



InterConnect 2016

The Premier Cloud & Mobile Conference

Session: CCD/HAM-2434

Migrating and Integrating Cloud Applications with On-Premise Resources

Lab Instructions

Authors:

Bobby Woolf, Bluemix Technical Enablement Specialist, IBM

bwoolf@us.ibm.com

Sandhya Kapoor, Cloud Architect, IBM

kapoor@us.ibm.com

February 21 – 25
MGM Grand & Mandalay Bay
Las Vegas, Nevada

February 2016 edition

NOTICES

This information was developed for products and services offered in the USA.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive, MD-NC119
Armonk, NY 10504-1785
United States of America

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM websites are provided for convenience only and do not in any manner serve as an endorsement of those websites. The materials at those websites are not part of the materials for this IBM product and use of those websites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

TRADEMARKS

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

IT Infrastructure Library is a Registered Trade Mark of AXELOS Limited.

ITIL is a Registered Trade Mark of AXELOS Limited.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

© Copyright International Business Machines Corporation 2015.

This document may not be reproduced in whole or in part without the prior written permission of IBM.
US Government Users Restricted Rights - Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Resource guide

IBM Technical Client Training has enhanced its training capabilities, and extended reach into new cities and countries, by partnering with five highly qualified IBM Business Partners who provide high quality, authorized training for IBM Clients, IBM Business Partners, and IBM employees. IBM continues to provide authorized training curriculum and content, and also maintains overall ownership of the IBM Training ecosystem.

The delivery of public, private and customized training to IBM Clients and IBM Business Partners is now done by the IBM Global Training Providers (GTPs):

Arrow

Avnet

Global Knowledge

Ingram Micro

LearnQuest

See ibm.com/training for information on the classes that the GTPs offer.

Completing this InterConnect lab is a great first step in building your IBM skills. IBM offers several resources to keep your skills on the cutting edge. Resources available to you range from product documentation to support websites and social media websites, including the following examples:

IBM Training website

- Bookmark the IBM Training website for easy access to the full listing of IBM training curricula. The website also features training paths to help you select your next course and available certifications.
- For more information, see <http://www.ibm.com/training>

IBM Certification

- Demonstrate your mastery of IBM products to your employer or clients through IBM Professional Certification. Certifications are available for developers, administrators, and business analysts.
- For more information, see <http://www.ibm.com/certify>



InterConnect 2016

The Premier Cloud & Mobile Conference

February 21 – 25
MGM Grand & Mandalay Bay
Las Vegas, Nevada

Table of contents

Resource guide	4
About these exercises	7
Prerequisites	7
Exercise 1: Migrating On-Premise Applications to the Cloud.....	9
Introduction	9
Step 1: Set up environment.....	11
Step 2: Run Application on JBoss	13
Step 3: Migrate application to Liberty	16
Step 4: Deploy to Bluemix	35
Exercise 2: Integrating Cloud Applications with On-Premise Resources.....	38
Introduction	38
Step 1: Set up environment.....	40
Step 2: Connect directly to database.....	49
Step 3: Connect via Secure Gateway.....	52
Step 4: Deploy to Bluemix	59
Appendix A Useful knowledge	63
Bluemix development environment	63
Using the Secure Gateway client CLI	66

About these exercises

The purpose of this Lab is to show you:

1. How to migrate existing Java EE on-premise applications to the Cloud. We will go step by step over the process and tools for migrating Java EE applications.
2. How to connect applications running in IBM Cloud to on-premise resources such as a database of record. Applications running in the cloud often need access to enterprise resources, but those are often deployed privately in on-premise data centers. We will cover the steps to integrate a cloud-hosted application into your enterprise.

Prerequisites

These are the prerequisites for performing these lab exercises.

Bluemix account

If you already have a Bluemix account, you can use that to perform this lab if it meets the following requirements:

- 1 GB memory for Cloud Foundry apps
- 4 GB memory for Virtual Machines
- 2 Services and APIs allowed

If you do not already have a Bluemix account, you can register for a free trial account by performing the following steps.

1. Browse to **<http://bluemix.net>** by using an Internet browser and select **SIGN UP**, as shown in Figure 1.

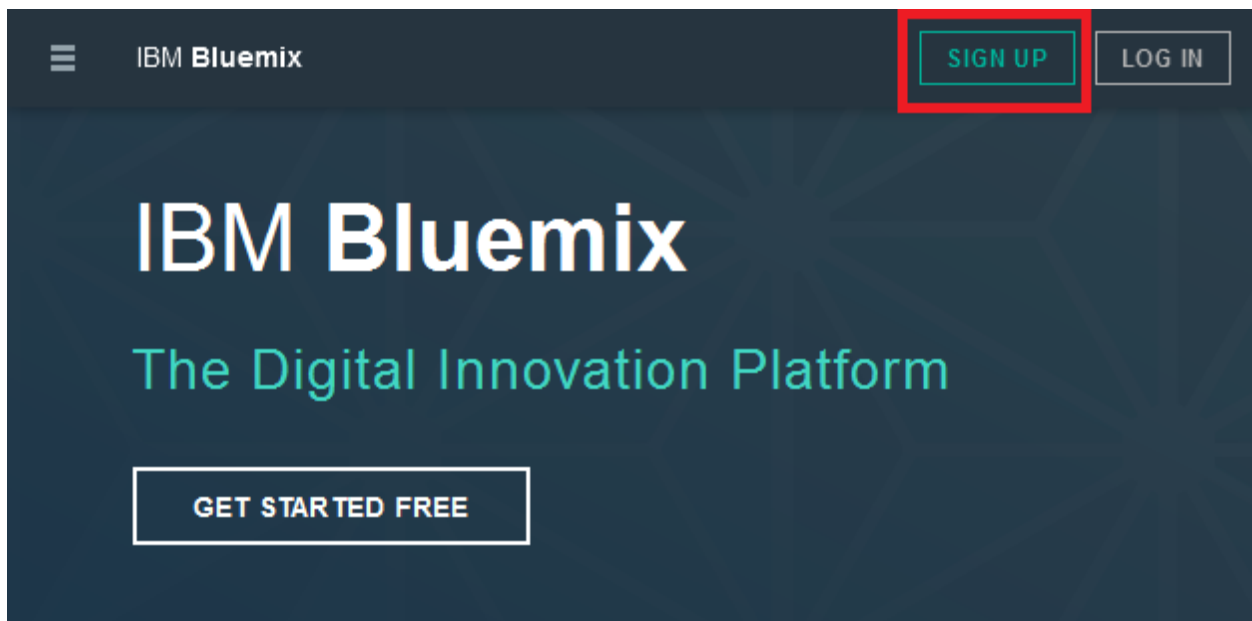


Figure 1: Bluemix homepage

2. At the **Sign up for IBM Bluemix** page, enter the required information and select **CREATE ACCOUNT**.

Conference: Since the sign-up process requires you to validate your email address, please use an email account that you can access right now.

3. Validate your email address when you receive an email from bluemix.net.

Locally installed software

The conference computer is running a virtual machine that includes the development tools you will need to perform this lab. For the list of tools and where to download them from, see “” in the appendices.

Liberty is installed in the directory **C:\dev\ApplicationServers\wlp-8.5.5.8**, so use that as the value of **<LIBERTY_ROOT>**.

Exercise 1:

Migrating On-Premise Applications to the Cloud

This exercise shows how to migrate a Java Enterprise Edition (JEE) application to Bluemix. The sample application runs on-premise in a JBoss application server. This lab will use the WAS Migration Tools in Eclipse to migrate the application to WebSphere Liberty and then deploy that into a Liberty for Java runtime in Bluemix. These techniques can be used to migrate a Java application from any major application server—Tomcat, WebLogic, JBoss, WebSphere Classic, etc.—to Bluemix.

Introduction

The sample application is representative of existing JEE applications that run on heavyweight containers like JBoss. We are going to migrate this application to a lightweight JEE container like Liberty profile on the IBM Cloud.

The sample application, Document Manager, enables users to view and upload documents. The application has security constraints. It consists of three pages:

1. Login Page
2. Documents Page
3. Login Error Page

The application's page flow is shown in Figure 2.

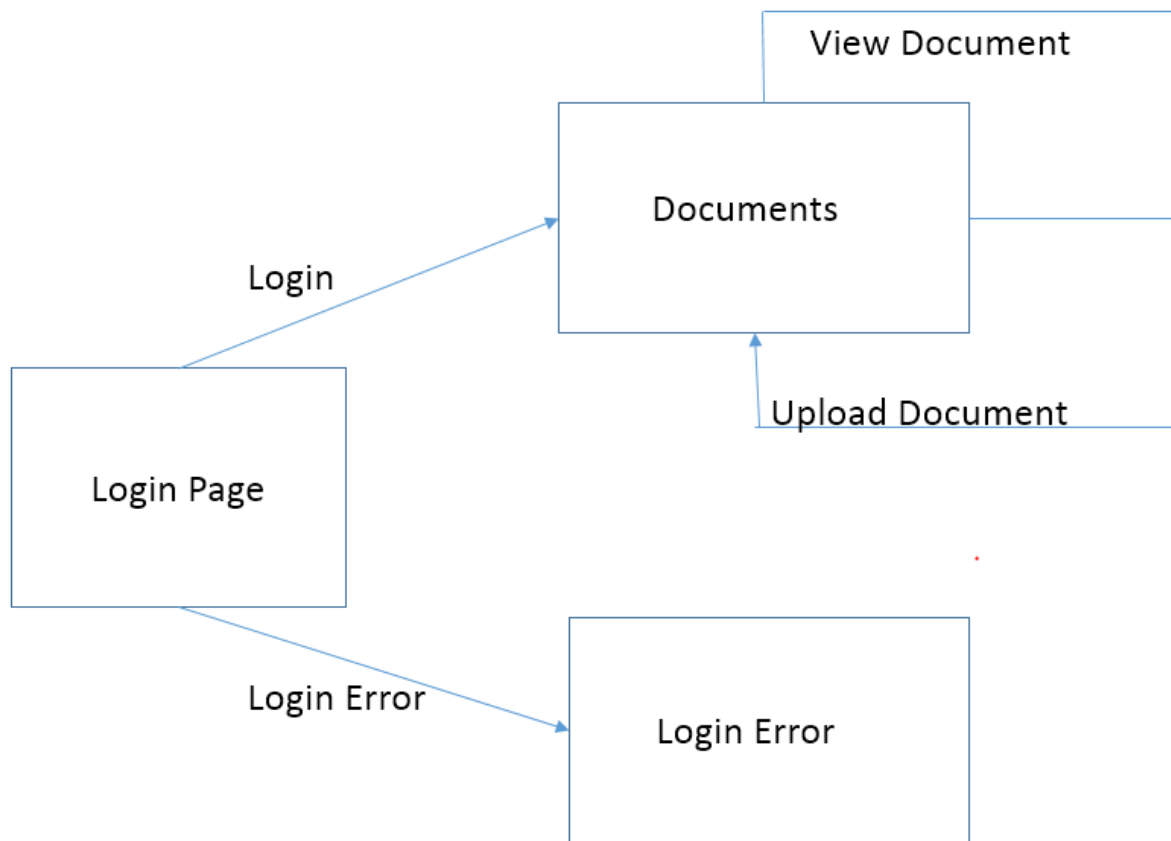


Figure 2: Sample application flow

The application starts with the Login Page. After logging in, the user is redirected to the Documents page. There are two types of users: viewers and editors. Both user types can view documents, but only the editor can upload new documents.

