An interactive user environment makes learning based on digital games attractive and efficient. And the data collected can help game developers improve the game’s educational value. See how data collected from the Oculus Rift virtual reality device can be streamed to a Bluemix server for analysis. The BlueMix application stores all data from the Rift, such as the user’s head movement, which can be analyzed to gain insight into the learning rate.

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.
Virtual Reality (VR) is the technology of the moment, and VR devices such as the Oculus Rift have gained prominence in technology companies, including Sony and Facebook. The way that users interact with VR is simple. The device captures movements from the player's head and sends this data to a computer, which creates an immersive, interactive virtual environment. In addition, this captured data can be stored for further analysis to improve the user experience. Bluemix is the perfect platform for the task.

This tutorial describes how to create a simple application using Bluemix and vr.js, a browser plug-in. This application captures the quaternion positions (x, y, z, and w) from the Oculus Rift and sends it to Cloudant, a database as a service (DBaaS). You can deploy this application to Bluemix and use it by plugging in the Oculus Rift and opening a browser.

“Create a flexible VR system that can be extended for many new types of applications, by integrating Node.js, Cloudant, and the Oculus Rift.”

Run the app
Get the code

What you need to build your app

- DevOps Services account linked to your IBM ID
- Bluemix account
- Express framework and nano
- Cloudant database
- vr.js browser plug-in
- Basic understanding of Node.js

Step 1. Fork this project

Click this tutorial's Get the code button and fork the project on DevOps Services.

1. Click EDIT CODE (enter your DevOps Services credentials if you're not already logged in).
2. Click the FORK button on the menu to create a new project.
3. When you see the prompt, choose a name to give to your new forked repository. We chose rift2cloudant.
4. Check Deploy to Bluemix.
5. Click SAVE to create a copy of the code into your project and a Git repository.

You can make this project private by clicking the Make it private (not public) checkbox. Leave it unchecked for now.

Configure a host and name for your application

To deploy this code to Bluemix, you need to set up some configurations so you can define a new host and name for your application.
1. Click **BUILD & DEPLOY**.
2. On the page that opens, select **OFF**. You don't want to deploy the changes made in your repository automatically.
3. Click **EDIT CODE**.
4. Click **manifest.yml**.

You need to change the existing host and name to a unique name for your server. We changed the original **host** and **name** designations to **rift2cloudant**.

**Commit changes**

Go to the git status page by clicking **Git Repository**:

Next, stage the modifications to commit and then push them to the repository.

1. In the text box, type the modifications you made.
2. Click **Select All** to add the files to commit.
3. Click **COMMIT**:

4. Click **Push**.
5. Go back to **Show Current Folder**.
6. Click **DEPLOY**.
7. In the pop-up, press **Deploy** to deploy the application.

**Add a Cloudant service**

If the application deploys successfully, you have a new application started within your Bluemix account. To add a Cloudant service:
1. Click the new app that appears on your dashboard. You should see a red dot that indicates the app is not running.

2. Click **ADD A SERVICE**.

3. Scroll down until you find **Cloudant JSONDB** and select it.

4. In the **Name** field, type `cloudant` and press **Create**. If you want to use a different name for the Cloudant service, be sure to also change the code inside the app.js. Look for this code:

   ```javascript
   var cloudantServiceName = 'cloudant';
   ```

   A pop-up appears and tells you to restart your app.

5. Click **OK**.

### Step 2. Clone and push

You want to keep a local repository of the application so that when you modify the code, you can push the changes into Bluemix. To begin:

1. Go to the main page of your project in DevOps Services and copy the git URL:

2. To clone the project to your computer, open a terminal (if you use Windows, we suggest git bash), type `git clone` and then paste the link that you just copied (for example, `git clone https://hub.jazz.net/project`).

3. Go back to your Bluemix account, click your application, and then open the **VIEW QUICK START**. This window appears:
4. In a terminal, go to your cloned folder and follow the steps one, two, and three that are listed there.

**Push changes to Bluemix**

Whenever you want your modifications to be deployed into your application in your Bluemix account, you need to push the modification.

1. Open a new terminal.
2. From inside the cloned repository, type `cf push -c "node app.js"`.

Your application should be working now. A green dot appears beside **Running**.

**Step 3. Prepare the DBaaS (Cloudant)**

First, create the Cloudant database. To begin:

1. Open the Bluemix dashboard, click on your application, and then click on the Cloudant service. A new page opens, showing information about the service and how to get started.
2. Click **LAUNCH** to access the service. Make sure that you are in the **Databases** section (marked in orange).
3. This section displays all your databases. You want to create a new database to be used in your application, so click **Add New Database**.

4. In the pop-up window, at **Name of database**, type `users` to create a database called users.

5. Next, create a new database called `riftbase`. Follow the same steps: Click **Add New Database** and, when prompted, type in `riftbase`.

### Step 4. Understand and configure the application

You still need to install the plug-in. First, however, before you go to the application itself, you should understand the most important sections of the code.

