****

|  |
| --- |
| Workshop  Application Container Cloud Service with Oracle JET and Developer Cloud Service  Linus Hakansson  Sales Consulting Centres |

Table of Contents

[1 Introduction 3](#_Toc464569501)

[1.1 Summary 3](#_Toc464569502)

[1.2 Assets 3](#_Toc464569503)

[2 Set up 4](#_Toc464569504)

[2.1 Set up Developer Cloud Service 4](#_Toc464569505)

[2.2 Connect Eclipse IDE 4](#_Toc464569506)

[2.3 Install Node.js 5](#_Toc464569507)

[2.4 Create Node.js skeleton application 5](#_Toc464569508)

[2.5 Include a sample Oracle JET application 6](#_Toc464569509)

[2.6 Prepare the app for deployment to Application Container Cloud Service 7](#_Toc464569510)

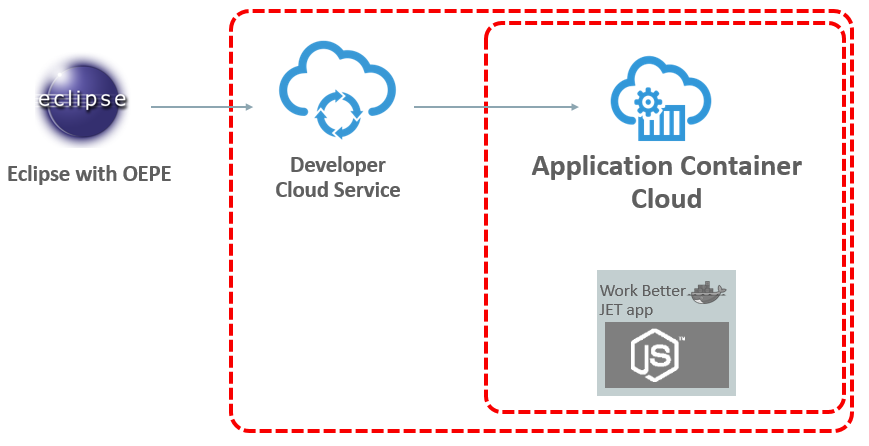
[2.7 Using Continuous Integration and Continuous Delivery with Developer Cloud Service 8](#_Toc464569511)

[2.8 Agile development using Developer Cloud Service 10](#_Toc464569512)

# Introduction

## Summary

At the end of this workshop we will have an Oracle JET (Javascript Extension Toolkit) application deployed in the cloud. The application will run in a Node.js server on Application Container Cloud Service. Except for the Oracle JET application, the Node.js server will also provide a public REST API that can be consumed by any REST client as well as the Oracle JET UI itself.



The Oracle JET application that will be running on Node.js in Application Container Cloud will be managed and deployed from Developer Cloud Service. In this demo we will explore how to create a Git repository to host the code of the application as well as setting up continuous integration and continuous deployment. We will also work in pairs (two and two) to collaborate in an agile manner in the development 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.

## Assets

Even though this script includes all commands and steps needed to configure the demo from scratch, it also comes with all the resources needed to deploy the application quickly.

This script comes with a **workbetter.zip** folder containing the whole project including Node.js and Express resources as well as the Oracle JET Work Better application. It also includes the manifest.json file needed to upload the application to Application Container Cloud Service. The **workbetter.zip** can be used when deploying the application directly to ACCS or extracted to be configured with a Git repository.

# Set up

## Set up Developer Cloud Service

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

Sign in to the account you have been assigned on [https://cloud.oracle.com/en\_US/sign-in](https://cloud.oracle.com/en_US/sign-in%20). From the Dashboard open the Service Console of Developer Cloud Service. Alternatively, enter the Developer Cloud Service URL that you have been assigned. Your user should already have a project with the name that is shown in the workshop document for the user accounts.

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 lisa-workbetter) 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.

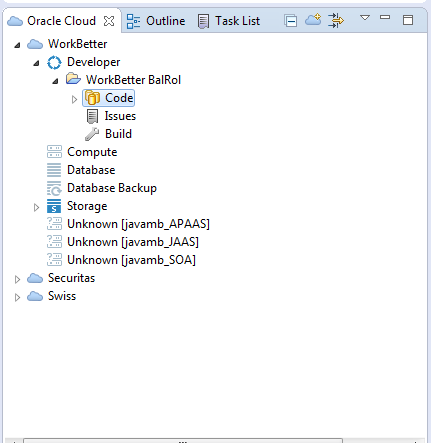
## Connect Eclipse IDE

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). 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.

