `url` modules and use them to code a REST client.

```
var https = require('https')
    , url = require('url');
```

- c. Create a utility function (`invokeRulesService`) that takes the following parameters:

- `rulesetPath`— a string in the form of `/ruleappName/ruleappVersion/rulesetName/rulesetVersion`. The Business Rules service supports versioning of RuleApps and rulesets. If you do not specify a version, the latest version will automatically be used.
- `inputParam`— a JSON object containing the payload of input parameters of the ruleset.

- **callback**— a callback function for the response because the HTTP request is asynchronous.

d. The function will make a **POST** request to **executionRestUrl+rulesetpath** with **Content-Type** set to **application/json** and the ruleset input parameters in the request body. The Business Rules service will execute the ruleset and return the output values in the response body under **JSON** format.

```
/*
 * Invoke the Rules Service to calculate the booking rates
 */
function invokeRulesService(rulesetPath, inputParams, callback) {
  var restUrl = url.parse(rules.executionRestUrl);
  var dataString = JSON.stringify(inputParams);
  // encode 'user:password' in Base64 string for basic authentication of the execution API
  var encodedCredentials = new Buffer(rules.user+':'+rules.password).toString('base64');

  headers = {
    'Content-Type': 'application/json',
    'Content-Length': dataString.length,
    'Authorization': 'Basic ' + encodedCredentials // basic authentication header
  };

  var options = {
    host: restUrl.host,
    path: restUrl.path + rulesetPath,
    method: 'POST',
    headers: headers
  };

  var req = https.request(options, function(resp) {
    resp.setEncoding('utf-8');
    var responseString = '';

    resp.on('data', function(data) {
      responseString += data;
    });

    resp.on('end', function() {
      console.log(responseString);
      if (resp.statusCode == 200)
        var responseObject = JSON.parse(responseString);
      callback(responseObject);
    });
  });

  req.on('error', function(e) {
    console.log(e.message);
  });

  req.write(dataString);
  req.end();
}
```

e. Finally, you can invoke the Business Rules service with the utility function for each hotel to get the result with rate. Make sure that the ruleset path is correct and the input parameter names are those defined in your ruleset. Because the call to the Business Rules service is asynchronous, you can use the **async** module to iterate through the array of hotels and return an array of results when all calls are complete. Don't forget to export this function so you can call it from **app.js**.

```
/*
 * Invoke the Rules Service to calculate the rate for each of the hotels
 */
function getResults(city, fromDate, toDate, callback) {
```

```

    var results = new Array();
    var rulesetPath = '/BlueBookingRuleApp/BlueBookingRules/';
    var hotels = findHotels(city);
    async.each(hotels, function(hotel, callback) {
    var inputParams = {"hotel": hotel, "checkin": fromDate, "checkout": toDate};
    invokeRulesService(rulesetPath, inputParams, function(responseObj) {
    results.push(responseObj['result']);
    callback();
    });
    }, function(err) {
    if (err) {
    console.log(err);
    } else {
    callback(results);
    }
    });
}
// export public functions
exports.getResults = getResults;

```

6. Display the results

Back to app.js; you can now render the hotel.ejs page with the results returned by the Business Rules service.

```

service.getResults(city, fromDate, toDate, function(results){
    // render the page with data
    res.render('hotels', {"city": city, "from": req.query.from, "to": req.query.to, "results": results});
});

```

Step 5. Push and run your application in Bluemix

We are now ready to push our Node.js application to Bluemix. You can find a manifest.yml file in your project folder that describes the name and domain of the application in Bluemix, how many instances to create, how much memory to allocate, etc.

1. Log in to Bluemix with the command-line `cf login -a api.ng.bluemix.net`. Enter your username, password, organization, and space when prompted.
2. Run `cf push` to deploy the application to Bluemix.
3. When the application is running, go to `http://mybookingapp.mybluemix.net` to test your app.

Conclusion

The Business Rules service allows you to separate business logic and application logic for greater application agility. You can update the business logic then redeploy the `RuleApp` without any change to the booking application. Therefore, you spend less time recoding and testing when business policy changes, such as pricing calculation, eligibility determination or credit approvals. This sample booking application shows the possibilities of integrating the Business Rules service with a Node.js application in the cloud, while also taking advantage of the ease of deployment and the scalability of the [Bluemix](#) platform.

Acknowledgments

Many thanks to Frederic Lavigne and Laurent Grateau for their encouragement and help, and to Christiane Mosbach for the document review.

The Business Rules service <http://www.ibm.com/developerworks/topics/business-rules-service> minimizes your code changes by keeping business logic separate from application logic.

About the authors

Yiquan Zhou



[@Yiquan_Zhou](#)

Benjamin Ratiarisolo



[@ratiaris](#)

© Copyright IBM Corporation 2014

(www.ibm.com/legal/copytrade.shtml)

[Trademarks](#)

(www.ibm.com/developerworks/ibm/trademarks/)