****

|  |
| --- |
| Developing Applications  Mobile Cloud with Oracle JET and Developer Cloud  ERIC MITHA  JTI WORKSHOP |

Table of Contents

[1 Introduction 3](#_Toc480213691)

[1.1 Summary 3](#_Toc480213692)

[2 Set up 3](#_Toc480213693)

[2.1 Set up Developer Cloud Service 3](#_Toc480213694)

[2.2 Connect Eclipse IDE 3](#_Toc480213695)

[2.3 Install Node.js 5](#_Toc480213696)

[2.4 Install and set up Oracle JET 5](#_Toc480213697)

[2.5 Connect the application to MCS 6](#_Toc480213698)

# Introduction

## Summary

At the end of this workshop we will have a hybrid mobile application that can run in a browser as well as on Android devices. The application will be built using Oracle JET (Javascript Extension Toolkit).

The Oracle JET application will be managed using Developer Cloud Service. In this demo we will explore how to create a Git repository to host the code of the application. This workshop will use the Eclipse IDE when editing the Oracle JET and Node.js code and the Eclipse OEPE Cloud plugin for connecting to the cloud (and its Git repositories). However, a CLI Git client or any other IDE supporting Git can be used.

# Set up

## Set up Developer Cloud Service

Before we create our application, let’s create a Git repository in Developer Cloud Service where we will host our code.

In the home (Project) page of your Developer Cloud Service project, press the “New Repository” button. Enter a name for your Git repository (i.e jetjtirepo) and select “Initialize repository with README file”. Press “Create”. This will create a new initialized Developer Cloud Service Git repository, only containing a README file, where you will later on host your code.

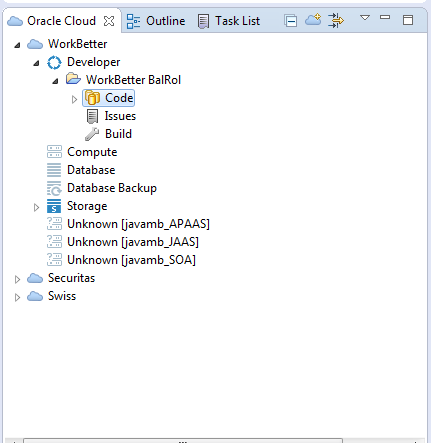
## Connect Eclipse IDE

Prior to this demo you should already have installed the Eclipse IDE with the OEPE (Oracle Enterprise Pack for Eclipse). If you want to use another IDE with Developer Cloud Service, make sure it is Git compatible (both NetBeans and JDeveloper have Developer Cloud Service plugins). **You don’t need an IDE. Just use a Git CLI and a text editor.** If you still want to use Eclipse but did not install it yet, install Eclipse with Oracle Enterprise Pack from <http://www.oracle.com/technetwork/developer-tools/eclipse/downloads/index.html>

Select the Mars version for Windows 64-bit.

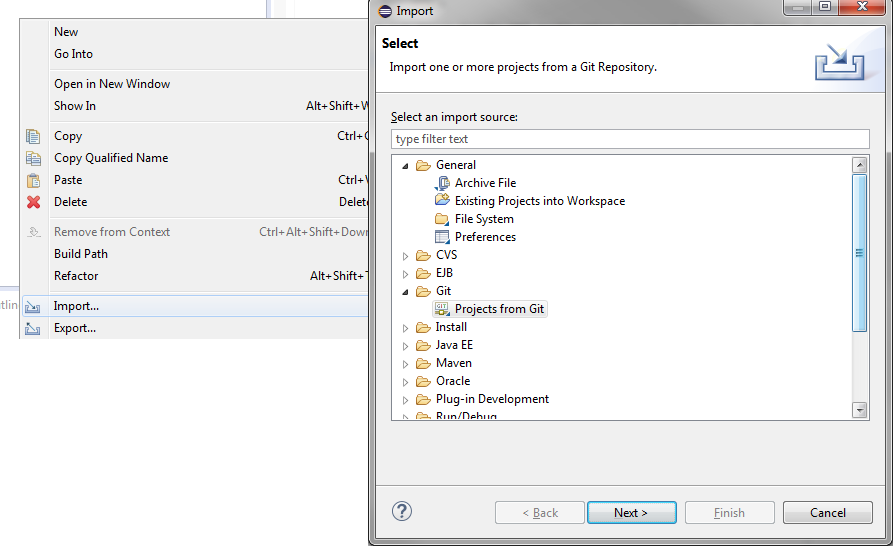
Open Eclipse and select a workspace. Go to File -> New -> Other -> Oracle -> Cloud -> Oracle Cloud Connection. Enter the credentials used when signing into the cloud services and use your preferred name of the connection.

In the Oracle Cloud window we can now navigate to the Developer Cloud Service instance -> Developer Cloud Service project -> Code and hit Activate on the Git repository we created earlier.



**Import application code**

Now, in the Package or Project Explorer, right click and go Import -> Git -> Projects from Git -> Existing local repository -> select the Git repository you just activated -> Finish.



In the Package Explorer we can now see our newly created project and the README file that exists in the Git repository. Before we create our application inside the Git repository you can go ahead and delete the README file.

## Install Node.js

**Download and install Node.js**

First we need to install Node.js and Node.js Package Manager which are needed in order to create the server running our Oracle JET application.