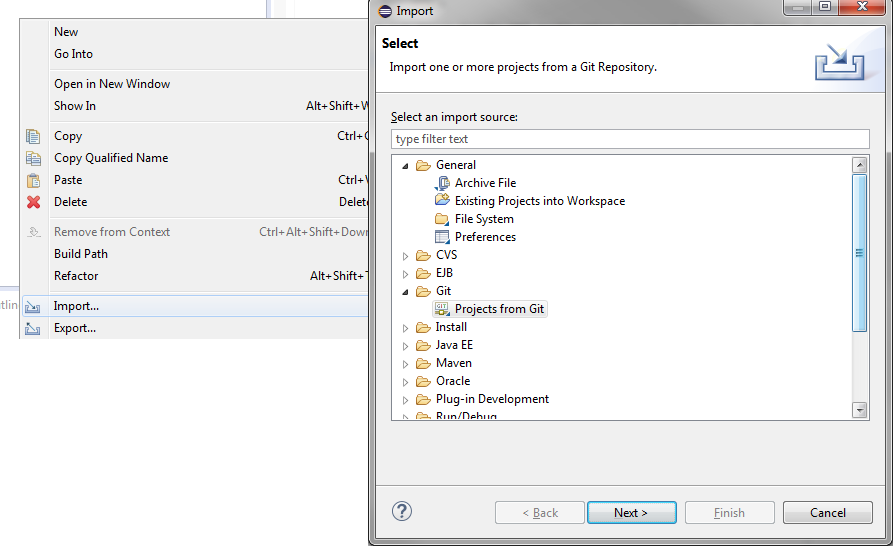
Open Eclipse and select a workspace. Go to File -> New -> Other -> Oracle -> Cloud -> Oracle Cloud Connection. Enter the credentials used when signing into the cloud services and use your preferred name of the connection.

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.

## Create Node.js skeleton application

**Set up and install Node 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\lisa-workbetter.git-bd18). If you are behind a corporate firewall, use below two commands to configure the Node.js Package Manager (npm) to use a proxy.

**npm config set proxy http://dmz-proxy-adcq7.us.oracle.com:80**

**npm config set https-proxy http://dmz-proxy-adcq7.us.oracle.com:80**

Now we want to use a simple Node.js application framework called Express as well as the express-generator plugin that lets us create a skeleton application.

**npm install -g express**

**npm install express-generator -g**

**Generate a skeleton Node.js application**

By using the express-generator we will now create a skeleton application that we will call workbetter. Remain in the same CLI in the same folder (i.e C:\Users\Lisa\EclipseWorkspace\lisa-workbetter.git-bd18) and type

**express workbetter**

What we want to do now is to navigate to the directory we just created as part of the express application generation and to install all Node.js dependencies in this folder.