If a user has an editor role, the upload form is displayed below the documents list. When the user selects a document and presses the **Upload** button, the upload method of the DocumentManagerBean session bean is invoked through the execution of the Upload servlet. The Security application will throw an exception if a non-authorized user attempts to call the Upload servlet.

The application has two predefined user IDs:

1. user with password 1
2. editor with password 2

Figure 3 shows the components in the EAR file.

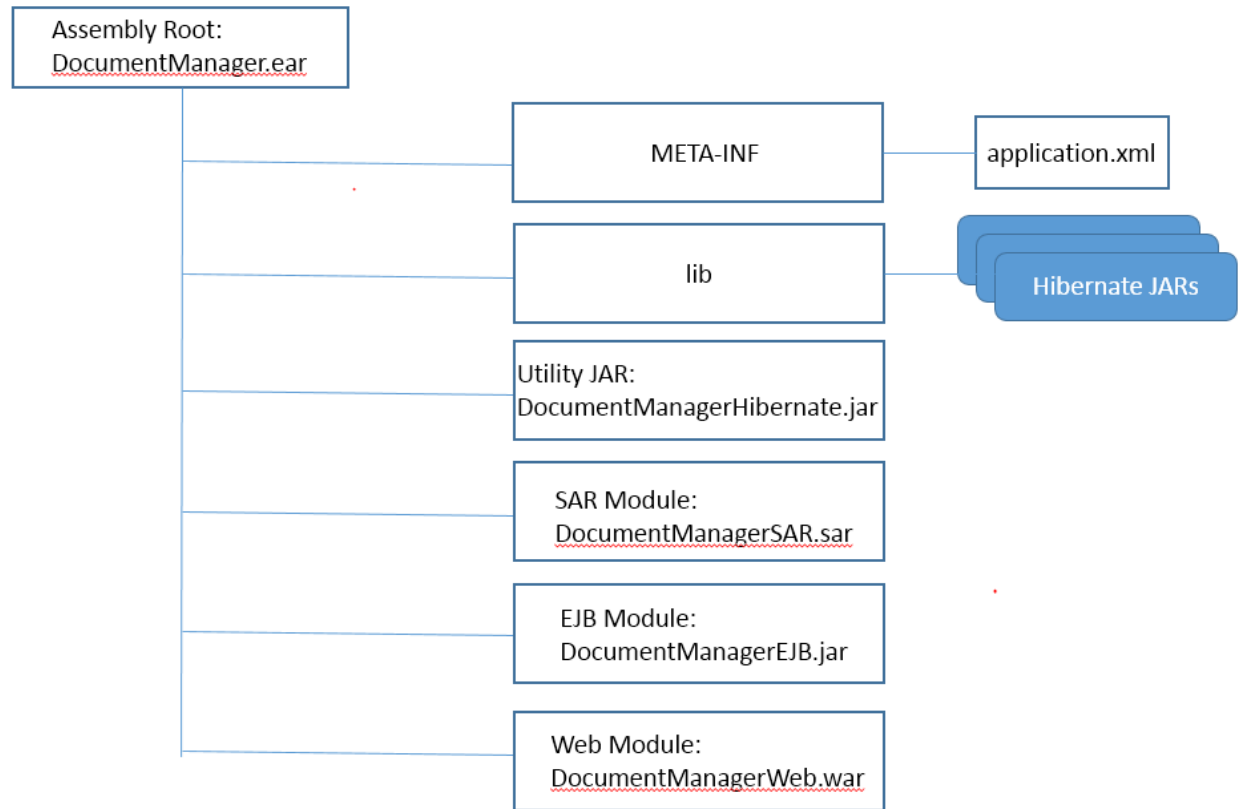


Figure 3: Sample application EAR structure

Let's get started with the tutorial.

Step 1: Set up environment

JDK

JDK 1.8.0_66 is installed in the VM and **JAVA_HOME** is set to **C:\Java\jdk.1.8.0_66**

DOWNLOAD from GitHub

Please go to <https://github.com/ibmecod/MigrateEnterpriseAppFromJBosstoLiberty>. Download the zip or clone the git repository

<https://github.com/ibmecod/MigrateEnterpriseAppFromJBosstoLiberty.git> using Git Bash available to you in the VM.

The cloned folder will be referred as **<CLONED_DIR>**.

MySQL Database

MySQL Server is already installed on the VM. Let's create the tables and populate the data.

1. Click All Programs > MySQL > MySQL Server 5.7>MySQL Command Line Client
A cmd window will open and ask you for a password.
2. The default password on the VM is 'object00'.

Once logged in to mysql, execute the following command to create the tables and load some data:

```
source <CLONED_DIR>\Lab2434_MySQL.sql;
```

Check that all worked successful by running the command

```
select * from files;
```

Setup JBoss Application Server

JBoss 5.1.0 GA is installed under C:\JBoss. **JBOSS_HOME** variable is set to **C:\JBoss\jboss-5.1.0.GA**. This location will be referred to as **<JBOSS_HOME>**.

There is a reported bug with JBoss 5.1 GA server. Therefore, edit

<JBOSS_HOME>/server/default/conf/bootstrap/profile.xml as follows:

The profile.xml has the configuration for AttachmentStore shown in Figure 4.

```
<bean name="AttachmentStore"
      class="org.jboss.system.server.profileservice.repository.AbstractAttachmentStore">
  <constructor>
    <parameter>
      <inject bean="BootstrapProfileFactory" property="attachmentStoreRoot" />
    </parameter>
  </constructor>
```

Figure 4: Original profile.xml for AttachmentStore

Change this to the configuration shown in Figure 5.

```
<bean name="AttachmentStore"
      class="org.jboss.system.server.profileservice.repository.AbstractAttachmentStore">
  <constructor>
    <parameter class="java.io.File">
      <inject bean="BootstrapProfileFactory" property="attachmentStoreRoot" />
    </parameter>
  </constructor>
```

Figure 5: Updated profile.xml for AttachmentStore

Run the command **run.bat -c default** to ensure default configuration for JBoss 5.1 server is working.

After verification, stop the JBoss 5.1 server (Ctrl-c) because soon we will run the JBoss 5.1 server in Eclipse.

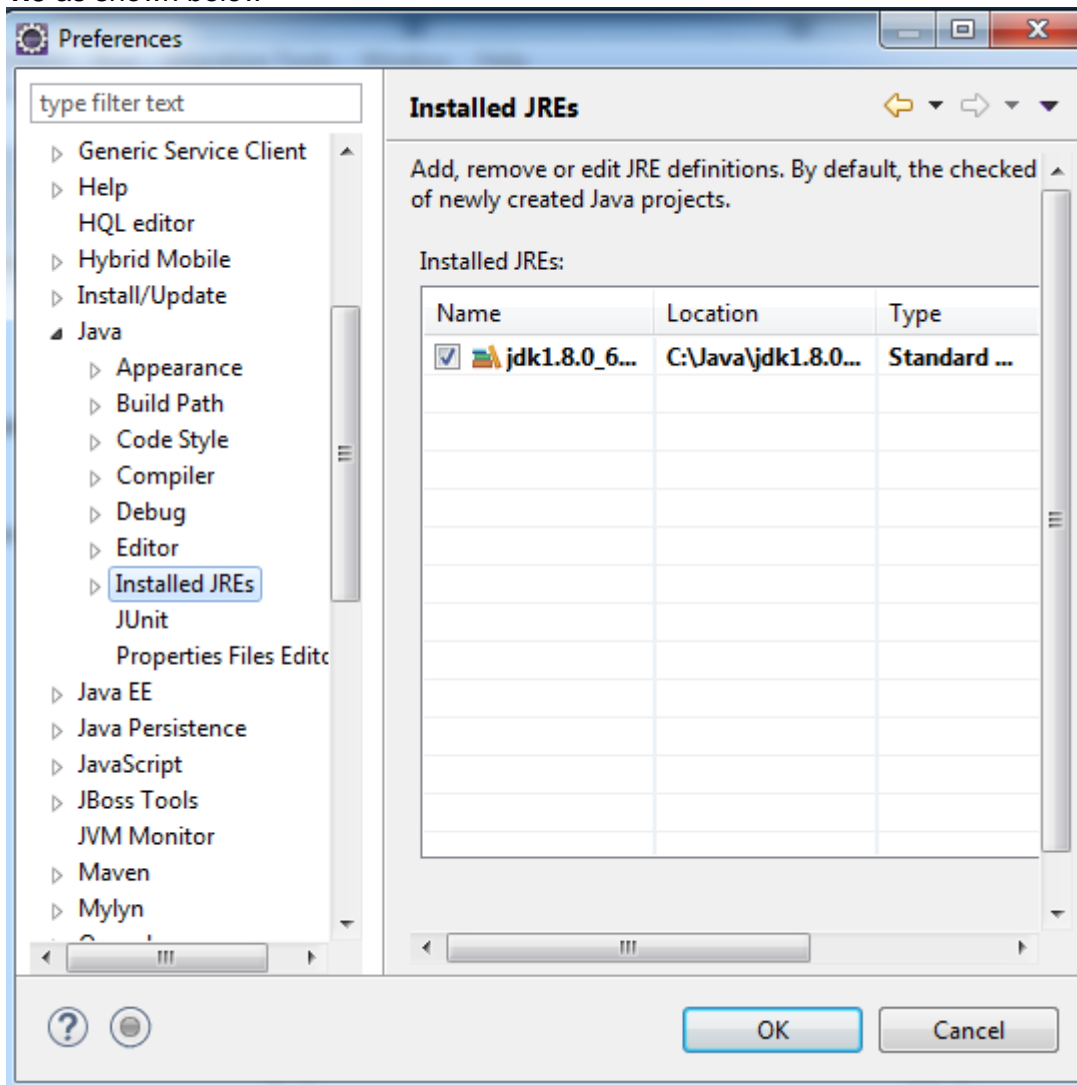
Step 2: Run Application on JBoss

We are using **<JBOSS_HOME>/server/default** configuration.

JBoss will need the mysql jdbc driver. Copy the java jdbc connector, named **mysql-connector-java-5.1.38-bin.jar**, from <CLONED_DIR> into <JBOSS_HOME>/server/default/lib. Also, copy **mysql-ds.xml** file from <CLONED_DIR> into <JBOSS_HOME>/server/default/deploy.

Setup Eclipse Environment:

Set Eclipse JDK to 1.8. Go to **Windows>Preferences>Java>Installed JREs** and point to **JDK 1.8** as shown below



JBoss Tools are installed in Eclipse in the VM. Next, create a JBoss 5.1 Server in Eclipse:

- In Eclipse, go to **Server**
- Right click and select **New->Server**
- Select **JBoss AS 5.1**

- Click **Next->Next**. On the JBoss Runtime dialog, set the home directory to <JBOSS_HOME>. For *Configuration*, select the default folder.
- Click Finish.

Load sample application

In <CLONED_DIR>, you will see **JBossApp.zip**. Import this into Eclipse.

