

# Build a portable Java travel app that integrates web services

Felicia Tucci

June 27, 2014  
(First published February 28, 2014)

Learn how to develop and deploy a Java™ PaaS web app on the cloud.

## Sign up for IBM Bluemix™

This cloud platform is stocked with free services, runtimes, and infrastructure to help you quickly build and deploy your next mobile or web application.

To demonstrate the high levels of interoperability and portability that can be available when you build and deploy applications in the cloud, I decided to build a fun little travel application.

My app, which consists of two parts, leverages the user preferences stored in a user profile to present a map showing what lodging is available in an area. The first part manages user preferences using the MongoDB service available through **Bluemix** and exposes the resulting service to the external world through an API. The second part integrates the first part with external services in a web application.

*“ A cloud application must be scalable, portable, and integrate easily with internal services. It should not be wearisome to provision or manage throughout its lifecycle. ”*

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

## Note:

- After you click on **Run the app**, you can log in with any ID and password you want.
- To fork the code for this exercise, after you click **Get the code**, click the **EDIT CODE** button in the upper right-hand corner (enter your DevOps Services credentials if you're not already logged in) and click the **FORK** button on the menu to create a new project. Alternatively, you can export the code by selecting the root folder, then select **File > Export** from the left navigation.

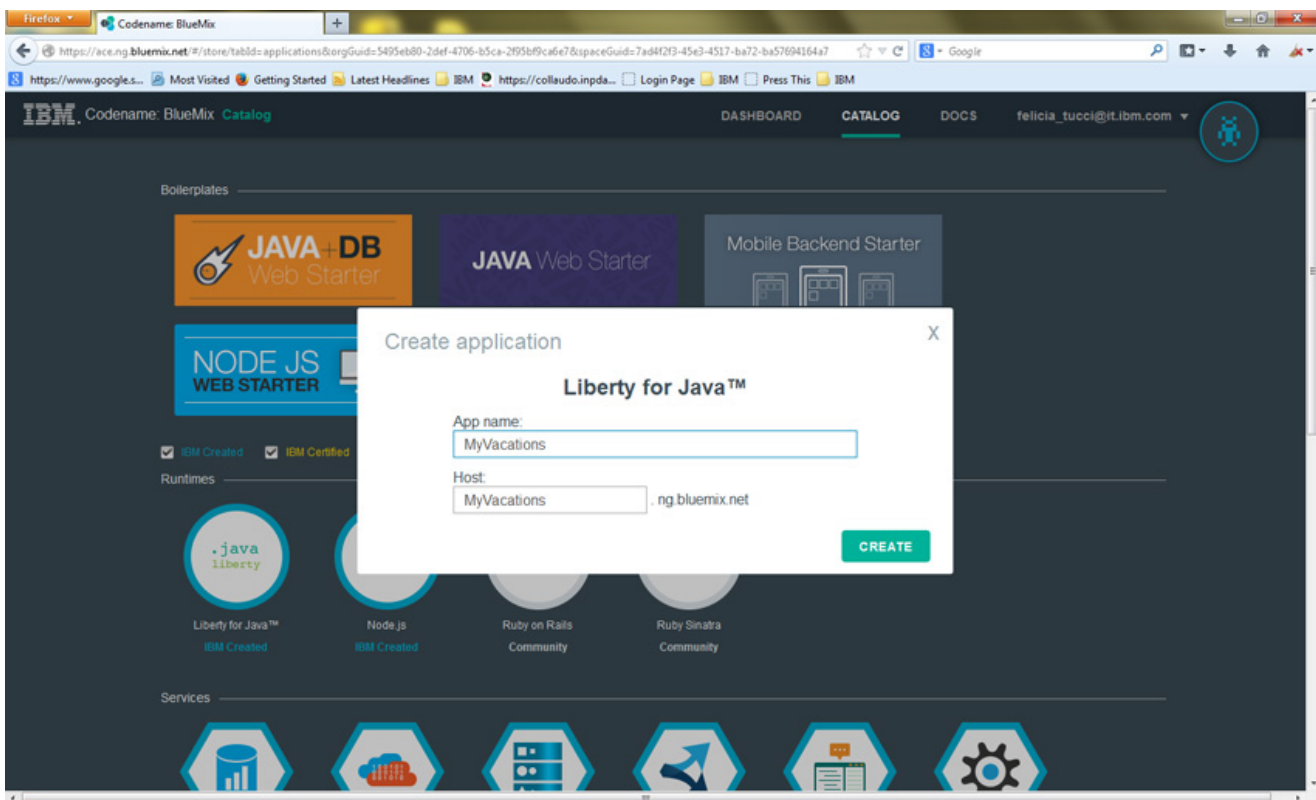
## What you'll need to build a similar app

