Getting started in a Cloud Foundry project can be a bit daunting for the new user. This article lays out the entire process of getting a simple Java servlet application working in the CenturyLink Cloud platform. I will walk through the entire process, from signup to working code. The article assumes that you are familiar with Java servlet programming, and the MySQL database.

The example is a Java application. IT IS NOT DESIGNED TO FUNCTION ON YOUR LOCAL MACHINE. You can run it, and the pages will come up, but the interaction is intended to function within the AppFog environment. To create similar code for your own use, you will need a few java libraries. The code was built on Java 1.7. You would need:

* The MySQL jdbc connector, which is available here: <https://dev.mysql.com/downloads/connector/j/5.1.html>. This needs to be installed in your JAVA\_CLASSPATH.
* The org.json library, which is available here: <http://central.maven.org/maven2/org/json/json/20140107/json-20140107.jar>. This is the version for Java 1.7. This also needs to be installed in your JAVA\_CLASSPATH

Again, you don’t need these for this exercise.

You need to install the Cloud Foundry Command Line tools, which are obtained here: <https://github.com/cloudfoundry/cli/releases>. Download and install the correct version for your local machine.

The overall process is:

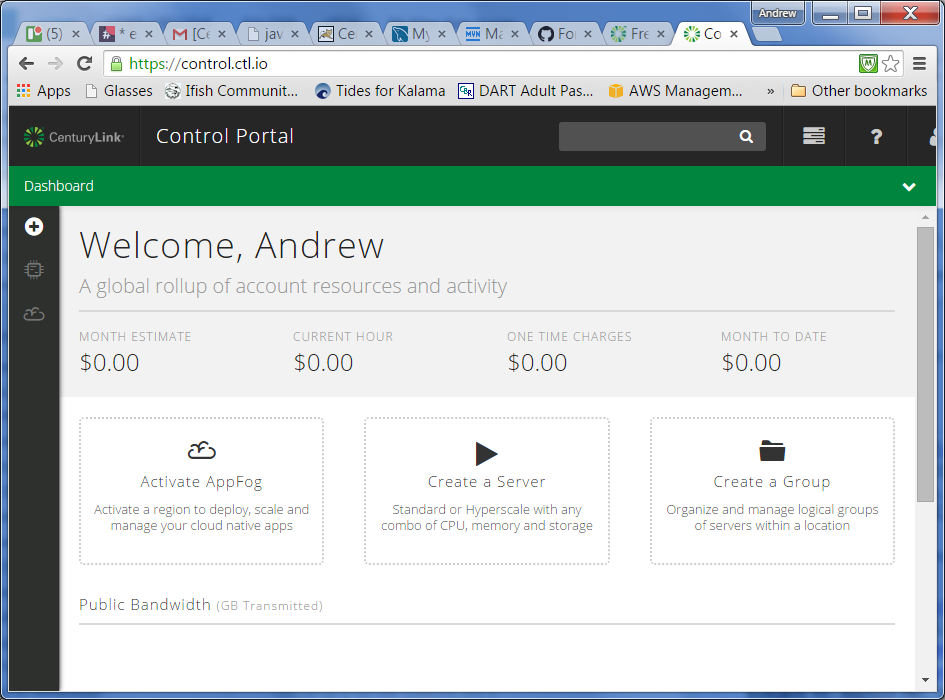
1. Create a CenturyLink Cloud account
2. Download and install the Cloud Foundry command line utilities
3. Download the example code
4. Log into AppFog
5. Push the code into AppFog to establish your instance
6. Define a database service instance
7. Set up a database in the service instance so that we have something to connect to
8. Register the database service with your application
9. Access the application to test
10. Review the results

Are you ready? Let’s start…

**Create A CenturyLink Account:**

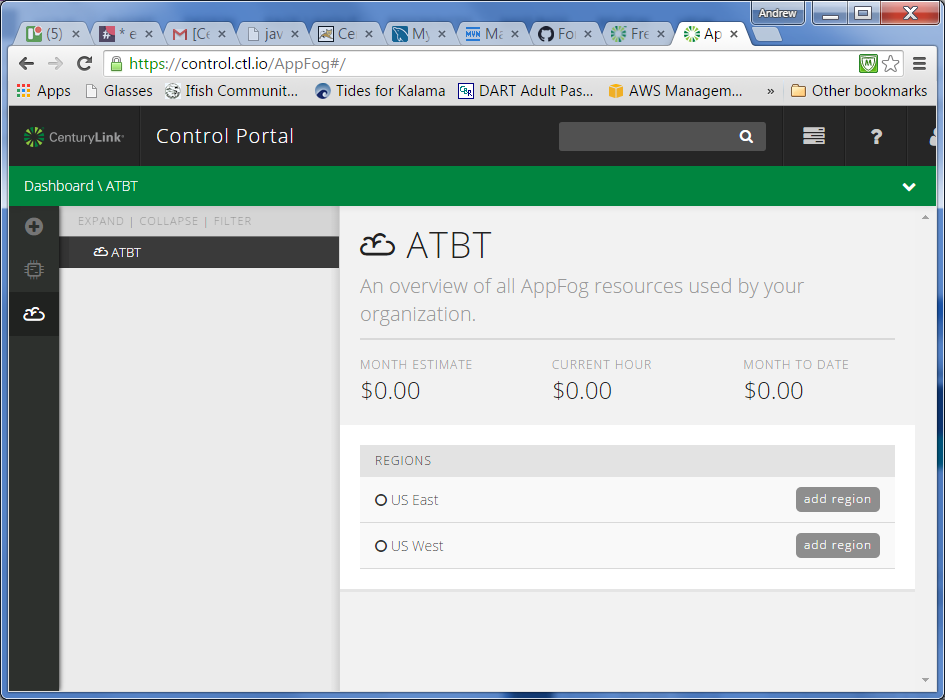
Access <https://www.ctl.io/>. Click the button to start a free trial. Enter an email address and phone number. You will be required to enter a credit card, but you will not be charged for your first 30 days’ usage. If you wish to cancel your trial without cost, <WHAT DO WE DO?>