You will see 5 projects as shown in Figure 6 figure below. To remove the errors, please ensure that the projects are bound to JDK 1.8.0_66 and JBoss 5.1 server.

You can set each project Java runtime to workspace runtime as follows

1. Right click on the project.
2. Select Properties.
3. Select Java Build Path.
4. From the *Libraries* tab, double click JRE System Library.
5. Select the Default Workspace JRE.
6. Click Finish.
7. Click Ok.

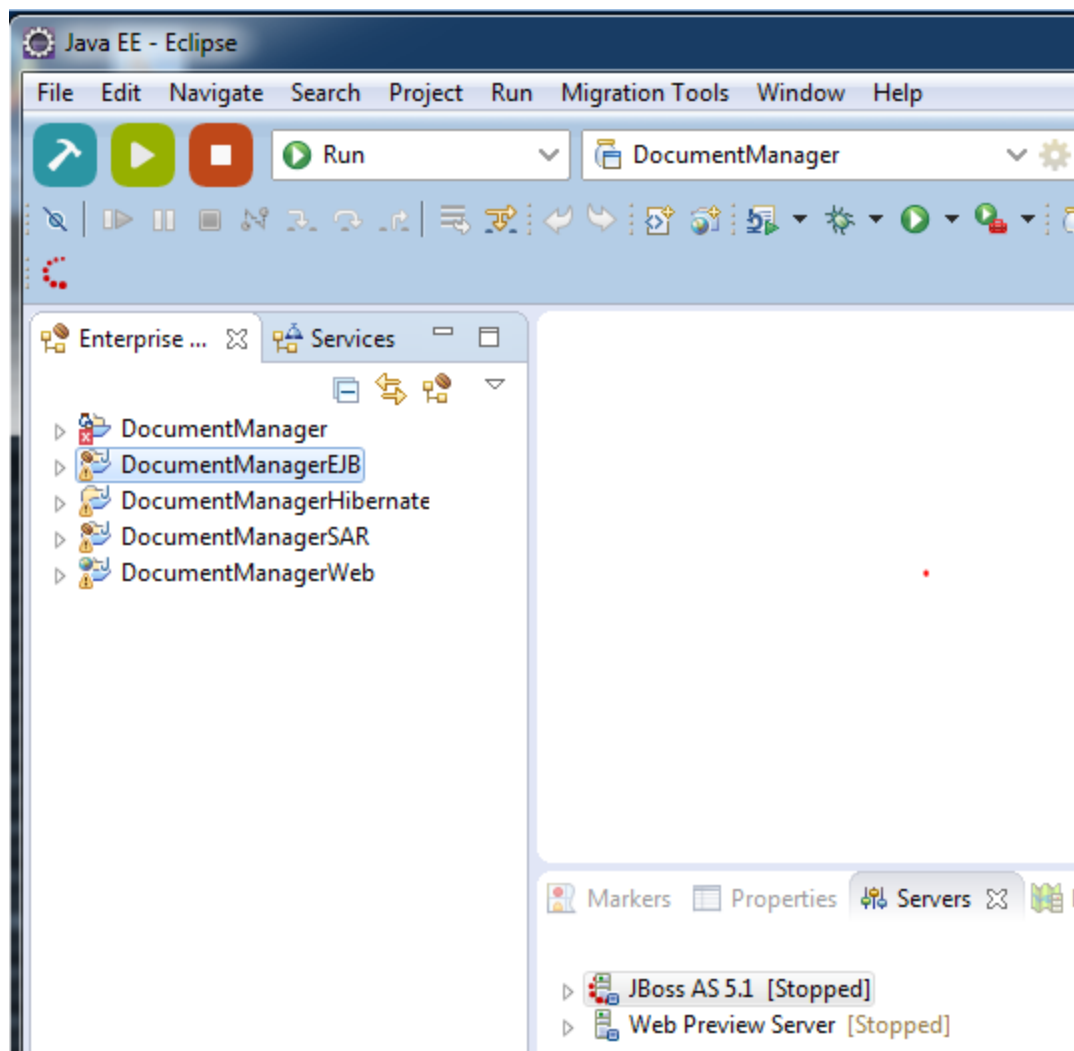


Figure 6: Projects in Project Explorer

Run sample application

Start JBoss 5.1 Server. Next, right click the project **DocumentManager** and select **Run As > Run on Server > JBoss 5.1 AS**. Access the application: **localhost:8080/document/jsp/main.jsp** in a browser, login with username editor and password 2. Browse files to upload them, as shown in Figure 7.

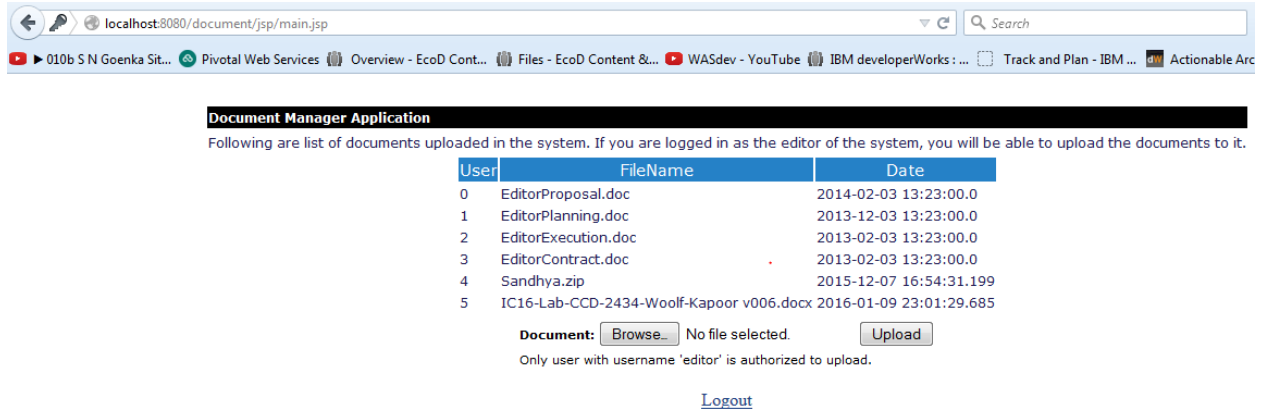


Figure 7: DocumentManager application with username “editor” logged in

Now login with username *user* and password ‘1’. You can only view documents, as shown in Figure 8.

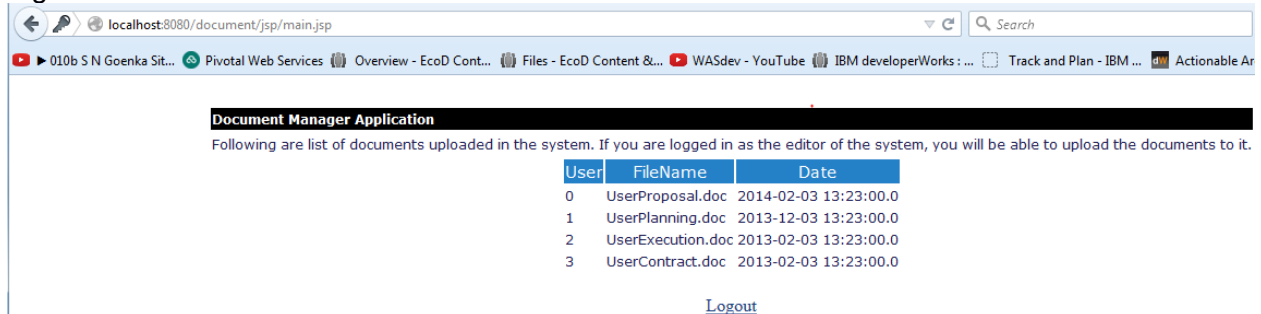


Figure 8: DocumentManager application with username “user” logged in

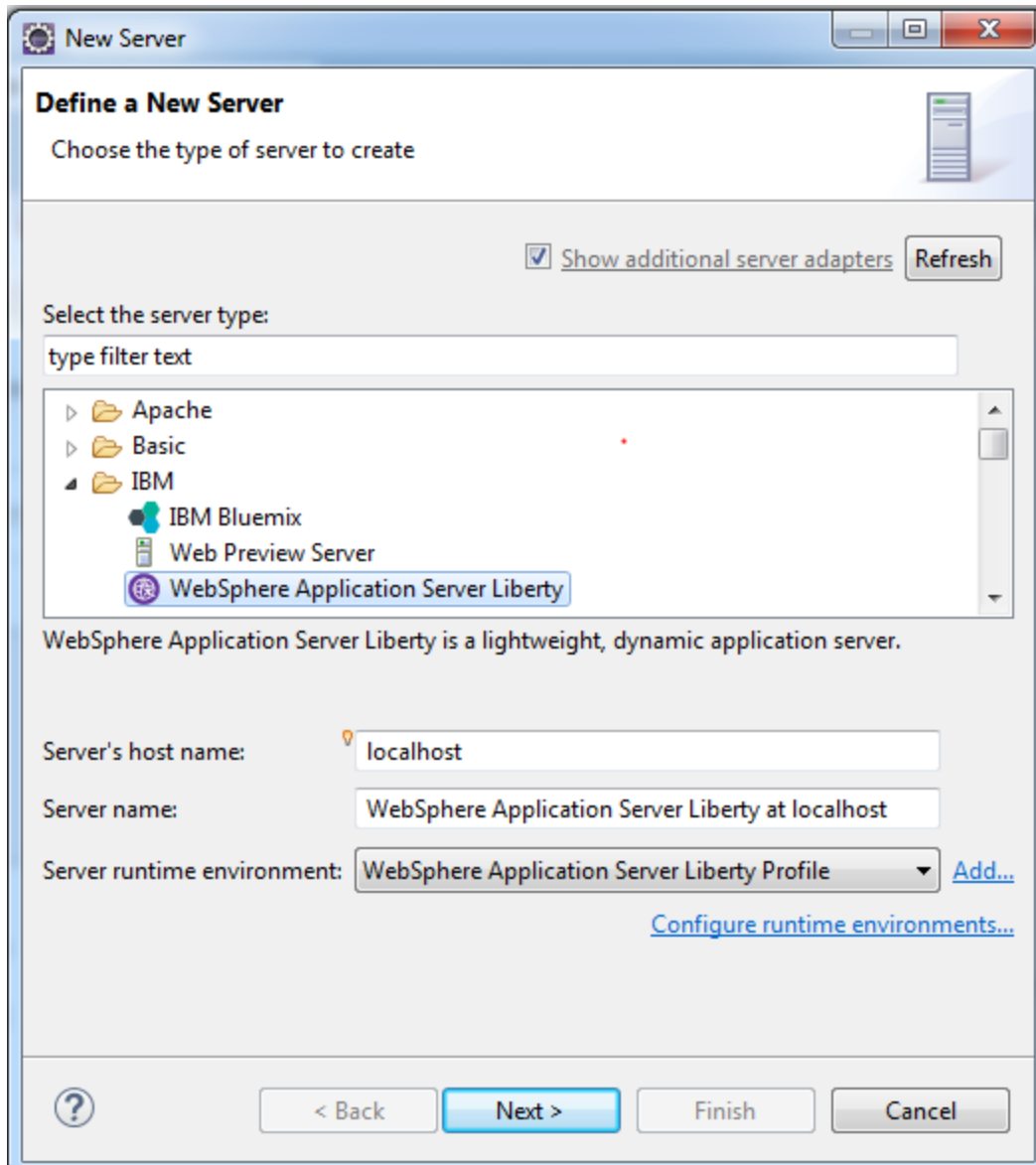
Stop the application now because we will migrate the application to standalone Liberty server.