- A basic familiarity with the [Spring MVC framework](#)

- [Spring Tool Suite](#) (or Eclipse IDE), with the Cloud Foundry plugin installed
- [MongoDB](#) NoSQL database
- [A Google Maps API](#) account for testing purposes
- [An Expedia Developer API](#) account for testing purposes
- [jquery-ui-map](#), the Google Maps V3 plugin for jQuery, which simplifies the work with the Google Maps API

## Step 1. Create the cloud application

I created and deployed this sample on Bluemix. For this sample, I chose Java with the Spring Framework. To create an application, it's enough to go to [Bluemix](#) and choose the type of application (Java stand-alone, Java Web, Ruby, etc.) — Java Web in our case.



## Step 2. Install and use cf command-line tool

You can manage the application with the Bluemix web interface or the command-line interface provided by the Cloud Foundry project. For this example, I chose the command-line interface `cf`. The command line allows you to deploy, associate service, control (start and stop) applications, and more. Download the CLI from [GitHub](#) and launch the installer. The installation result is an executable file: `cf.exe`. First, you must set the target API endpoint, then log in.

```
C:\Program Files (x86)\CloudFoundry>cf api api.ng.bluemix.net
Setting api endpoint to api.ng.bluemix.net...
OK

API endpoint: https://api.ng.bluemix.net (API version: 2.0.0)
Not logged in. Use 'cf login' to log in.
No org or space targeted, use 'cf target -o ORG -s SPACE'

C:\Program Files (x86)\CloudFoundry>cf login
API endpoint: https://api.ng.bluemix.net

Username> felicia_tucci@it.ibm.com

Password>
Authenticating...
OK

Targeted org felicia_tucci@it.ibm.com

Targeted space dev

API endpoint: https://api.ng.bluemix.net (API version: 2.0.0)
User:      felicia_tucci@it.ibm.com
Org:      felicia_tucci@it.ibm.com
Space:     dev
```

Now you can list applications, services, and bound services.

### Step 3. Prepare the development environment

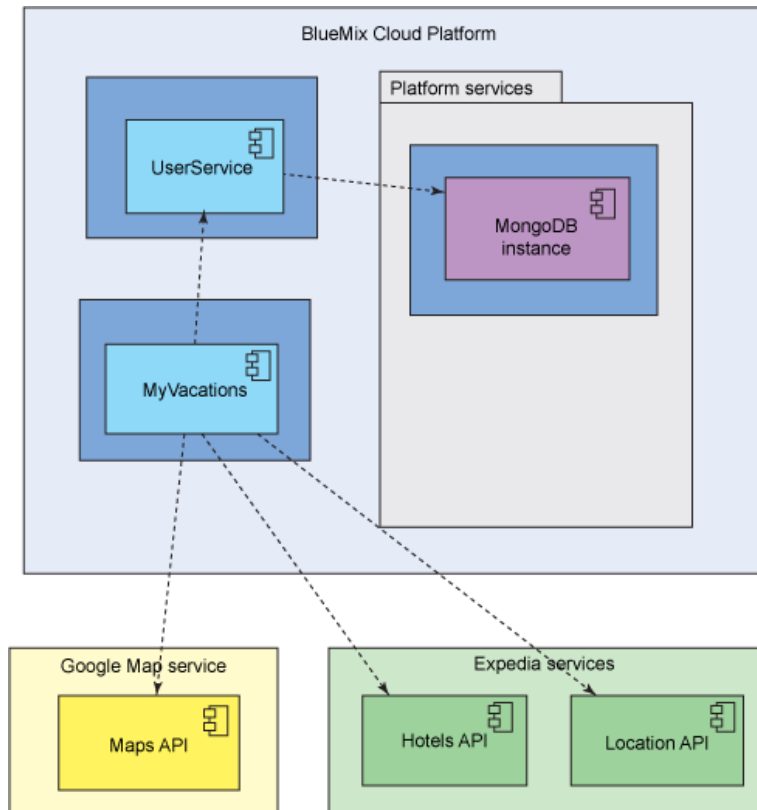
This sample uses the MVC Spring Framework. The environment used is the Spring Tool Suite with the Cloud Foundry plugin. The tools and technologies used are:

1. Spring 3.1.1
2. JDK 7
3. Spring Tool Suite 3.4.0+ Cloud Foundry Integration for Eclipse 1.5.1

When you create a Java Web Project you want to deploy on a Cloud Foundry platform, you must add the Cloud Foundry nature to project. By doing so, you create the manifest.yml file that describes the app and its resource needs to the Cloud Foundry runtime.