This code is inside the app.js file:

```javascript
var cloudantServiceName = 'cloudant';
```

This is where the program will look for the Cloudant service in Bluemix. If you followed the previous steps and created a service called `cloudant`, you don’t need to modify that. However, if you created a service with a different name, you just need to change the code to the name you chose. Also, you should have two databases, `user` and `riftbase`, created in Cloudant. These databases will be invoked in two parts of the code:

```javascript
var db = nano.db.use('riftbase');
```

and

```javascript
var db = nano.db.use('users');
```

If you use different names for your databases, you should modify this code.

**Sending data to Cloudant**

This code, located in app.js, creates a post handler so it can send information to be stored in our Cloudant database:

```javascript
app.post('/endpoint', function(req, res){
    res.send(req.body);
    db.insert({position:req.body},function(err, body, header) {
        if (err) {
            res.send("Error creating file");
            return;
        }
    });
});
```

This post request is triggered every second by a function called `myTimer` (you can find `myTimer` inside the oculus.ejs code, which is the modified raw_data.html code located in the views folder):

```javascript
function myTimer() {
  if(state.hmd.presents){
    var data = {};
    data.x = state.hmd.rotation[0].toFixed(2);
    data.y = state.hmd.rotation[1].toFixed(2);
    data.z = state.hmd.rotation[2].toFixed(2);
    data.w = state.hmd.rotation[3].toFixed(2);
    data.seconds = (endTime - startTime)/1000;
    data.seconds = (data.seconds).toFixed(0);
    $.post("/endpoint", data, function(response){
      console.log(data);
    });
  }
}
```

This code for the users page is placed inside app.js. It works by sending a Name and an Age to the users database on Cloudant:

```javascript
app.post('/insert',function(req, res){
  var db = nano.db.use('users');
  var name = req.body.name
  var age = req.body.age
  db.insert({name:name,age:age},function(err,body,header){
    if(err){
      console.log(err);
      return res.status(500).send(err.message);
    }
    res.send("Contact was created successfully");
    console.log("success");
  });
});
```

### Step 5. Install the vr.js plug-in

The vr.js plug-in lets your Oculus Rift work using the browser (Chrome or Firefox). You need to **install the plug-in** to open the main page of your application, but you can still open the users page without it. (The main and users pages are covered in the following step.)

The code used in this Oculus Rift application is a slight modification of the code that can be found on [GitHub](https://github.com).

Now, within the app.js file, we can write a `get` method that renders the oculus page, which uses the vr.js plug-in:

```javascript
app.get('/', function (req, res){
  res.render('oculus');
});
```

### Step 6. Run the application

The application has two pages: the main page and the users page.
Main page

When you open the main page of your app, you are sent to a web page showing data captured from the Oculus Rift that is connected to the computer. If your Rift is connected, the page should look something like this (be aware that this page uploads the quaternion position to the riftbase database in Cloudant every second):

```
oculus rift detected
name: Oculus Rift DK1 v0 (Oculus VR)
desktop position (px): 0,0
resolution (px): 1280x800
screen size (m): 0.14976x0.0936
screen center (m): 0.0468
eye-to-screen (m): 0.041
lens separation (m): 0.0635
IPD (m): 0.064
distortion k: 1.0,0.2199999988079071,0.2399999946358197,0,
chromaAbCorrection[0]: 0.995999918937683
chromaAbCorrection[1]: -0.00400000189989805
chromaAbCorrection[2]: 1.0140000581741333
chromaAbCorrection[3]: 0
rotation:
  0.16
  -0.00
  0.03
  0.99
```

The oculus uses the quaternion system, which is the last four-element vector displayed in bold in this page. The quaternion system is used to encode any rotation in a 3D coordinate system.

You could store any value from this page. In our application, only the last four elements (quaternion values) are stored. You can check the `myTimer` function again to understand how we retrieve these values from the headset and send them with the `post` method.

READ:  [More about quaternions](#)

Users page

If you don't have a Rift, or something went wrong and you want to make sure that your application is working, you can access the users page by adding "/users" to the end of the URL (for example, http://bluemix.net/users). This page is just an example for sending some data to Cloudant; it doesn't require the Rift. Fill in the boxes and click **Upload to Cloudant**. Afterwards, check your Cloudant database to see if your information uploaded successfully.
Step 7. Validate data collection and query data stored in Cloudant

The last thing to do is check whether the data was uploaded to the Cloudant database. To show the complete document for each database, access the link of your Cloudant database and add this to the URL: _all_docs?include_docs=true.

Similarly, if you click Edit doc:

the data loads in the page:
Conclusion

By integrating technologies such as Node.js, Cloudant, and the Oculus Rift, you created a flexible VR system that can be extended for many new types of applications. While VR devices provide data about the player, Bluemix offers many tools to store and analyze it. Try using Bluemix for other services. Use your imagination and merge services such as SQL Database, MySQL, Analytics Warehouse, and more.

The Cloudant NoSQL service provides access to a fully managed NoSQL JSON data layer that's always on.

RELATED TOPICS: Node.js Cloudant Oculus Rift