Step 3: Migrate application to Liberty

We will do migration analysis and identify changes required to move the application to WebSphere Application Server Liberty profile. We want to migrate the development environment, application source code and deployment descriptors, JBoss application server configuration and JBoss custom JAAS login module.

Liberty server is installed at <LIBERTY_ROOT> in the VM as mentioned in the Prerequisites section.

Create Liberty server instance: In Servers view, click **New->Server**, select **WebSphere Application Server Liberty**



Click **Next->Next->Finish**.

Analyze the application

The IBM WebSphere Application Server Migration Toolkit – Competitive Tool 8.5.5 is already installed in the Eclipse instance that you are running. You can verify this, by clicking on **Installed** tab in **Eclipse Marketplace**, as shown in Figure 9.

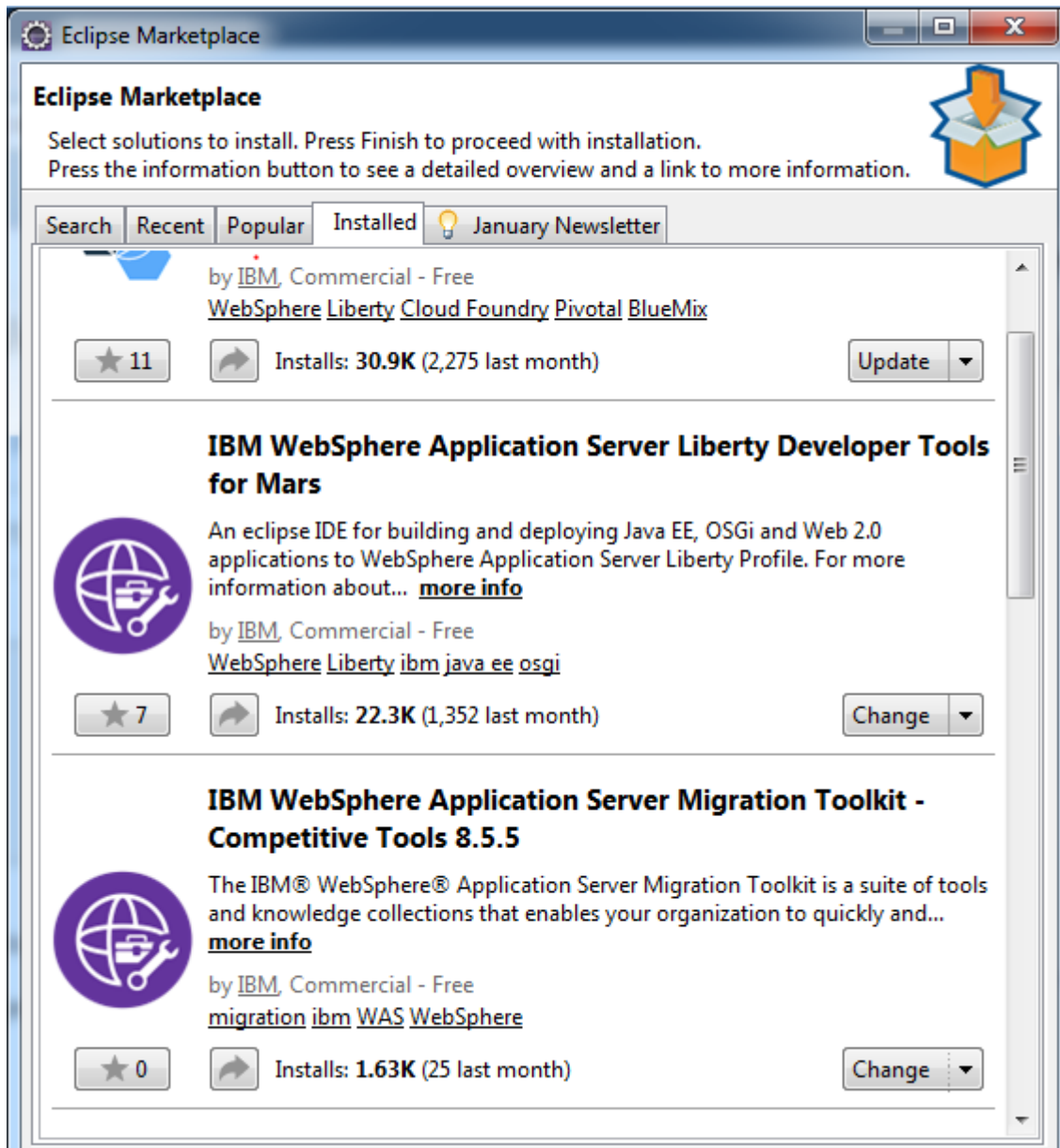
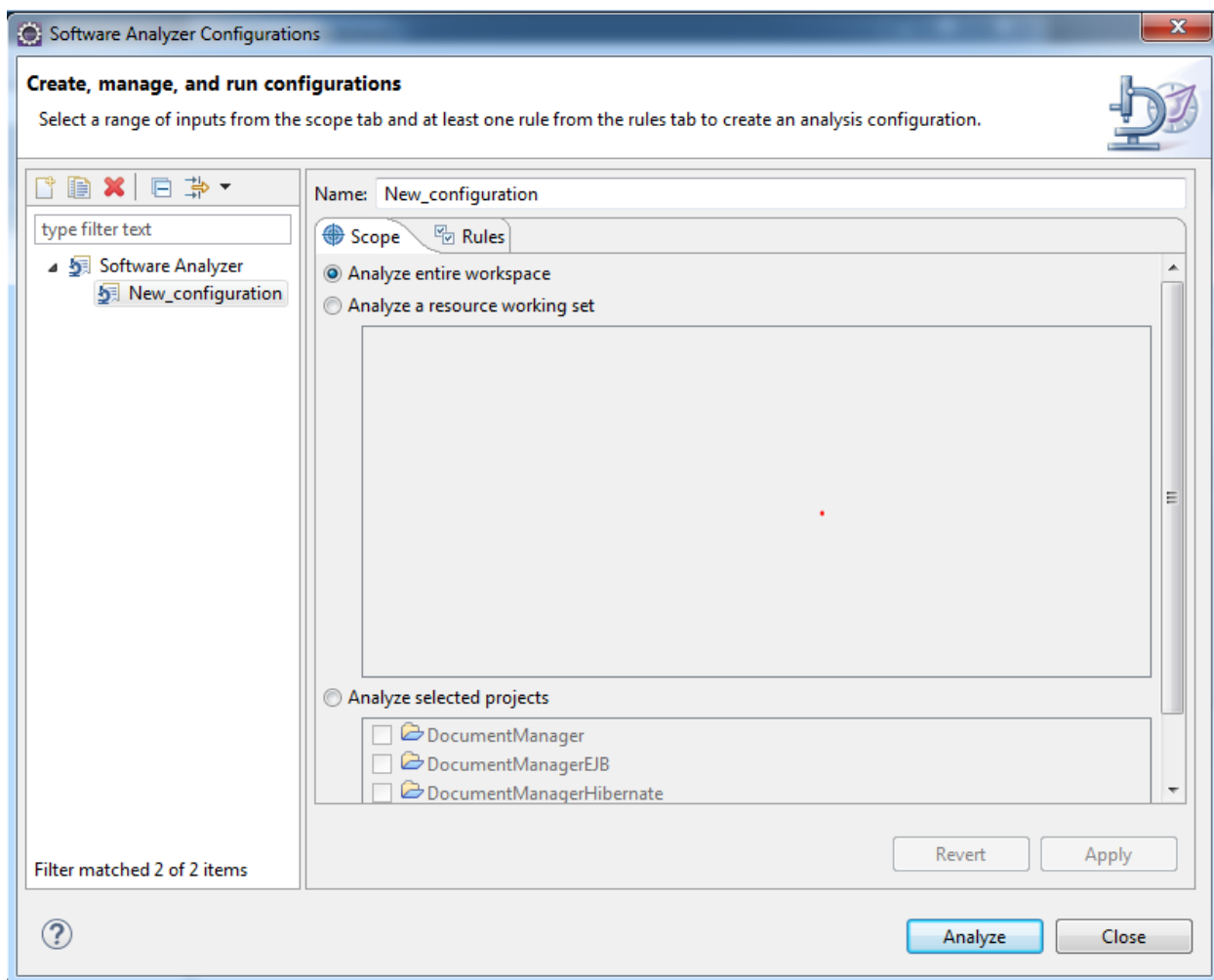


Figure 9: Eclipse Marketplace: WAS Migration Tools

Now right click on **DocumentManager** project and select **Software Analyzer->Software Analyzer Configurations**. Double click on **Software Analyzer** to create a new configuration



For **Name**, enter Liberty Technology Rules.

On the scope tab, select **Analyze entire workspace** to scan all projects in the workspace as shown in Figure 10.