We are going to develop two parts to our application:

- The first is the UserService. It exposes the API to manage the user info. It uses an internal cloud platform MongoDB, a popular **NoSQL database**, to realize the persistence of data.
- The second is the MyVacations, which allows the logged-in user to search available hotels using some parameters. The UserService application provides the values of some of these search parameters. Expedia Services provides the list and info of hotels. **Google Maps API Web Services** positions the list of hotels on a map.



The UserService shows how to store information about users in a central location using the MongoDB service.

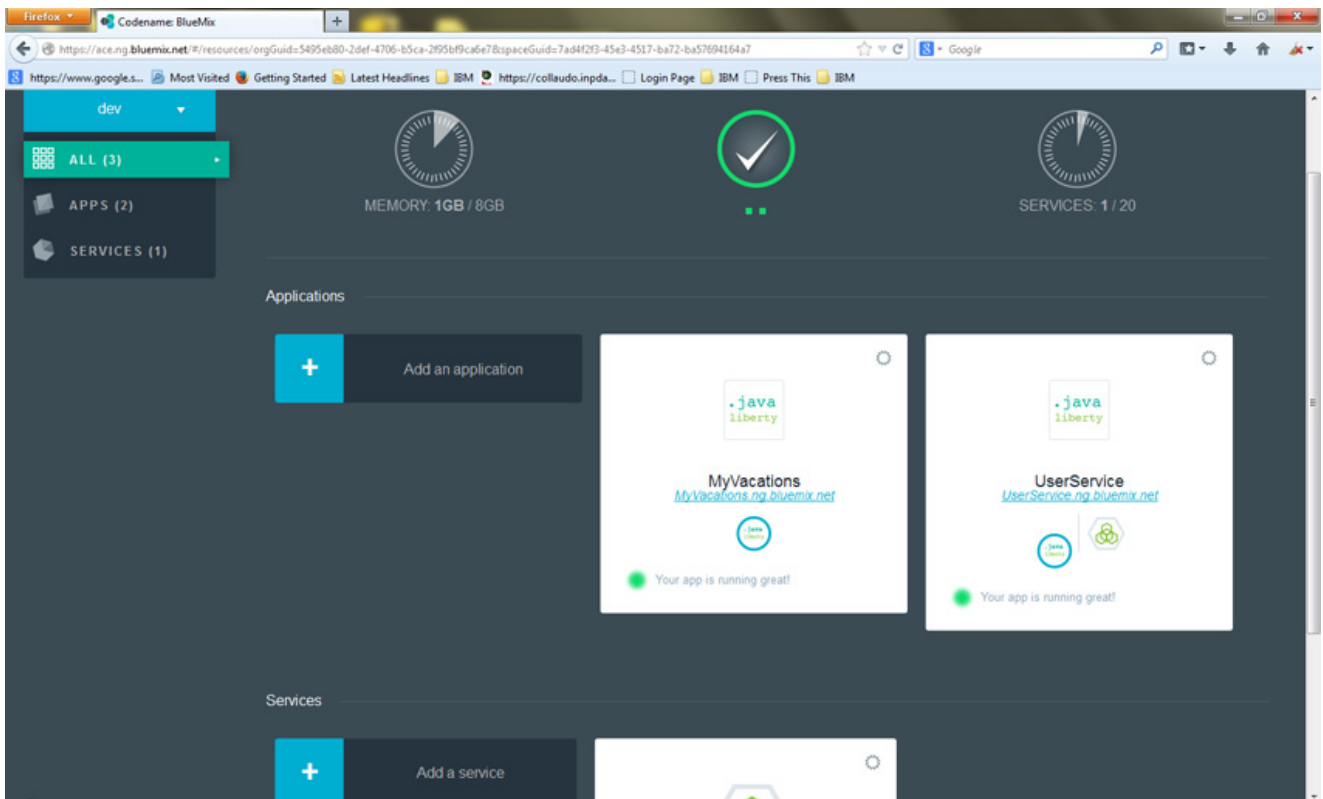
The UserService functions are:

1. The logging function — The UserService receives the user name and password, and searches for the user in the database. If it finds the user, UserService returns it; otherwise, it creates a new record.
2. The profiling function — The UserService receives the user name and searches the user information (preferred location, number of adults, number of children) and returns them to the client.