At the end of the signup, you will receive an email with your username and a system defined password. Go to <https://control.ctl.io> and login with these credentials. When you do that, you will be given a chance to change your password, and then you should see this page:

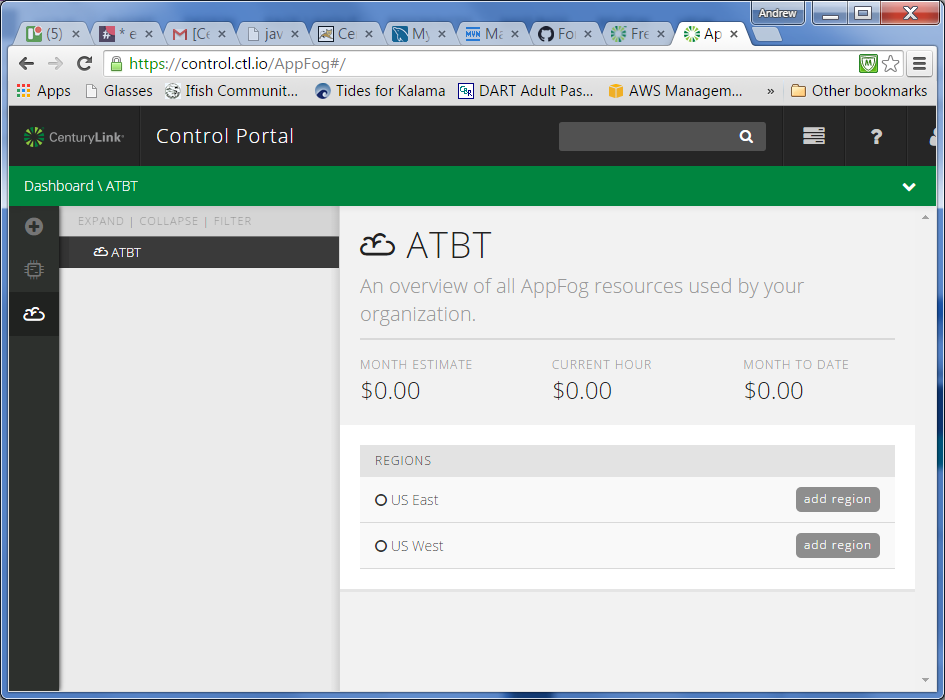


Select “Activate AppFog”.

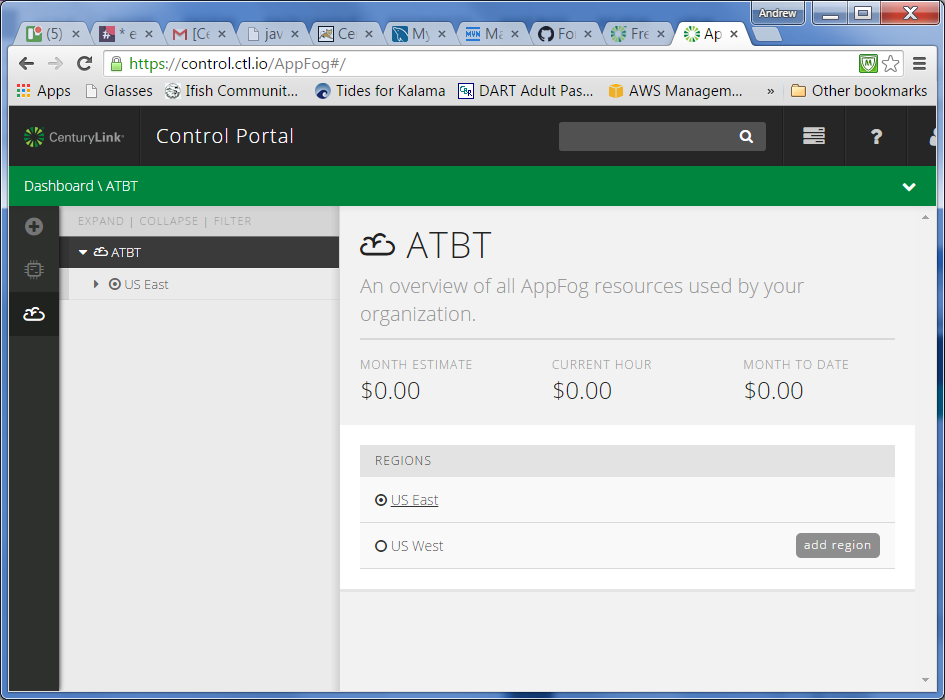
Now the page will update so that the fog icon on the left is no longer greyed out:



Select that icon, and you will see the following page:

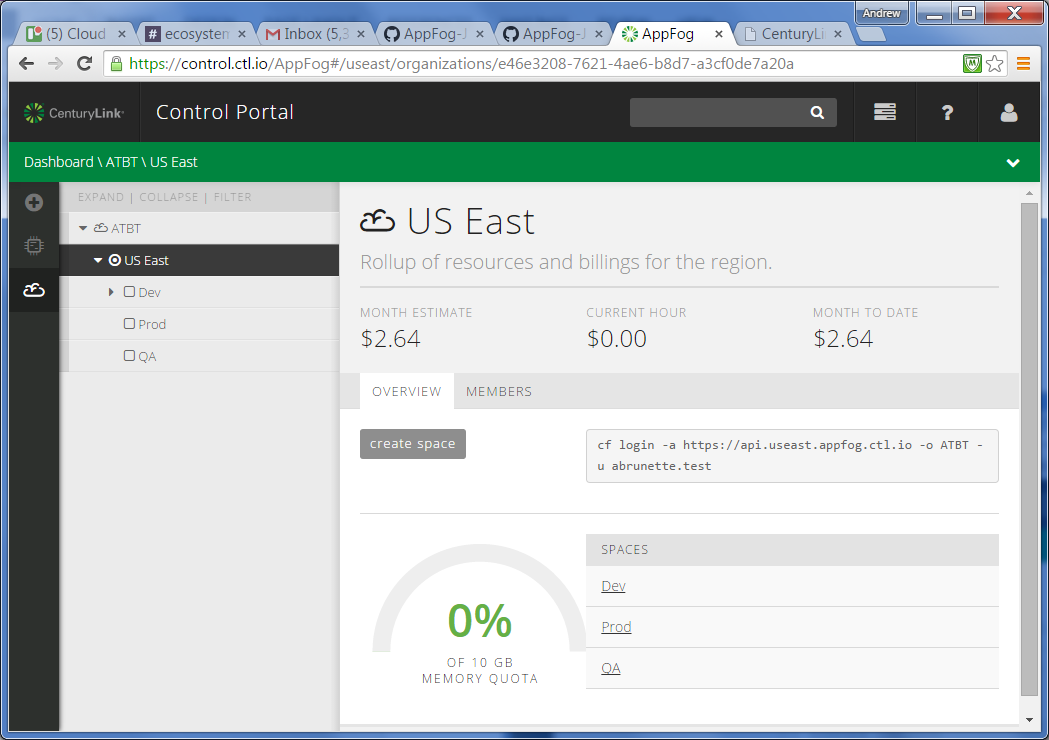


Click the “add region” button for US East. The radio button will highlight, and the page will look as follows:



Select the “US East” link.

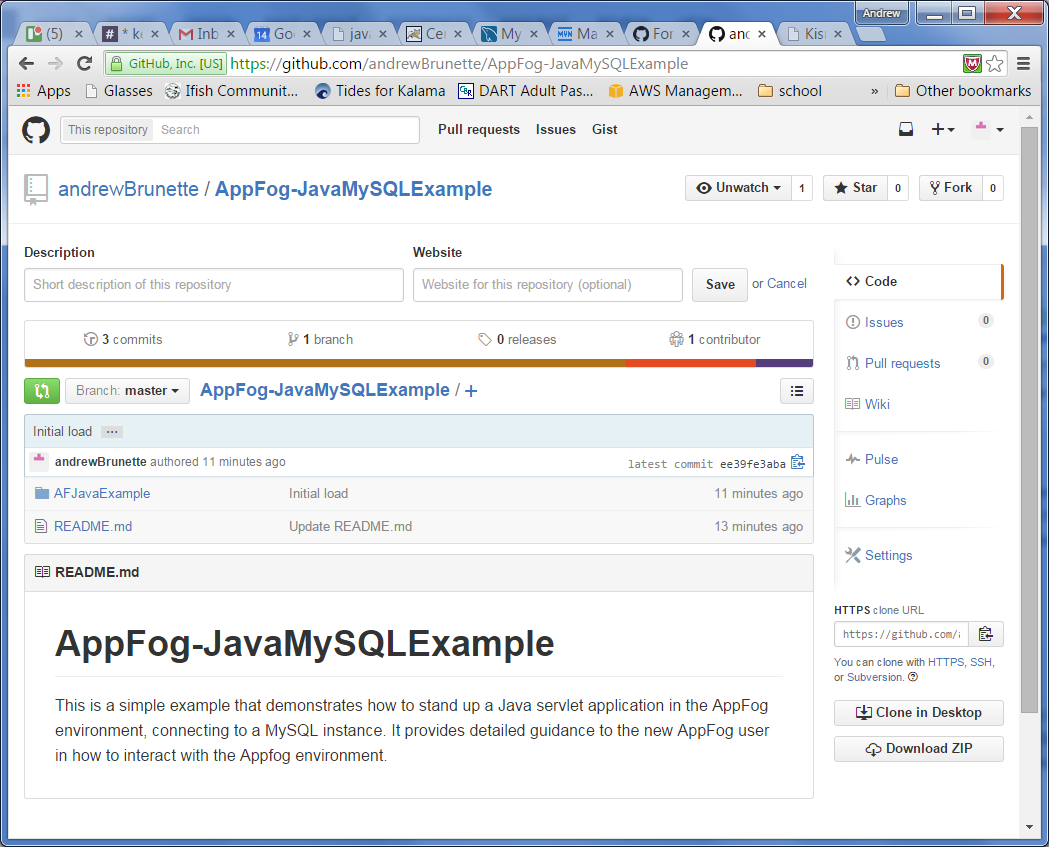
The will bring you to this page:



The “cf login” string will have your account information, with your alias (“ATBT” above) and username (“abrunette.test” above”). We’ll use that in a bit. At this point, you have successfully setup your AppFog instance. Good work!

**Download The Example Code:**

The next step is fetching the example code from GitHub. The example application is a simple servlet app that allows you to see your database configuration variables, and test connection to a database. It therefore demonstrates the code needed to do so. The example code is hosted at <https://github.com/andrewBrunette/AppFog-JavaMySQLExample>. <TODO correct URL> Access this link in your web browser. It will bring up this page:



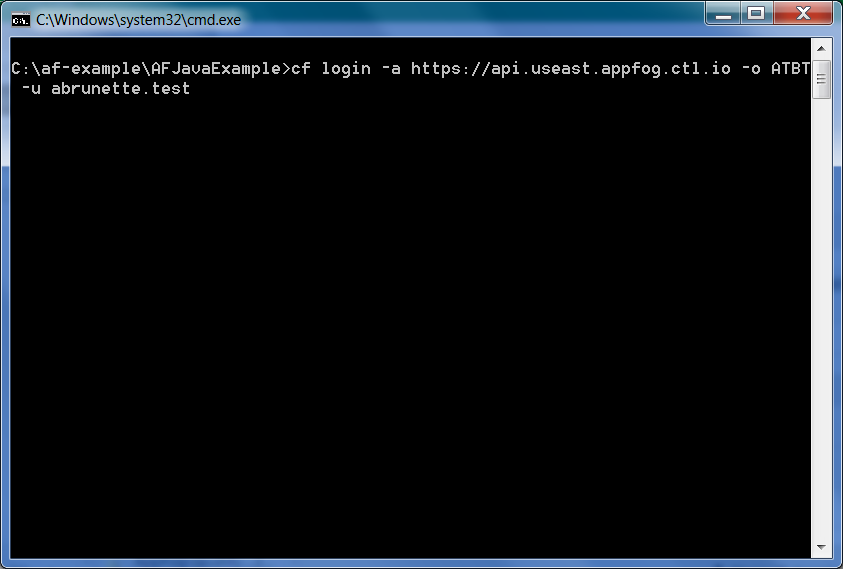
<TODO – update graphic with correct URL>

The appearance of the page may change over time, but the instructions are the same. We will omit the robust GitHub cloning approach, and download the repository as a zip file, using the “Download Zip” button in the lower right of the page. Unzip this file into an empty directory of your choice.

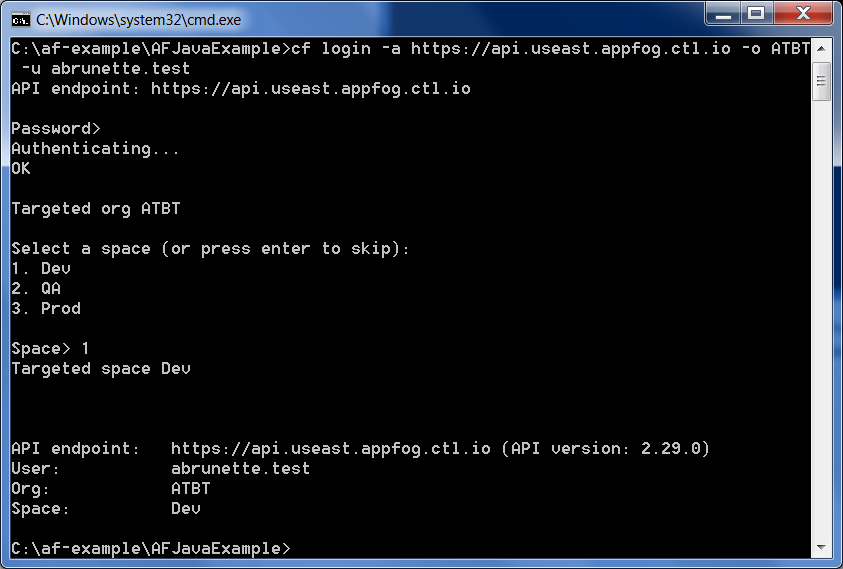
**Log Into AppFog:**

AppFog is based on Cloud Foundry, an open source environment used to build Platform as a service environment. As such, the command line environment is the common Cloud Foundry environment, and any errors you see can be searched using Cloud Foundry as a search term.