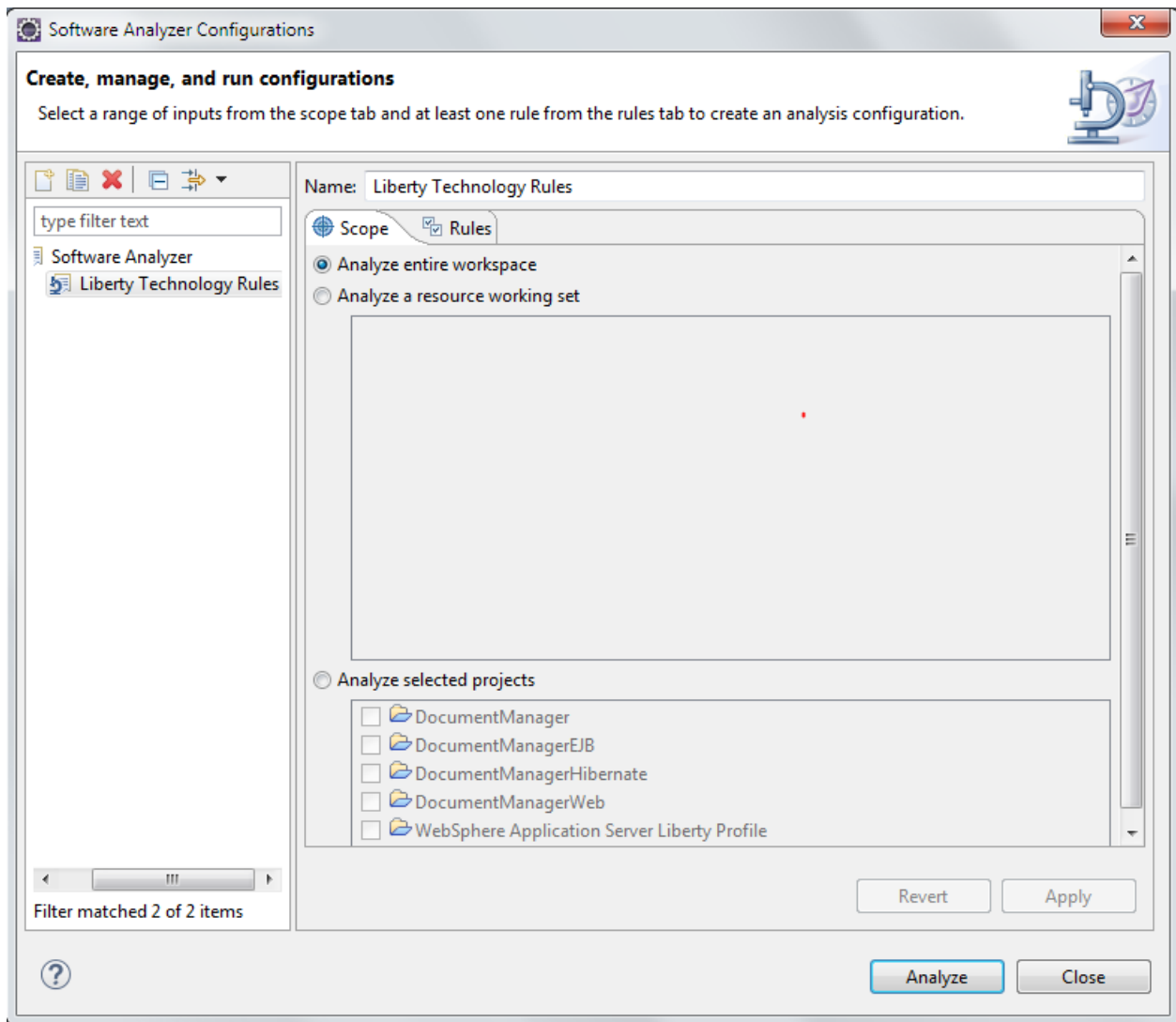


Figure 10: Software Analyzer: Scope view

On the Rules tab, as shown in Figure 11, select **JBoss Application Migration** using the **Rule Sets** list.

Next, click the **Set** button as shown below. Select all the **Analysis Domains** except **Java Architectural Discovery**.

For the **Target application server**, select **Liberty for Java on IBM Bluemix**, for **Source Java Version** choose **Oracle Java 8**, and for Target Java version choose **IBM Java 7**.

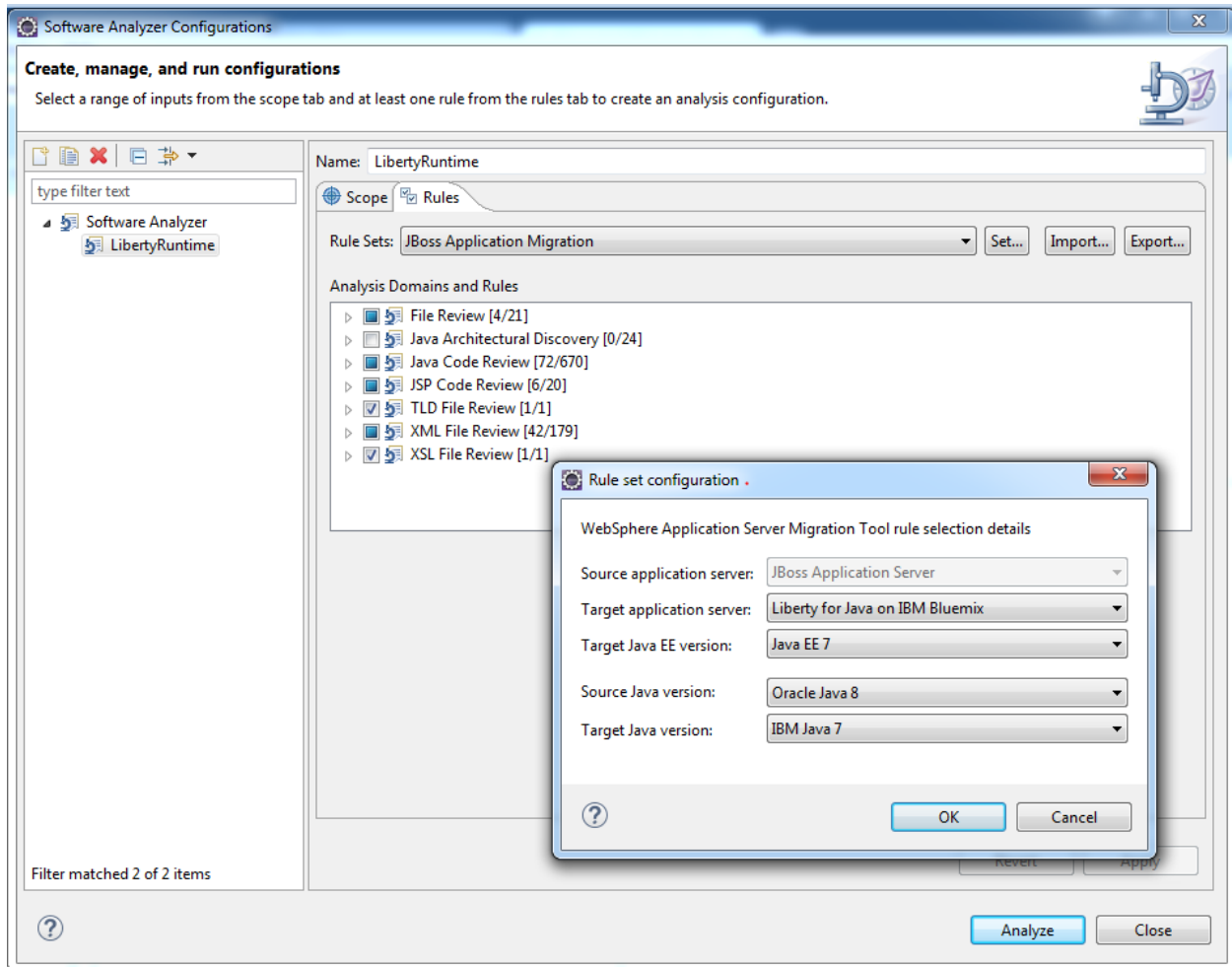
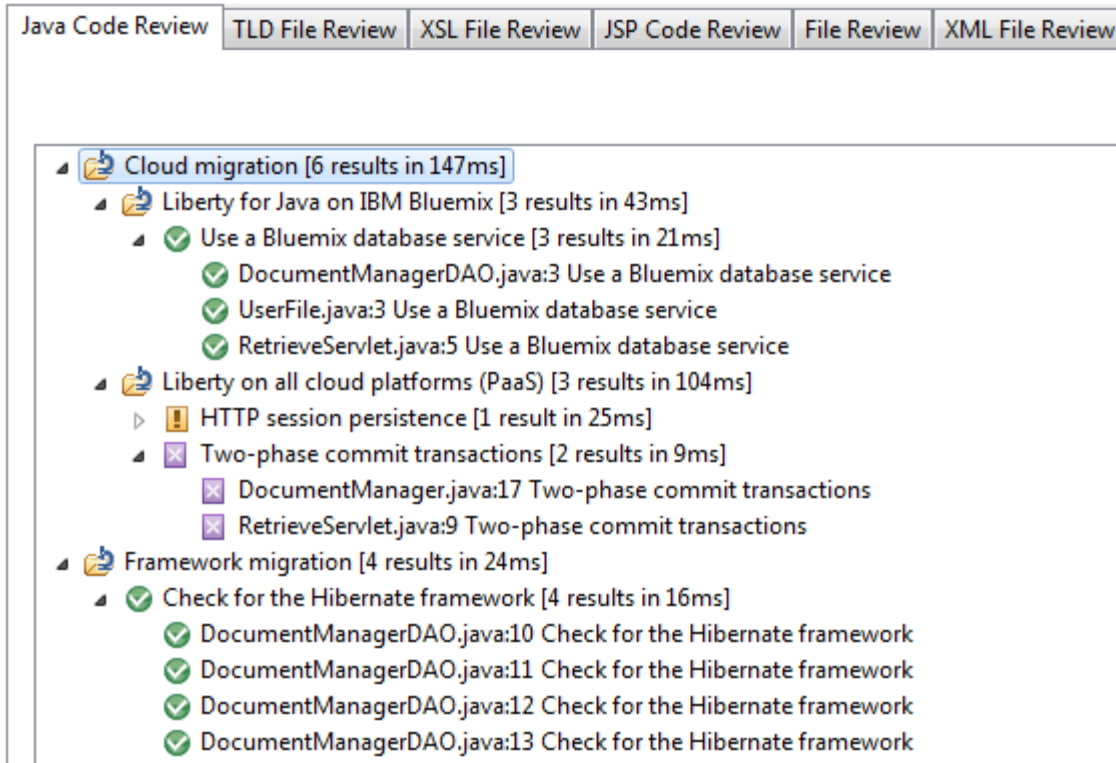


Figure 11: Software Analyzer: Rules view

Click **OK** and **Apply**.

Click the **Analyze** button. The tool will analyze the application and generate a list of potential problems in the Software Analyzer Results view. The results under Java code review tab are shown below.



The results under the XML File Review tab are shown in Figure 12.

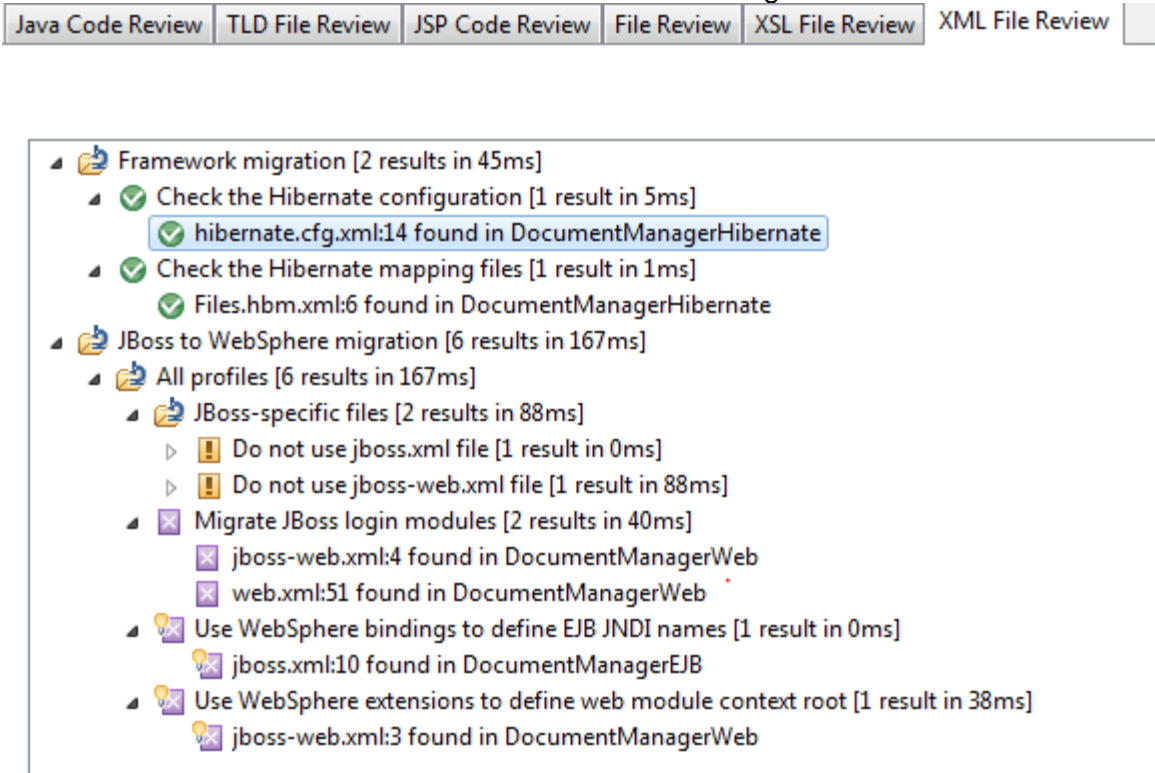


Figure 12: Software Analyzer Results: XML File Review view

The Green Check symbol implies that there is no need to migrate that code. You can click on the row with the Filename and line number to see the source code.