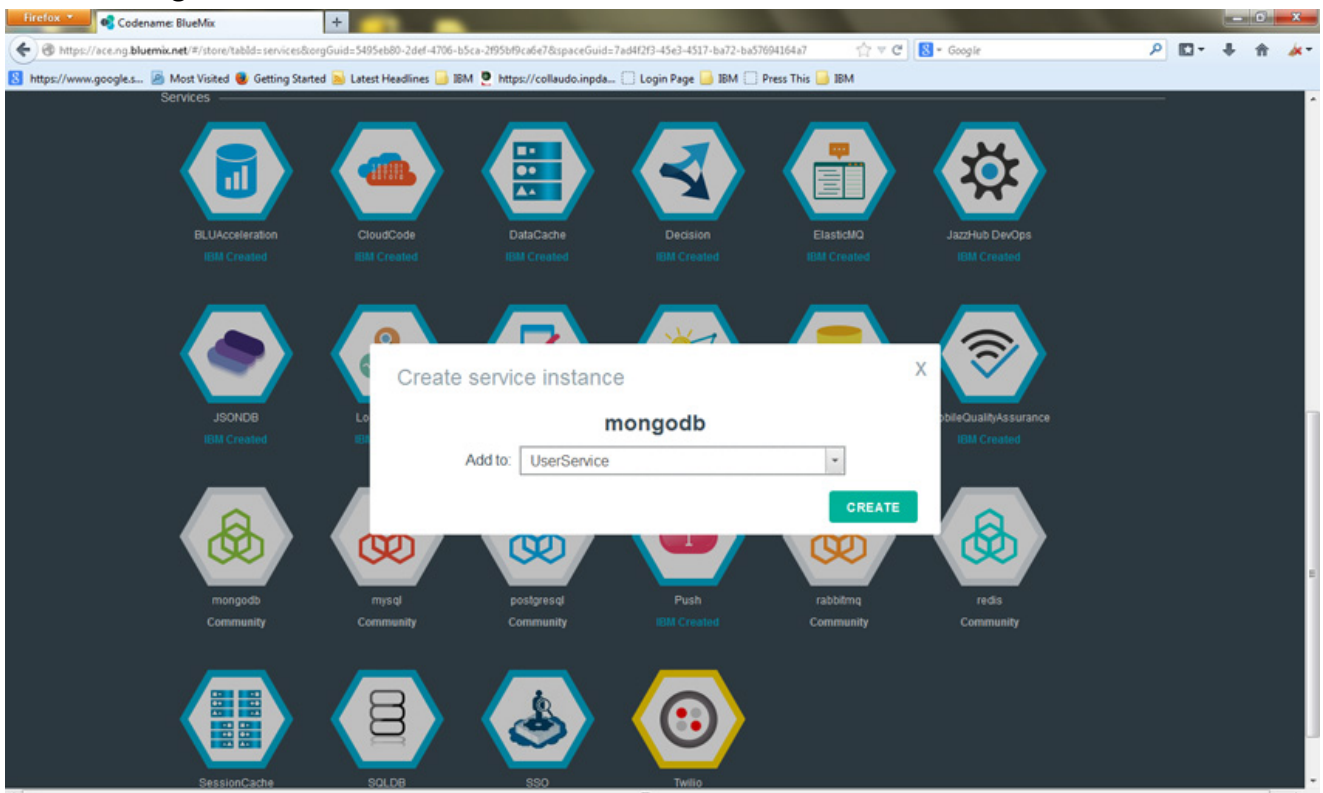
## Step 4. Bind a cloud service (MongoDB)

To use a MongoDB service, you must first create an instance of the service:

1. Connect to Bluemix and select **Add a Service** in the dashboard view.



2. Select **MongoDB** from the list of available services and create the service instance.



Alternatively, you can use the command line:

```
cf create-service mongodb 100 mongodb_ser1
(USAGE:  cf create-service SERVICE PLAN SERVICE_INSTANCE)

cf bind-service UserService mongodb_ser1
(USAGE:  cf bind-service APP SERVICE_INSTANCE)
```

Now a MongoDB service instance is ready and bound to UserService.