**cd workbetter**

**npm install**

## Include a sample Oracle JET application

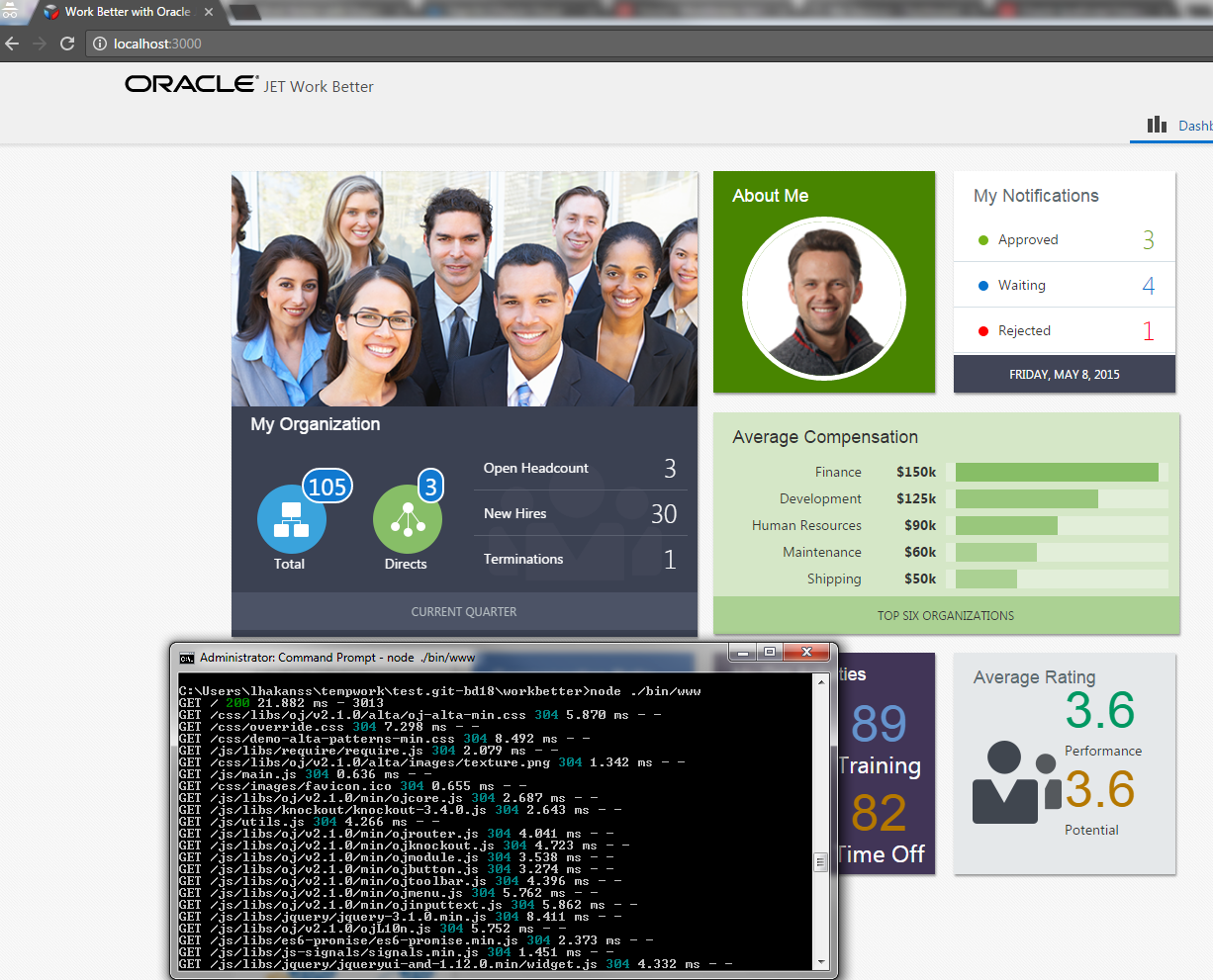
We now have a skeleton Node.js application which we can actually run on our local machine. However, before we try that, let’s add a sample Oracle JET application inside of our skeleton in order to have something cooler than an empty skeleton app. Download the Oracle JET sample app Work Better from <http://www.oracle.com/technetwork/developer-tools/jet/downloads/index.html>. First accept the Licence Agreement and then press “[WorkBetter: Sample Application showing full single-page functionality for Oracle JavaScript Extension Toolkit](http://www.oracle.com/technetwork/developer-tools/jet/downloads/index.html)” to download the zip-file.

You should now have a folder called “public” in C:\Users\Lisa\EclipseWorkspace\lisa-workbetter.git-bd18/workbetter/public. Delete all content in this folder and then extract the content of the WorkBetter.zip inside this folder. Refresh your Eclipse projects to verify that the files have been extracted.

You should now have a Node.js server that serves a static Oracle JET application using the Express framework. To verify that the application can run on the Node.js server, open a Windows CLI window in C:\Users\Lisa\EclipseWorkspace\lisa-workbetter.git-bd18/workbetter and enter the below command.

**node ./bin/www**

Now open your favourite browser and type “localhost:3000”.



## Prepare the app for deployment to Application Container Cloud Service

**Create manifest.json file**

When an application is deployed to Application Container Cloud Service, the service will require a “manifest.json” file to exist in the root directory. The file needs to contain both the “majorVersion” field as well as the “command” field. The “majorVersion” field is used for Application Container Cloud Service (ACCS) to determine what version of Node.js or Java SE should be used when creating the application. The “command” field is used by ACCS to actually start and run the application. In this example we are using version 0.12 of Node.js and a node command: “node ./bin/www” to start and run our application. This is the same command that you used when you ran the server locally.

In the root folder of the workbetter project (C:\Users\Lisa\EclipseWorkspace\lisa-workbetter.git-bd18/workbetter), create a file called manifest.json and put the following content inside it:

**{**

**"runtime":{**

**"majorVersion":"0.12"**

**},**

**"command": "node ./bin/www",**

**"release": {},**

**"notes": "Sample, Quick Start Oracle JET Application prepared to run on Oracle Application Container Cloud"**

**}**

If you want, you can zip all files in the workbetter folder and upload that zip file to ACCS for deployment. However in this workshop, we will leave that step to the continuous deployment phase of Developer Cloud Service.