The yellow exclamation symbol, as shown in Figure 13, tells us that we cannot use those files while running the application on Liberty server:

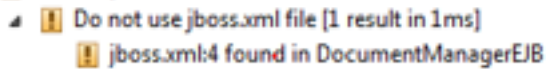



Figure 13: XML File Review: Do Not Use files

The purple  symbol, as shown in Figure 14, indicates that the code has to be fixed to run the application on Liberty server.

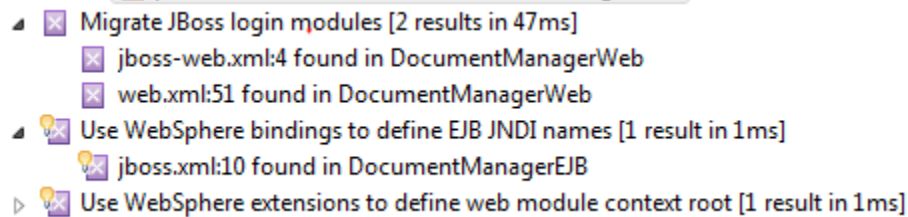




Figure 14: XML File Review: files that need to be fixed

And the yellow with purple cross  symbol means that the Toolkit provides a Quick Fix and will apply that to the code, once you right click on the required change and select Quick Fix.

Right click on  `jboss.xml:10 found in DocumentManagerEJB` and select Quick Fix Preview (Figure 15).

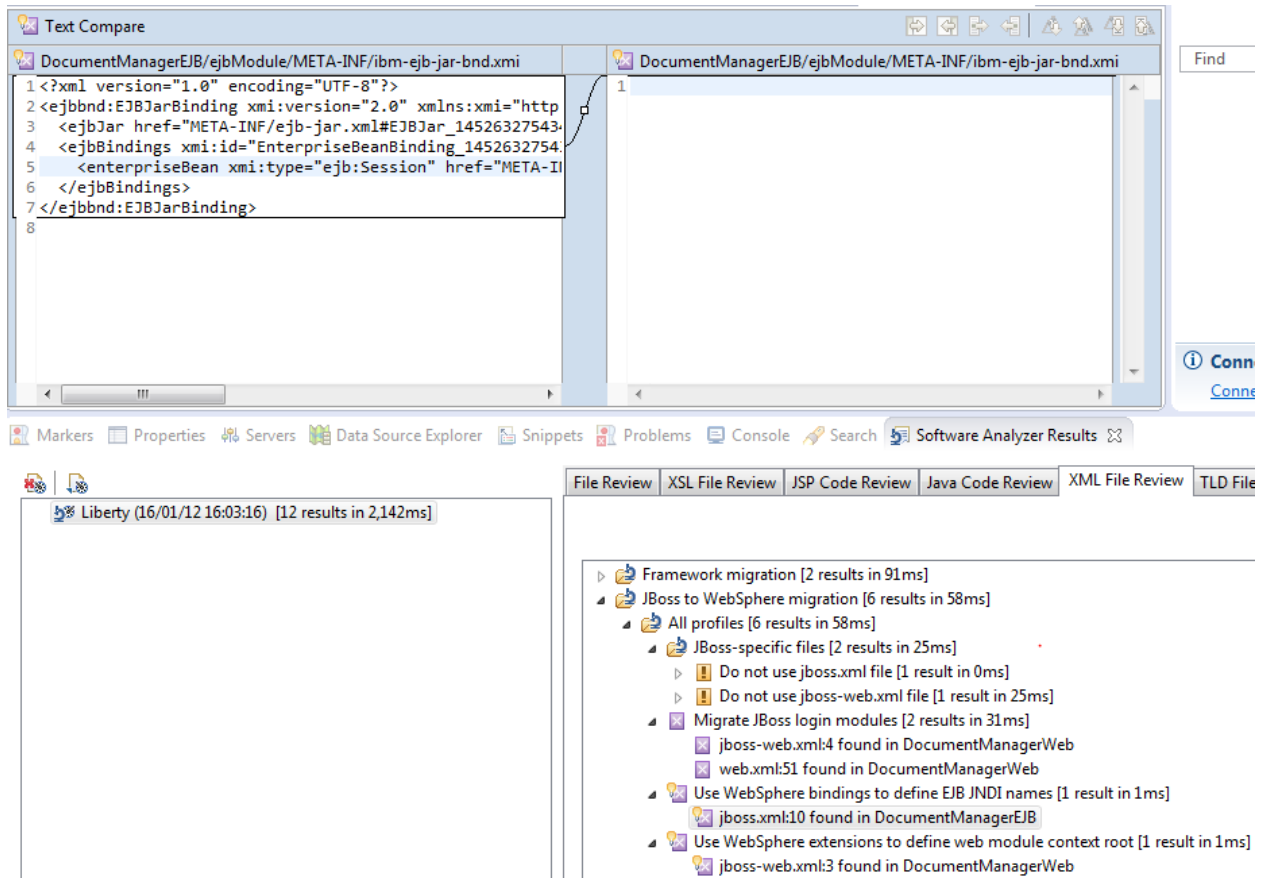


Figure 15: Quick Fix Preview for DocumentManagerEJB

The tool will create ibm-ejb-jar-bnd.xml deployment descriptor under DocumentManagerEJB/ejbModule/META-INF directory with the content shown above. Next, right click and select Quick Fix to apply the code changes. After you apply the code changes, the **jboss.xml:10 found in DocumentManagerEJB** will be removed from the Software Analyzer results.

Next, right click on **jboss-web.xml:3 found in DocumentManagerWeb** and select Quick Fix Preview (Figure 16).

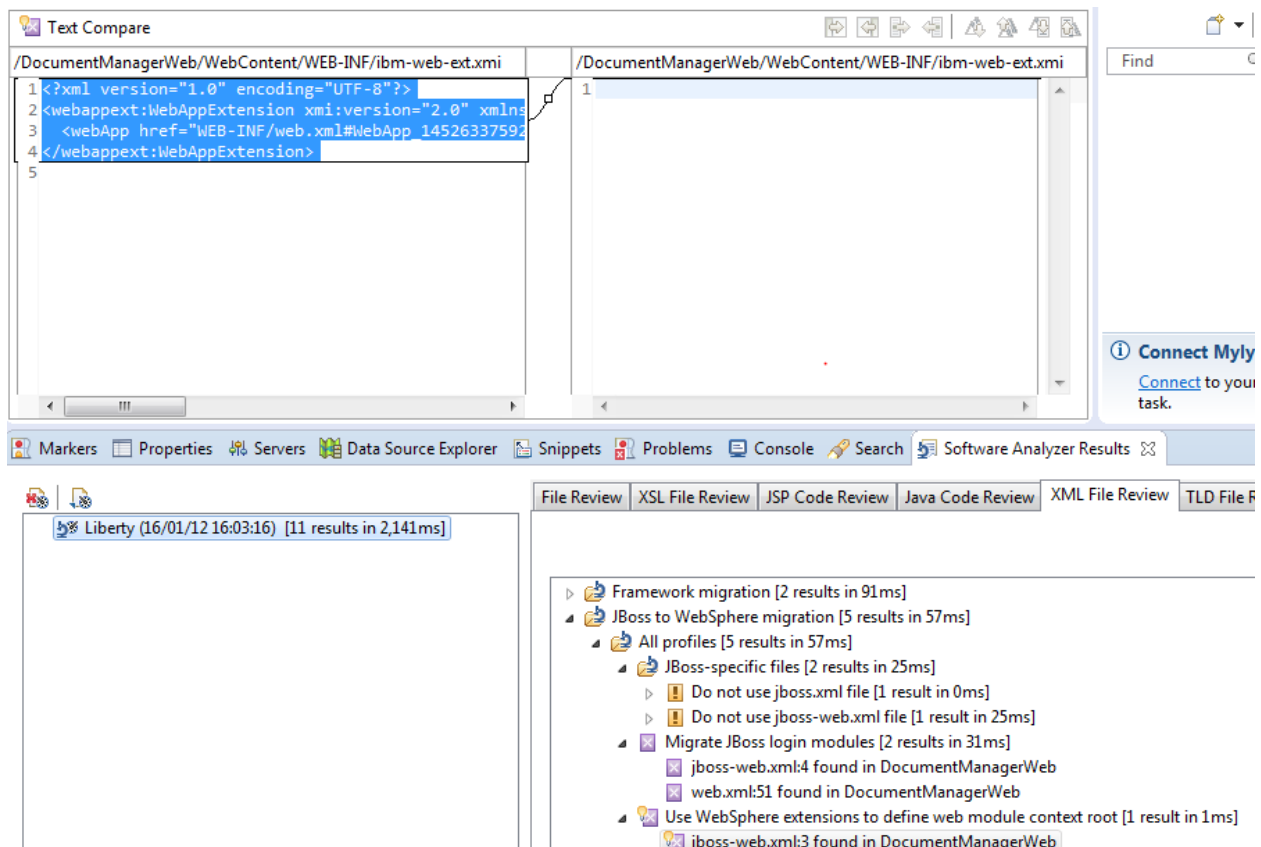




Figure 16: Quick Fix Preview for DocumentManagerWeb

The tool will create `ibm-web-ext.xml` deployment descriptor under `DocumentManagerWeb/WebContent/WEB-INF` directory. Next, right click and select Quick Fix to apply the code changes. After you apply the code changes, the entry

 `jboss-web.xml:3 found in DocumentManagerWeb` will be removed from the Software Analyzer results.

Highlight each entry in the Software Analyzer view and press F1, you will see explanation on why the code change is required,

For example, press F1 on  `Do not use jboss-web.xml file [1 result in 25ms]` and click on **Detailed Help** to display a detailed explanation (Figure 17).