## Step 5. Use MongoDB service in the application

After creating the service and associating it to your application, its configuration is added to `VCAP_SERVICES` as a read-only environment variable that contains information you can use in code to connect to your services. In this case, you have:

```
{
  "mongodb-2.2": [
    {
      "name": "mongodb-ser1",
      "label": "mongodb-2.2",
      "plan": "100",
      "credentials": {
        "hostname": "10.0.116.106",
        "host": "10.0.116.106",
        "port": 10192,
        "username": "46c538a6-e6e1-4d02-8132-77b0a4b2dc1c",
        "password": "0ceea0ea-5548-46ad-9b09-1002683aeca7",
        "name": "946dc87b-b455-4d12-b977-1b1ee22f1ade",
        "db": "db",
        "url": "mongodb://46c538a6-e6e1-4d02-8132-77b0a4b2dc1c:
0ceea0ea-5548-46ad-9b09-1002683aeca7@10.0.116.106:10192/db"
      }
    }
  ]
}
```

To connect to the MongoDB service instance, using the `VCAP_SERVICES` variable, you extract `"url"` `JsonNode`. Notice that the URL contains all the parameters to connect to the database (user credentials, host name, port, DB name).

```
private static String getUrlConnection() {
    String env = System.getenv("VCAP_SERVICES");
    ObjectMapper mapper = new ObjectMapper();
    try {
        JsonNode node = mapper.readTree(env);
        Iterator<JsonNode> dbNode = node.get("mongodb-2.2").getElements();

        JsonNode cred = (JsonNode)dbNode.next().get("credentials");

        String uri = cred.get("url").getTextValue();

        logger.debug ("url db: " + uri);

        return uri;
    } catch (JsonGenerationException e) {
        logger.debug(e.getMessage());
    } catch (JsonMappingException e) {
        logger.debug (e.getMessage());
    }
}
```

```

    } catch (IOException e) {
        logger.debug (e.getMessage());
    }

    return null;
}

```

You can create a Spring configuration class to create a DB connection, using the MongoDB Java driver, to return the DB object to the client.

```

@Configuration
public class MongoConfiguration {

    public @Bean DB mongoDb() throws Exception {
        MongoClientURI mcUri = new MongoClientURI(getUrlConnection());
        MongoClient mc = new MongoClient(mcUri);
        mc.getDB(mcUri.getDatabase());
        return mc.getDB(mcUri.getDatabase());
    }
}

```

The UserManager class uses the MongoConfiguration to interact with DB. The `init` method gets the "users" collection or creates one if it doesn't exist.

```

private void init(){
    ApplicationContext ctx =
        new AnnotationConfigApplicationContext(MongoConfiguration.class);
    db = (DB) ctx.getBean("mongoDb");
    coll = db.getCollection("users");
}

```

Let's say that MongoDB is not a relational DBMS but oriented to the document. This means that instead of having tables, we have collections; instead of having rows (or tuples), we have documents; and instead of columns, we have fields. These fields are not predefined, as is the case for the columns in a table. You can enter in a collection any kind of data. To find a document, you must create a `BasicDBObject`.

```

BasicDBObject user = new BasicDBObject("username", userData.getUsername());

return (DBObject)coll.findOne(user);

```

UserController is the UserManager client, and it uses its functions to get and save the logged-in user information.

```

@RequestMapping(value="/user", method = RequestMethod.GET)
public @ResponseBody String getUser(@RequestParam("username") String username,
    @RequestParam("password") String password) {
    logger.debug("BEGIN: controller getUser - username:" + username);
    UserManager userManager = new UserManager();
    BasicDBObject user = (BasicDBObject) userManager.getUser(username, password);
    logger.debug("END: controller getUser - user:" + user);

    return user.toString();
}

```

## Step 6. Deploy a Spring application

MyVacations uses UserService to get user information to profile the search with the values that the UserService has saved during the user's last login. The user can visualize, on a map, a set of hotels resulting from the search. Also in this case, the controllers are not configured in the xml config file but are dynamically detected by the Spring Framework, thanks to this directive contained in the servlet configuration file.

```
<context:component-scan base-package="com.myvacations.app" />
```

The first controller, `HomeController`, is called to display the login page.

```
/**
 * Simply selects the home view to render login page.
 */
@RequestMapping(value = "/", method = RequestMethod.GET)
public String home(Locale locale, Model model) {

    return "login";
}
```

The `HotelsController` activates upon submission of the login page.

This controller, called through an HTTP `get` request with the "username" parameter, accesses UserService to get any user preferences saved during the last access by this user.