Open a command/terminal window, and change your current directory to the directory where you saved the example code. Enter the login string as you saw in the last page of the control portal, similar to what is shown below:



The system will ask for your password, and to select a space for the app to run in. The password is the same as your CenturyLink account password. Choose 1 for “dev”. The result should look similar to:

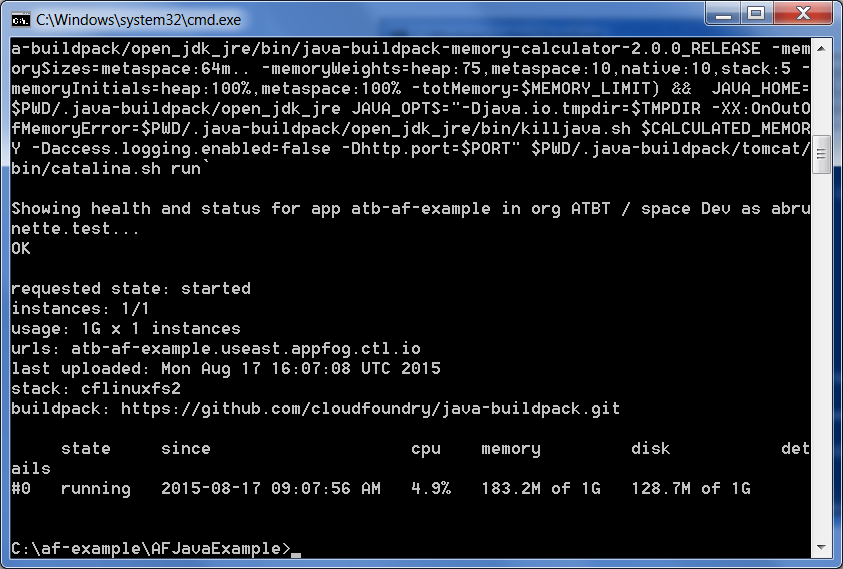


Now we are going to push the sample application from our local machine into Appfog. This moves the code into AppFog and configures an instance to support the application. The instance is an empty execution environment. It needs to be configured to support our application. This is done by specifying a ‘buildpack’. The buildpack configures the execution environment to support our application, in this case by provisioning Java and Tomcat. There are buildpacks available for Java, Ruby, Python, Go, Node.js, and other environments. Since it is an open standard, you can also build your own, but that is out of scope for this article.

You need to specify a unique name for your example. I will be using “atb-af-example”. This name must be globally unique within appfog, as it is used to form the URL that is used to reach your application.

Make sure you are in the directory that contains the various application directories for the example. You should have a directory ‘dist’ in your current directory.

The application is pushed up using the CLI command ‘cf push’. The full command we will use is ‘cf push atb-af-example -p dist\AFJavaExample.war -b <https://github.com/cloudfoundry/java-buildpack.git>’. See ‘cf help push’ for a detailed explanation of the parameters. We’ll execute the command, but not show the terminal window, because it spits a lot of messages. The command takes a minute or so to run. At the end, you will see a screen that looks like this:



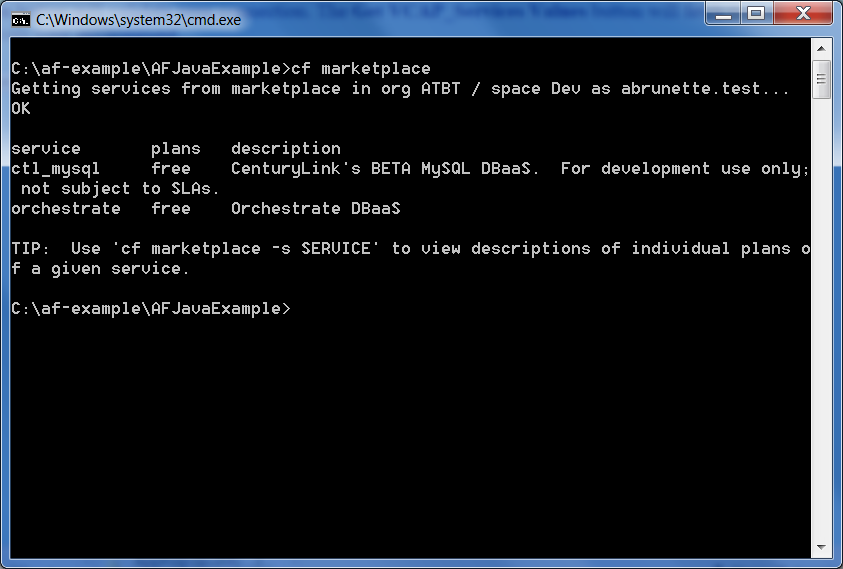
Note the “urls:” item. This is the URL of the application. You can access this now. Try it to verify your success so far. You should see this page:



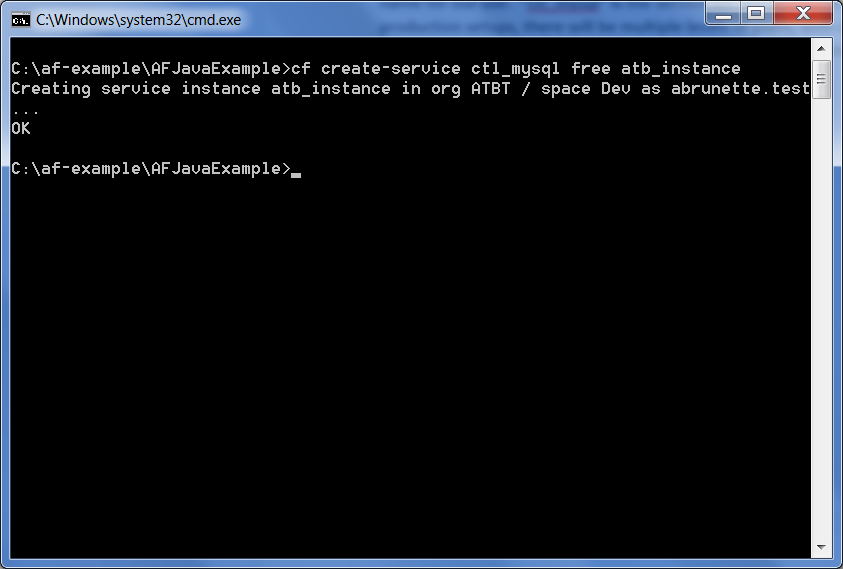
Great work. This app is designed to access your environment variables, and check a db connection. Since we haven’t connected a database yet, these variables don’t exist, and we can’t connect yet. So, let’s set that up…

**Configuring The Database Service:**

The next thing we want is a database to run against. We will get database features through CenturyLink’s MySQL as a service. We find the name of available services through the AppFog marketplace. This is accessed through a call to the Command Line Interface (CLI), “cf marketplace” as shown below:



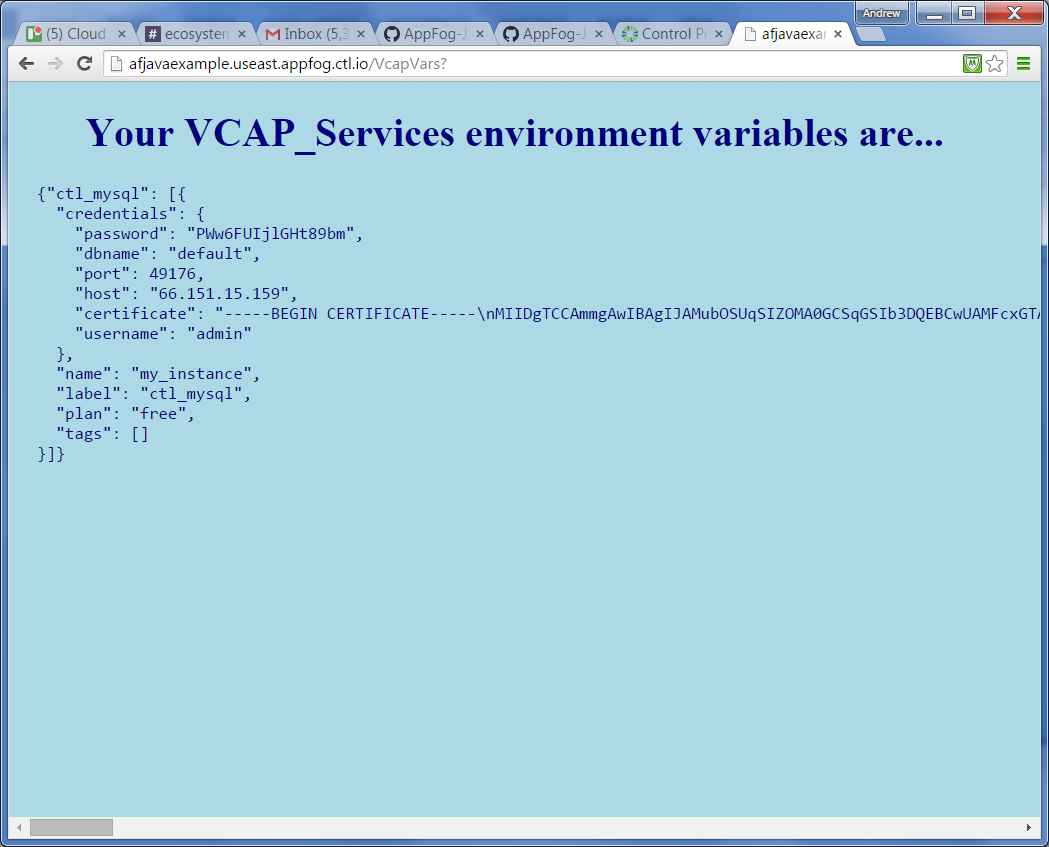
This shows that there are two services available in the marketplace, “ctl\_mysql” and “orchestrate”, a NoSQL database solution. The next step is to create an instance of the ctl\_mysql service, which causes the service to provision a service instance for our use. This is done using the CLI again, using the command, “cf create-service ctl\_mysql free <my\_instance\_name>”, where <my\_instance\_name> is a name for our use. “ctl\_mysql” is the service we are binding to, “free” is the billing plan name. In production setups, there will be multiple levels of plans, with different costs and levels of resources. For now, though, let’s keep it cheap! Create the DB instance as shown below:



For the next step, we need to know the name of our application. Our application name was given in the ‘cf push’ command above, and is ‘atb-af-example’. YOU WILL NEED A UNIQUE NAME. We are going to bind the database service to the application, through the command, ‘cf bind-service atb-af-example atb\_instance’. This command will end with a note that you need to restage your app to be able to access the environment variables that having the database login information. That command is ‘cf restage atb-af-example’. Do that now. This restarts the application, allowing it to pick up the environment variables that contain the DB connection parameters. You should now enter the URL “<my appname>.useast.appfog.ctl.io” into your browser and access our app. Try that. You should see a page that looks like this again.



Try the “VCAP\_SERVICES” Values button. That should give you a page similar to this:



These are the environment variables needed to access the database service. You can use these values to access the database via your usual client, but we’ll access using Java. The code is in CheckConnection.java, and the interesting bit is:

JSONArray serviceData = parmList.getJSONArray("ctl\_mysql");// fetch the credentials for the my\_sql service

JSONObject payload = serviceData.getJSONObject(0);

JSONObject credentials = payload.getJSONObject("credentials");

String host = credentials.getString("host");

Integer port = credentials.getInt("port");

String dbUser = credentials.getString("username");

String dbPassword = credentials.getString("password");

String database = credentials.getString("dbname");

String certificate = credentials.getString("certificate");

try {

addCertToKeystore(host, certificate.getBytes());

Class.forName("com.mysql.jdbc.Driver");

String url = new StringBuilder("jdbc:mysql://").append(host).append(":").append(port)

.append("/").append(database).append("?")

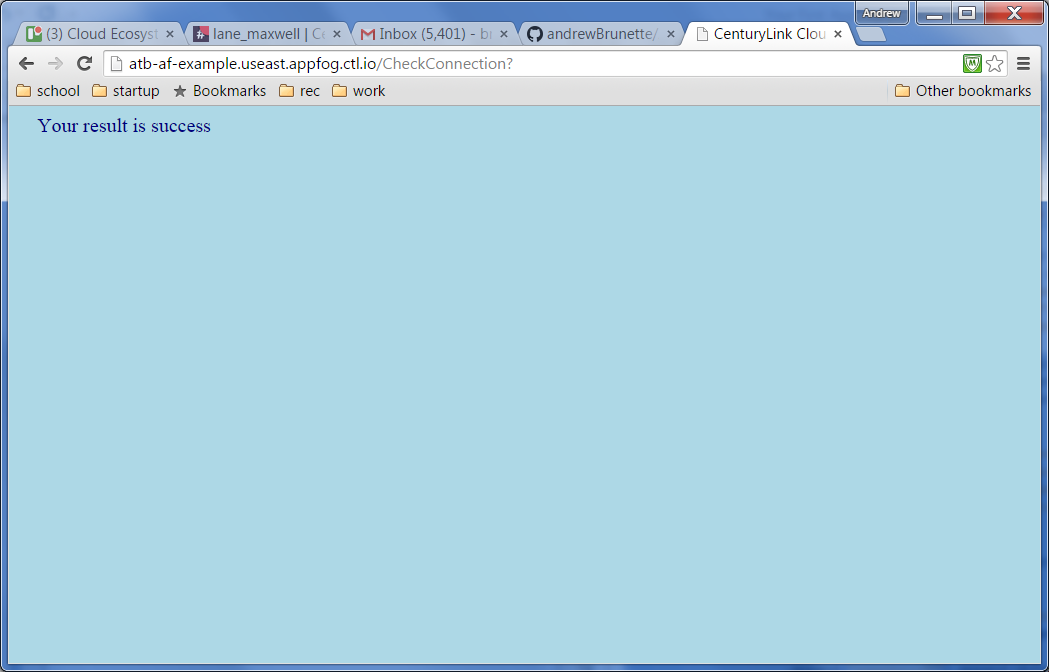
.append("user=").append(dbUser)

.append("&password=").append(dbPassword)

.append("&useSSL=true").toString();

Connection conn = DriverManager.getConnection(url);

If we click the “test connection” button, the result should be as shown below. If there is a problem, the exception message should display.



Congratulations! You have successfully run your first AppFog example.