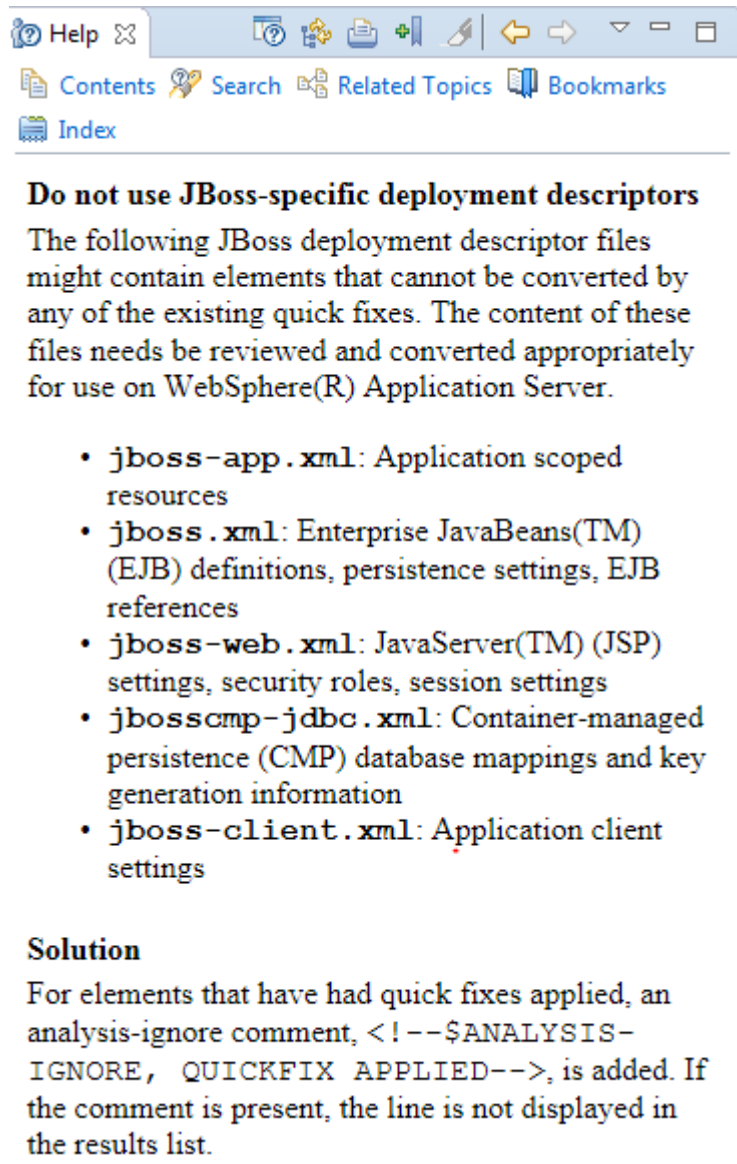


Figure 17: JBoss deployment descriptor migration help

Summary of Software Analyzer results

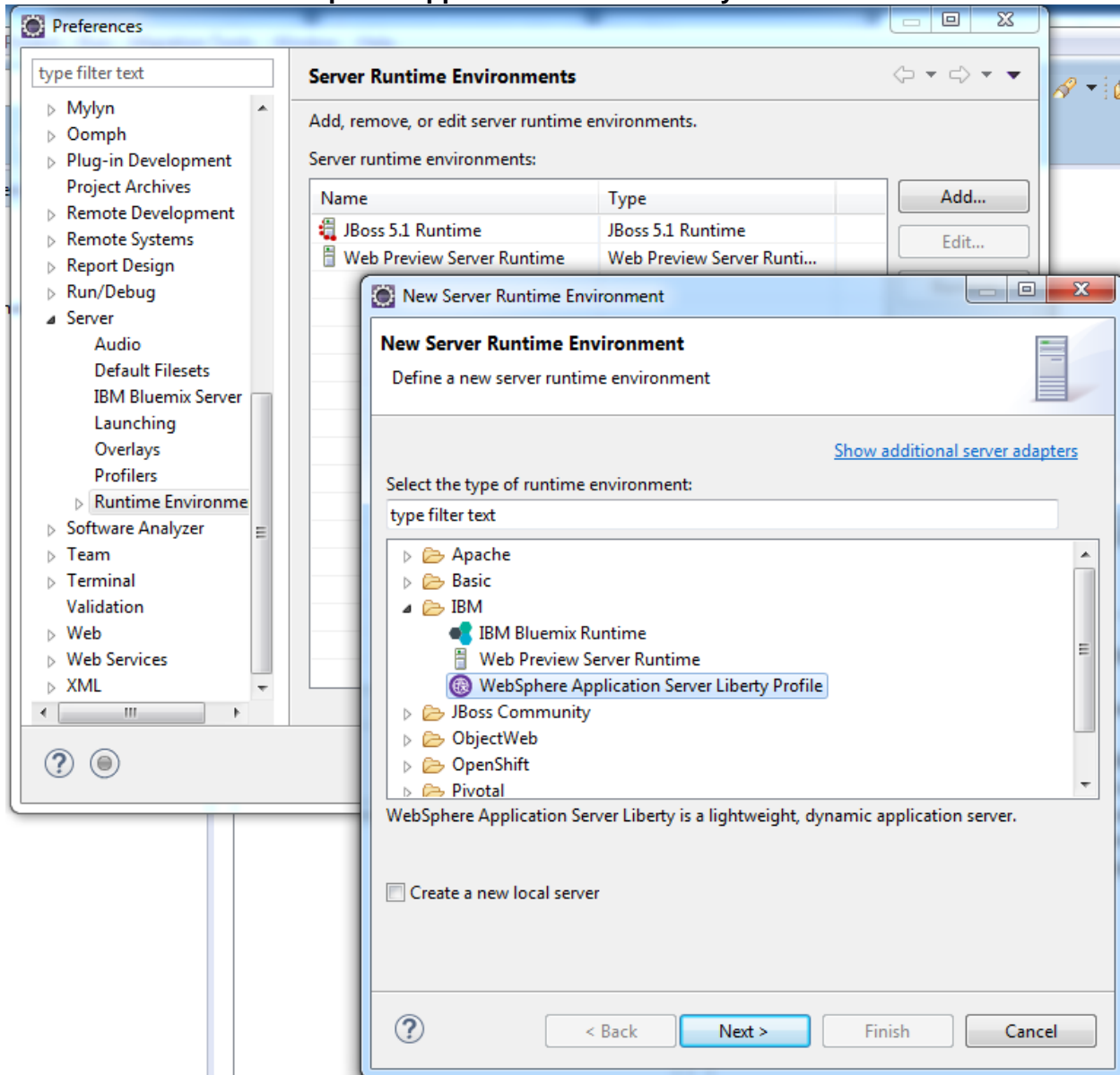
Type	Issue	Component	Description
Java, XML	Check for Hibernate framework	DocumentManagerDAO.java, Hibernate.cfg.xml in DocumentManagerHibernate	Hibernate framework is used by the application. In general, no further action is needed during migration, but it is recommended to make sure WebSphere Application Server best practices for Hibernate are used before going to production.
XML	JBoss specific files	jboss.xml, jboss-web.xml	JBoss specific deployment descriptor information needs to be migrated to be used on Liberty.
XML	Use WebSphere bindings to define EJB JNDI names	jboss.xml	A JNDI name is defined for each EJB home object (as EJB 2.1 mandates). The binding definition needs to be migrated to WebSphere Application Server.
XML	Use WebSphere extensions to define web module context root	jboss.xml	Context root need to be migrated from the JBoss file to WebSphere Application Server web module extensions.
XML	Remote interfaces for Enterprise JavaBeans(EJB) are unavailable	ejb-jar.xml	Remote EJB interfaces are not supported in the Liberty profile.
XML	Enterprise JavaBeans (EJB) 1.x/2.x unavailable	ejb-jar.xml	Enterprise JavaBeans (EJB) 1.x/2.x are not supported in Liberty profile or Liberty Core. Upgrade the application to use the EJB 3.1 Lite specification.

Figure 18: Software Analyzer results

Define WebSphere Liberty runtime in Eclipse

In Eclipse, **Windows->Preferences->Server->Runtime Environments**

Click **Add** and select **WebSphere Application Server Liberty Profile** as shown below:



Upgrade the specification levels for modules in the application

To do this, modify the project facets and Java version for the Eclipse project. These changes will result in the upgrade of deployment descriptors and will enable you to associate the project with a target runtime. Specifying a target runtime adds the supported runtime Java EE libraries to the project's build path.

Right click on the EAR project and select **Document Manager->Properties->Project Facets**. See Figure 17.

On runtimes tab, uncheck JBoss runtime.

Set EAR version to 6.0. From the runtimes tab select **WebSphere Liberty Profile** and click **OK**

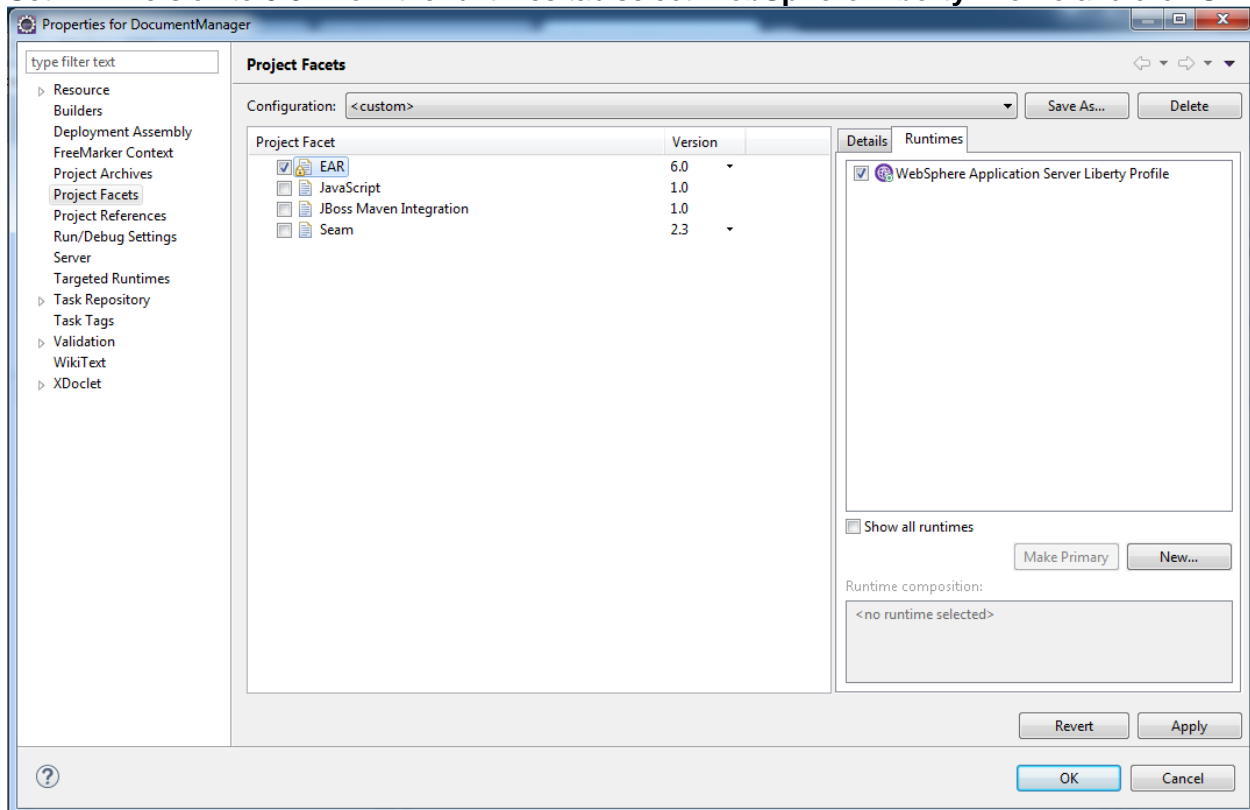


Figure 17: Project Facets: EAR view

To change the project facets for the EJB module, right click on the EJB project and select **DocumentManagerEJB->Properties->Project Facets**.