This controller uses the RestTemplate to make a RESTful call to UserService. RestTemplate is a helper Spring class for client-side HTTP access. The objects are passed to and returned from the methods `getForObject()`, and are converted to HTTP requests and from HTTP responses by [HttpMessageConverters](#). In our sample, this class is used to call the RESTful services, such as UserService.

```
@RequestMapping(value = "/hotels", method = RequestMethod.GET)
public String getHotels(@ModelAttribute("username") String username, Model model) {

    logger.debug("BEGIN HotelsController: username=" + username);

    RestTemplate restTemplate = applicationContext.getBean("restTemplate",
        RestTemplate.class);

    user = (UserData) restTemplate.getForObject(new
        URI("http://userservice.mybluemix.net/userpref?username="+username),
        UserData.class);
}
```

## Step 7. Integrate services

You will want to integrate services to bring more data to MyVacations. I'm going to provide two examples of integrating services:

- The Expedia RESTful service to bring in information on hotels.
- The Google map service to visualize the hotels on a map.

Let's integrate the Expedia web services first to get the information about hotels. First, you must go to [Expedia API Developer](#) to register for a developer account and an API key.



The `SearchController` activated by the **Submit** button click is an Ajax call in `Hotels.jsp`.

```
$.get('/search',
    $("#ricerca").serialize(),
    function (data, status) {
        if (status == 'success') {
            if (data.toString()==""){
                $("#map_canvas").hide();
            }
        }
    });
```

In `SearchController`, there is the call to the `ExpediaClient` to get the `HotelSummary` List.

```
List<HotelSummary> response=null;
response = (new ExpediaClient()).getHotels(
    location, dateFrom, dateTo, numAdults, numChildren);
```

The `ExpediaClient`, using the user information obtained by `UserService`, extracts the `ExpediaObjects` decoding the `Expedia` JSON response.

## Listing 1. Extracting a list of hotels

```
ExpediaObjects hotels= (ExpediaObjects)
restTemplate.getForObject(buildQueryHotelsURL(location, dal, al, numAdulti, numBambini),
ExpediaObjects.class);
```

The sample then uses Google map service to visualize, on a map, the hotel list obtained from `Expedia`.

The Google Maps API lets you embed a Google map image on your web page. Before you start, you will need a specific API key from Google. The key is free, but you must create a Google account.

```
<script src="http://maps.googleapis.com/maps/api/js?
key=YOUR_API_KEY&sensor=TRUE_OR_FALSE"></script>
```

For interacting with the Google Maps API, I chose `jQuery-ui-map`. This is a good jQuery plugin to embed maps in web and mobile applications. It allows you to view maps and markers, and to take advantage of advanced services and the management of trails, the view in street-view mode, and dynamic loading of geographic data represented in JSON.

After you create a `div` or another similar HTML container, it is time to launch `gmap` — the key method of the plugin that allows us to invoke the functions of the Google Maps API — to feed to the display the coordinates of a marker reference on the map.

```
var map = $('#map_canvas').gmap({
    'center': new google.maps.LatLng(data[0].latitude, data[0].longitude),
    'minZoom': 5,
    'zoom': 8
});
```

We have created a map centered on the first hotel's geographic coordinates.

Now, for every hotel in the result list, we create a marker and just place it on the map.

```
$.each(data, function (i, m) {  
    $('#map_canvas').gmap('addMarker', {  
        'position': new google.maps.LatLng(m.latitude,  
                                             m.longitude),  
        'bounds': true  
    }) .click(function () {...
```

Register a function on-click event to load an info window with the short hotel description.

```
$('#map_canvas').gmap('openInfoWindow', {  
    'content': descr  
},
```

## Step 8. Push the application to the cloud

After building and creating the applications, we are ready to deploy them on Bluemix. Deployment is an automated matter of moving the apps from the local VM to a cloud-based VM.

Use the Cloud Foundry CLI `cf push` command to initiate the deployment. (In Cloud Foundry documentation, the deployment process is often referred to as *pushing* an application.)

```
cf push MyVacations -path MYDIR\MyVacations.war
```

The `push` command performs a variety of staging tasks, such as finding a container to run the application, provisioning the container with the appropriate software and system resources, starting one or more instances of the application, and storing the expected state of the application in the Cloud Controller database.

## Step 9. Test the application

For testing purposes, you'll want to do a simple run test on the application, then you'll want to test to see if the application is portable.

For simplicity, the login page allows the access to any user name and password, and if the user is not yet present in the system, it is created.

After you log in, go to the search page, insert the search parameters and click **Search**.