**Code Review:**

The code to access the environment variables in in src\com\centurylinkcloud\ecosystem\VcapVars.java. The interesting bit is:

String json = new String();  
 JSONObject jObj = null;  
 try {  
 int spacesToIndentEachLevel = 2;  
 json = new JSONObject(System.getenv("VCAP\_SERVICES")).toString(spacesToIndentEachLevel);

}

This simply gets the environment variable “VCAP\_SERVICES”, and formats it as a string for purposes of display. To get the data into a connection call, see CheckConnection.java in the same directory, where the interesting bit is:

private JSONObject getEnvData()throws Exception {

try {

return new JSONObject(System.getenv("VCAP\_SERVICES"));

}

catch (Exception e){

throw new Exception("Your VCAP\_SERVICES environment variable is not set, indicating no service available");

}

}

private String tryConnection(JSONObject parmList) {

System.setProperty("javax.net.ssl.keyStore", getKeyStorePath());

System.setProperty("javax.net.ssl.keyStorePassword", PASS);

System.setProperty("javax.net.ssl.trustStore", getKeyStorePath());

System.setProperty("javax.net.ssl.trustStorePassword", PASS);

JSONArray serviceData = parmList.getJSONArray("ctl\_mysql");// fetch the credentials for the my\_sql service

JSONObject payload = serviceData.getJSONObject(0);

JSONObject credentials = payload.getJSONObject("credentials");

String host = credentials.getString("host");

Integer port = credentials.getInt("port");

String dbUser = credentials.getString("username");

String dbPassword = credentials.getString("password");

String database = credentials.getString("dbname");

String certificate = credentials.getString("certificate");

try {

addCertToKeystore(host, certificate.getBytes());

Class.forName("com.mysql.jdbc.Driver");

String url = new StringBuilder("jdbc:mysql://").append(host).append(":").append(port)

.append("/").append(database).append("?")

.append("user=").append(dbUser)

.append("&password=").append(dbPassword)

.append("&useSSL=true").toString();

Connection conn = DriverManager.getConnection(url);

conn.close();

return new String ("success");

} catch (Exception e) {

System.out.println(e.getMessage());

return e.getMessage();

}

}

protected void processRequest(HttpServletRequest request, HttpServletResponse response)

throws ServletException, IOException {

response.setContentType("text/html;charset=UTF-8");

JSONObject credentials = null;

try {

credentials = getEnvData();

}

….

String result = tryConnection(credentials);

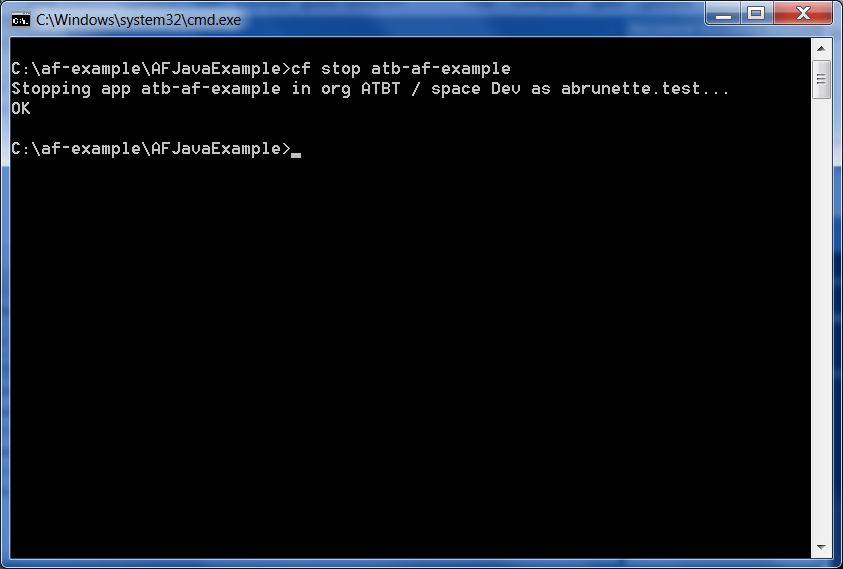
The processRequest() method calls getEnvData() method to get the environment, and then executes the tryConnection() method to use them.

Using SSL:

Since you are accessing your data over the open internet, it is important to use an encrypted connection. The code included here includes the necessary steps to use the certificate provided, and should need no modification. The “useSSL=true” parameter in the connection setup tells MySQL to use SSL. The four methods at the bottom of the file, addCertToKeystore(), getKeystore(), getKeyStorePath(), and persistKeyStore() are necessary to handle the certificate so that the MySQL connector can access it. You will need to either use these, or manage your own keystore in some other manner. You are free to use this code without attribution.

**Housekeeping:**

AppFog and the database connections are services. You accumulate charges if they stay running. To avoid this, shut them down when done. You shut down your AppFog instance using the CLI command, ‘cf stop <APPNAME>’ or, in this example, as shown below:



You remove the database service by first unbinding it from the app, then deleting it. Unbind it using the CLI command ‘cf unbind <appname> <servicename>’. Then delete it using ‘cf delete-service <servicename>’.

Finally, delete your instance using the ‘cf delete <appname>’ command

See below for the complete sequence. This will cause your account to not be further charged for any activity.