## Using Continuous Integration and Continuous Delivery with Developer Cloud Service

We now want to achieve the following: Whenever we change any code in the application, we want to automatically archive the content in a zip file and deploy a new version of the application to ACCS.

**Create a Hudson job in Developer Cloud Service**

Go back to Developer Cloud Service and create a Hudson job in Developer Cloud Service. Go to Build -> New Job -> Enter a name (i.e lisa-workbetter-build) and select Create a free-style job -> Save.

In the Source Control tab select Git and in the URL menu select your Git repository. This connects this Hudson job to the Git repository. Next go to the Triggers tab where we will configure when this job should actually run. Select Based on SCM polling schedule to have the job automatically trigger whenever any changes are pushed to the Git repository we just configured under Source Control. Finally, go to the Build Steps and select Add Build Step -> Execute shell. In the command field, input the following:

**cd workbetter**

**zip -r workbetterapp.zip .**

The above command will first navigate inside the root folder of the workbetter project and then zip all files to a zip file called “workbetterapp.zip”.

Finally, we will tell Developer Cloud Service to archive this zip file so that we can deploy it to ACCS later on. Go the Post Build tab and check   
Archive the artifacts

In the “Files To Archive” field, enter “workbetter/workbetterapp.zip”.

Press “Save”.

The last thing that is needed to deploy the archived workbetterapp.zip to ACCS is to create a Deployment Configuration in Developer Cloud Service. Before we can do that, Developer Cloud Service first needs to have been able to create that zip-file once. To achieve that, let’s push our initial code to the Git repository.

**Commit and push to remote Git repository**

Now we want to commit and push our application to the remote Git repository on Developer Cloud Service. To do this, open Eclipse and right click on any file in the project and go Team -> Commit. Depending on the version of your Eclipse you might need to select all the files under “Unstaged Changes” and right click to select “Add to Index”. In some versions of Eclipse, just make sure all files are selected for the commit. Enter any message in the Commit message field and hit Commit and Push.

This step will take roughly 5 minutes (a lot of files to push the first time) – go get a coffee! Alternatively, explore the locally running JET application or the Oracle JET cookbook: <http://www.oracle.com/webfolder/technetwork/jet/jetCookbook.html>

☺

**Creating the Deployment Configuration**

Finally, go into the Project page of Developer Cloud Service. Depending on the length of your break, there will either be a news component that tells you that “Build 1 of job lisa-workbetter-build ended:” or that “Build 1 of job lisa-workbetter-build is running”. If it is the latter you can have a look at the live console output of the build in the Build tab until it completes.

Now, go to the Deploy tab of Developer Cloud Service and press “New Configuration”. For the “Configuration Name” and “Application Name” enter WorkBetter-Lisa or whatever you want to call it. For “Deployment Target”, press “New” -> “Application Container Cloud…”. Select us2 as the “Data Center” and enter your provided credentials. Press “Test Connection” and “Save”. Back in the Deployment Configuration screen, chose men j“Runtime” as “Node” and select “Automatic” as the “Type”, and then your Job, Build and Artifact. Press “Save”. You now have your Deployment Configuration, to deploy the application directly without waiting for the next build to trigger – click on the cogwheel for your configuration and hit “Start”.

**Accessing the application**

After a while, the zip file will be deployed to ACCS. To confirm, go back to the Cloud Dashboard of your account and navigate to the Service Console of Application Container Cloud Service. You should now see a Node.js application called WorkBetter-yourName. Press the URL of the application to access it.

## Agile development using Developer Cloud Service

We will now work together in pairs to create two different features for our application. Of course it is also possible to create both features yourself if you are doing the workshop by yourself. In this chapter we will see how the deployed application is automatically updated when new code enters the master branch of the Git repository. We will also explore some of the agile features in Developer Cloud Service (Merge Requests, Issues and Scrum Sprints) as well as playing around with Node.js and Oracle JET.

The Oracle JET Work Better application is not really doing anything exiting – i.e the data shown in the UI is coming from hardcoded values in JSON files. It would be more exiting if some of the presented values actually came from a REST service – maybe even more exiting – a REST API hosted on a Node.js server hosted in a different cloud container than the one providing the UI.