### Where are you going?

Location:

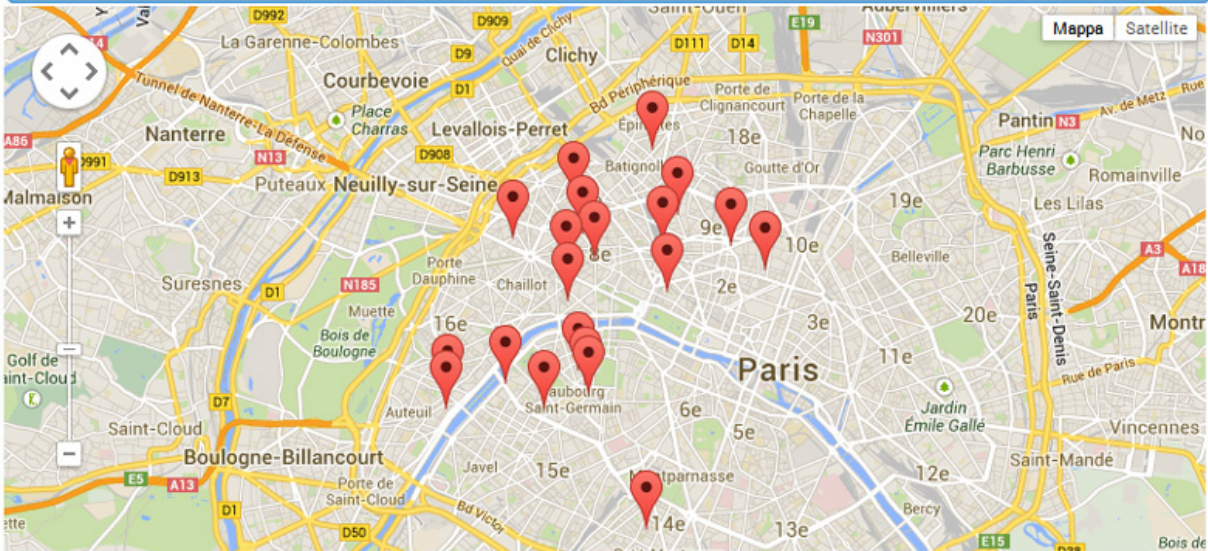
Arrival:

Departure:

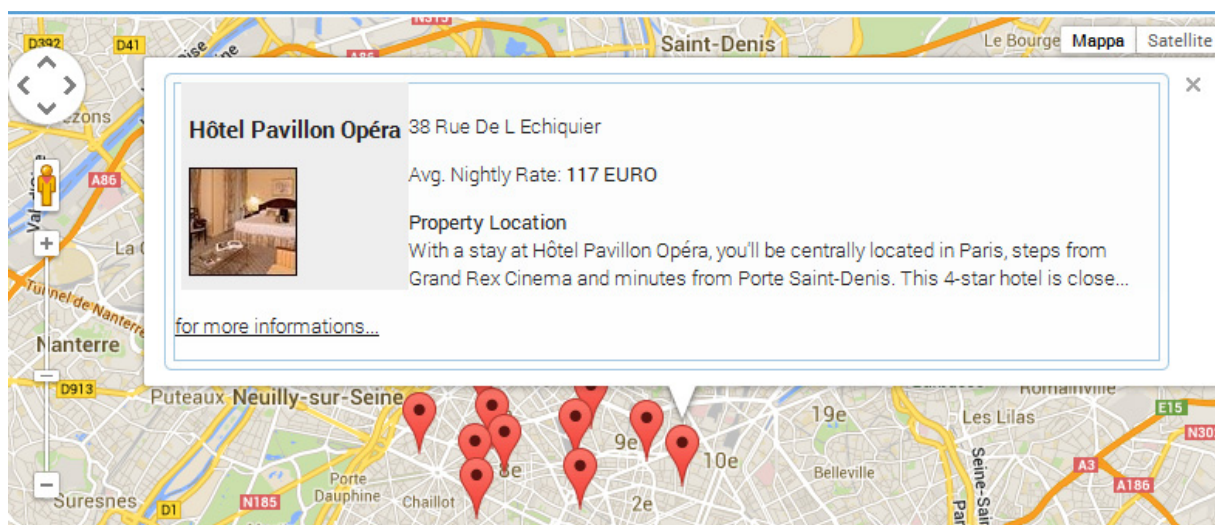
Number of Adults:

Number of children:

### Results



If you click one of the markers, brief information about that hotel is displayed.