This demo follows the installation of a 64-bit msi Windows installer.

Download Node.js installer (which contains Node.js Package Manager – NPM) from <https://nodejs.org/en/download/>

Install Node.js in your preferred directory.

## Install and set up Oracle JET

**Set up and install Node and Oracle JET options and tools**

Open a Windows command line interface in the **folder of the Git repository you just cloned to your local machine (i.e C:\Users\Lisa\EclipseWorkspace\name-of-repo.git-bd18)**. If you are behind a corporate firewall, use below command to configure the Node.js Package Manager (npm) to use a proxy.

**npm config set proxy YOUR\_PROXY**

e.g @oracle – if VPN activated

npm config set proxy <http://www-proxy.us.oracle.com:80>

npm config set https-proxy <http://www-proxy.us.oracle.com:80>

First, we want to install a few Node packages, namely **yo**, **grunt-cli**, **bower** and **generator-oraclejet** as well as **cordova**.

**npm install -g yo grunt-cli bower generator-oraclejet**

**npm install -g cordova**

**Generate a hybrid Oracle JET mobile application**

By using yo (Yeoman) we now want to create a hybrid mobile application that can run on both Android devices as well as in a web browser. If we wanted to, we could add iOS and Windows options as well but in this workshop we will stick to Android only.

**yo oraclejet:hybrid JETJTIApp --template=navdrawer --platform=android**

Of course, we cannot build the app to Android without the SDK installed on our computers. However, we can simulate the look and feel of the Android platform by launching the app in our browser.

Next, let’s try out the app in our browser. Again, if we had an Android SDK installed, we could build the app to our device by specifying device as destination instead of browser.

**cd JETJTIApp**

**grunt serve --platform=android --destination=browser**

Open a browser and enter <http://localhost:8000/browser/www/index.html> to have a look at the running application.

## Connect the application to MCS

Before we connect our application to MCS we need to setup a Client definition for our Mobile Backend. Go to your Mobile Backend and press Clients -> New Client. Since we will actually not deploy this application to a real device (since it requires installation of Android SDK), enter some properties for a Web Platform client. When the Client definition is created, copy the Application Key to your notes as you will need it in a couple of minutes.

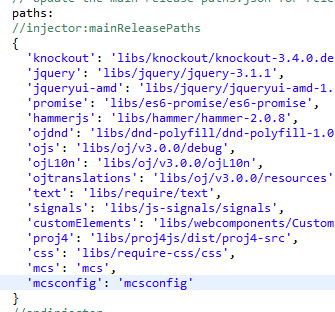
As the last step of the lab, let’s see how easy it is to connect our hybrid Oracle JET mobile application to our Mobile Cloud. There are a number of ways to do that and a lot of SDKs to choose from such as iOS SDK, Android SDK, Windows SDK, Cordova SDK and Javascript SDK. In this lab we will use the Javascript SDK.

**Add the SDK to the project**

Firstly, we want to add the Javascript SDK to our project.

Copy the mcs.js file and mcsconfig.js file from labfiles/AppDevLab/mcsSDK folder and paste it into JETJTIApp/src/js folder. In the mcsconfig.js file, change YOUR\_BACKEND\_NAME, YOUR\_BACKEND\_BASE\_URL, YOUR\_BACKEND\_APPLICATION\_KEY, YOUR\_BACKEND\_ID and YOUR\_BACKEND\_ANONYMOUS\_TOKEN according to the values in your Mobile Backend’s Settings page (as you did for the Postman step) and the Client Application Key you saved in the previous step.

In order for our application to be able to use the mcs.js SDK, we need to declare it in JETJTIApp/src/js/main.js file. We also need to declare our mcsconfig.js file in the same way. The mcsconfig.js file contains keys and ids relevant to your specific MCS instance which we will configure in the next step. Edit the main.js file that so it looks like below.



**Add a list of Incidents**

We now want to add a list to our Incidents page of our app so that we can see all our current incidents. Of course, this information will be retrieved from the MCS API we created earlier. To achieve this we will be using an Array List View component from the Oracle JET Cookbook. The JET Cookbook is a huge library of free graphical components that can be reused. The one we will implement can be found here: <http://www.oracle.com/webfolder/technetwork/jet/jetCookbook.html?component=listView&demo=arrayListView>

Instead of listing different Oracle products, we want to list our Incidents coming from our MCS API.

Under JETJTIApp/src/js/viewModels and JETJTIApp/src/js/views, replace the incidents.html and incidents.js files from labfiles/AppDevLab/incidentCodes folder. The added code simply accesses the MCS SDK to connect to the values you provided in the mcsconfig.js file and then makes a call to the Incidents API. The result from the call is then displayed in the list component.

Change “YOUR API NAME” in the incidents.js file to the name of your API.

**Let’s see if it works!**

Now, before we push the code up to Developer Cloud Service, let’s see if it runs locally. Still in the JETJTIApp folder of your Git repository, once again execute the below command to run the application.

**grunt serve --platform=android --destination=browser**

When the command has completed, you should now try to access <http://localhost:8000/browser/www/index.html> once again. Navigate to the Incidents page. You should now see a list of all the incidents in the backend system, retrieved from your MCS API.

Finally, head back to Eclipse (or whatever IDE you are using) and refresh the project. Add all the latest files to the staging area and Commit and Push the code up to Developer Cloud Service.