To test the portability of my application on different cloud platforms, I chose to deploy MyVacations on Pivotal platform and Google App Engine.

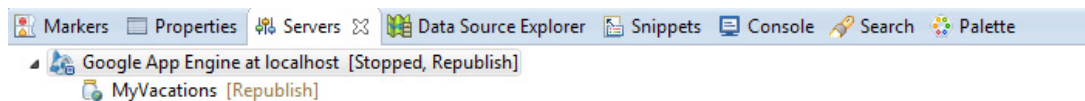
To deploy on GoogleAE, the tools and technologies used are:

1. Google App Engine Java SDK 1.8.8
2. Spring 3.1.1
3. Eclipse 4.2+ Google plugin for Eclipse

Because Google App Engine supports the Java web application based on Spring Framework, my application doesn't need any changes.

The Google App Engine SDK (installed on Eclipse) includes a web server for testing your application in a simulated local environment, so you can test the application without a Google user account. (It is also possible to run the application on a remote Google server).

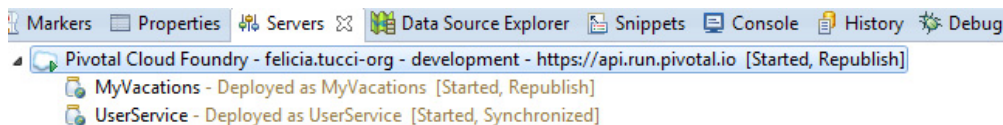
The Google plugin for Eclipse adds items to the **Run** menu for starting this server. In this scenario, the MyVacations installed on Google App Engine calls the UserService application installed on Bluemix through its RESTful API, which demonstrates the high level of portability of an application that doesn't use internal platform services.



The tools and technologies to use to deploy on Pivotal are:

1. Spring 3.1.1
2. JDK 7
3. Spring Tool Suite 3.4.0+ Cloud Foundry Integration for Eclipse 1.5.1

The Cloud Foundry plugin for Eclipse enables you to deploy on the Pivotal platform. In this way, you can test the application directly on the target environment, without leaving the IDE. You need a valid Pivotal user account.



On this platform, you can deploy the application with no changes, and you can deploy the UserService application with minor changes.

## Conclusion

This app shows just some of the possibilities of integrating internal and external services with a cloud application. I took advantage of some of the positive attributes that ClueMix offers:

- Reduced provisioning needs (app or infrastructure)

- Scalability
- Easy integration of internal services
- Eased management
- Portability on similar cloud platforms

On the portability issue, because Bluemix is based on Cloud Foundry, you have the freedom to move it to other platforms. Let me demonstrate portability with two examples:

- Deploying MyVacations to the Pivotal cloud requires no changes. Pivotal is also based on Cloud Foundry, so the compatibility is almost total. UserService uses a MongoDB service on the Bluemix platform; on Pivotal, there is a similar MongoDB service available, but you must change the URL connection from the VCAP\_SERVICES variable in the MongoConfiguration class.
- For a non-Cloud Foundry-based cloud (say, in this case, Google's cloud), it's still pretty easy to deploy MyVacations. (MongoDB is not among the services provided by the Google platform, so I chose another solution as the "Big Table" service, which — unlike MongoDB — is a proprietary solution for data persistence.) You simply add `appengine-web.xml` file to the `web-inf` directory to enable the Google Application Engine, like so:

```
<?xml version="1.0" encoding="utf-8"?>
<appengine-web-app xmlns="http://appengine.google.com/ns/1.0">
  <application>_your_app_id_</application>
  <version>1</version>
  <threadsafe>true</threadsafe>
</appengine-web-app>
```

On a final note, you must design a cloud application to exploit the possibilities of a distributed platform and provide reliable, efficient, and fast service. For the most part, I think I accomplished this with the MyVacations app. Of course, when UserService gets hit with a large number of requests, it could result in slow response times for users. However, you can experiment with performance tweaks for that (like using asynchronous messaging to decouple the components so a task isn't blocked until a response is received). There are several performance tuning tricks to apply to this type of application, have a good time experimenting with them.

## Acknowledgment

I want to thank Fabio Castiglioni for his encouragement and review of this article.

## Related topics

- [See IBM Bluemix in action](#)
- [Cloud computing](#)
- [Java technology](#)
- [MongoDB](#)

© Copyright IBM Corporation 2014

([www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml))

[Trademarks](#)

([www.ibm.com/developerworks/ibm/trademarks/](http://www.ibm.com/developerworks/ibm/trademarks/))