On runtimes tab uncheck JBoss runtime.

Change EJB Module to **3.1** and change Java to **1.8**. See Figure 18.

On Runtimes tab, select **WebSphere Application Server Liberty Profile** and click **OK**.

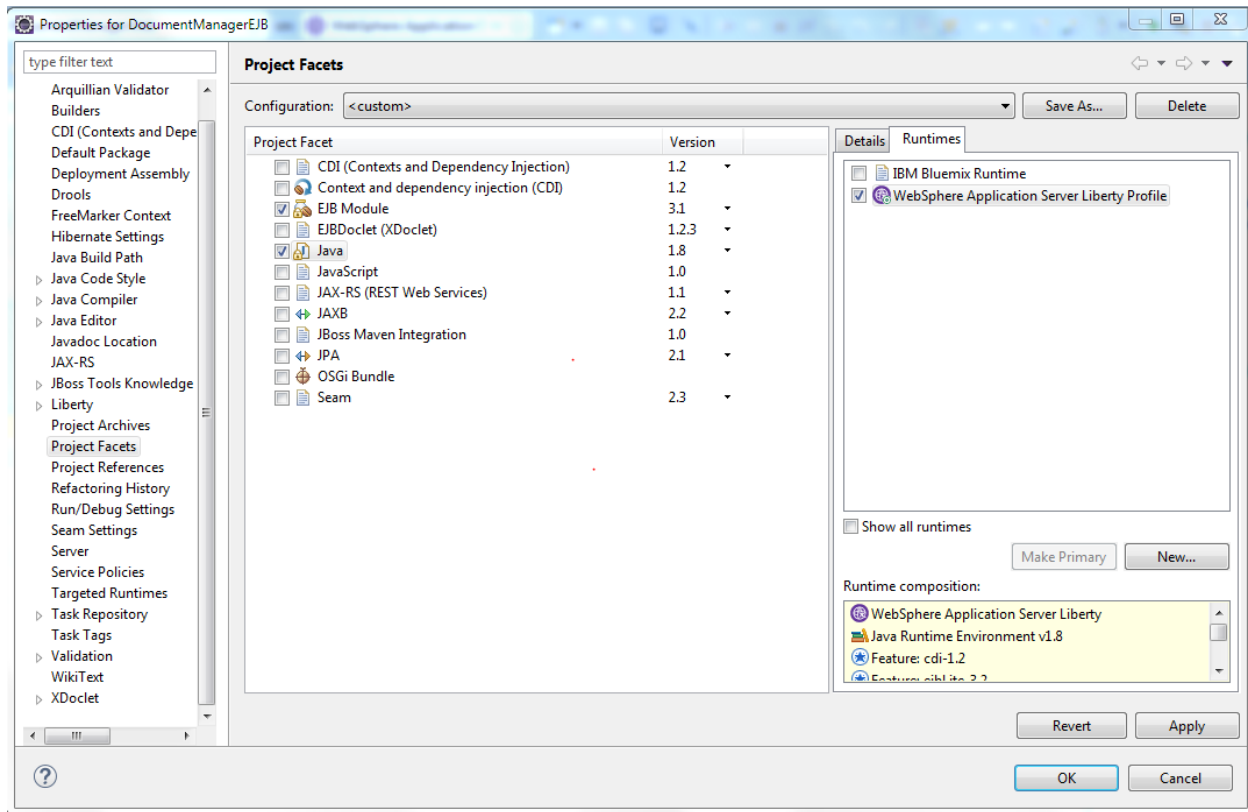


Figure 18: Project Facets: EJB Module view

To change the project facets for the web module, right click on **DocumentManagerWeb->Properties->Project Facets**. See Figure 19.

On runtimes tab, uncheck JBoss runtime.

Change Dynamic Web Module to **3.0** and change Java to **1.8**.

On the Runtimes tab, select **WebSphere Application Server Liberty profile** and click **OK**.

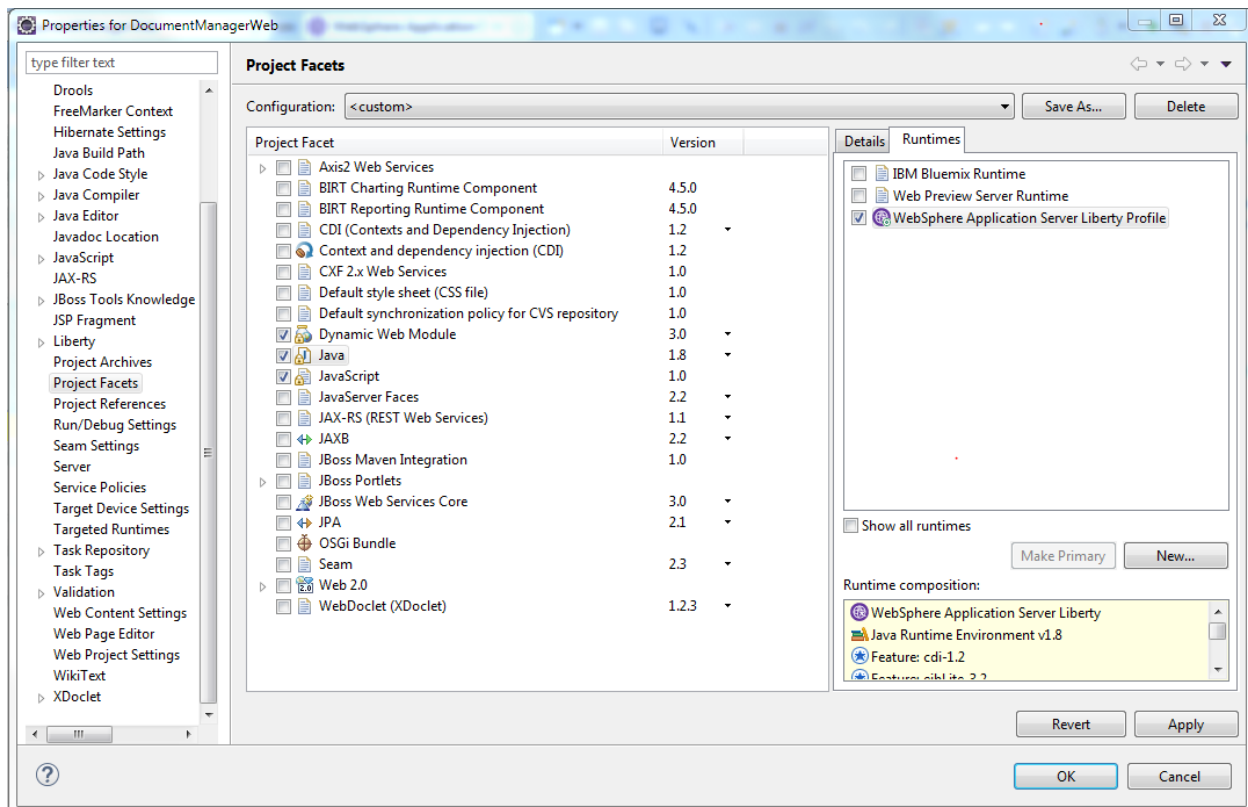


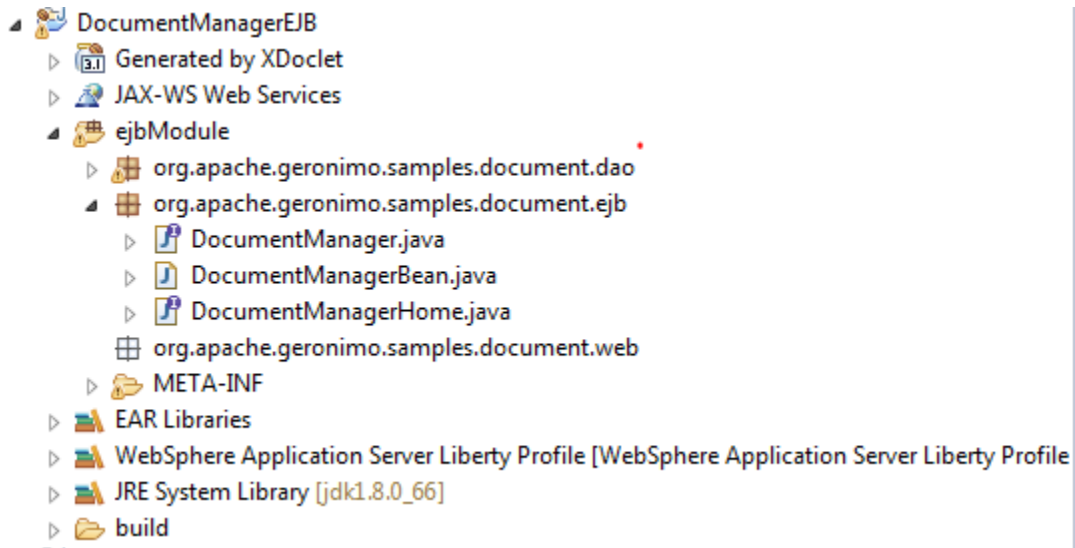
Figure 19: Project Facets: Java view

Migrate application code

All the code snippets discussed in this section and sub sections are available in **CodeSnippets.txt** in **<CLONED_DIR>**. Please copy and paste as required.

Let's look at upgrading the stateless session bean from EJB 2.1 to EJB 3.1. There are many ways to upgrade EJBs from 2.x to 3.x. The steps below illustrate one such way with no interface view.

1. Implement session bean changes
2. The session bean class **org.apache.geronimo.samples.document.ejb.DocumentManagerBean** in **DocumentManagerEJB** project implements the session bean interface.



3. The session bean class has an `ejbCreate()` method, three business methods (`upload()`, `getFilesByUserId()`, `addUserFile()`) and the callback methods, shown in Figure 20.

```

1 package org.apache.geronimo.samples.document.ejb;
2
3 import java.rmi.RemoteException;
4
5 /**
6  * @ejb.bean name="DocumentManager"
7  *           description="DocumentManager"
8  *           display-name="DocumentManager"
9  *           jndi-name="ejb/DocumentManager"
10 *           type="Stateless"
11 *           transaction-type="Container"
12 *           view-type="remote"
13 */
14
15 * @ejb.interface generate="remote" remote-class="org.apache.geronimo.samples.document.ejb.DocumentManager"
16 * @ejb.home generate="remote" remote-class="" -class="org.apache.geronimo.samples.document.ejb.DocumentManagerHome"
17 */
18
19 public class DocumentManagerBean implements javax.ejb.SessionBean {
20
21     /**
22      *
23      */
24     /**
25      *
26      */
27     private static final long serialVersionUID = 5910882211249828856L;
28
29     /**
30      * @ejb.interface-method view-type="remote"
31      * @ejb.permission role-name = "uploader"
32      * @return result message
33      */
34
35     public String upload(String userId, String filename) {
36         addUserFile(userId, filename);
37         return "File successfully uploaded";
38     }
39
40     public List<UserFile> getFilesByUserId(String userid) {
41         List<UserFile> list = null;
42         try {
43             DocumentManagerDAO dmDao = new DocumentManagerDAO();
44             list = dmDao.getUserFilesByUserId(userid);
45         } catch (Exception e) {
46             // TODO Auto-generated catch block
47             e.printStackTrace();
48         }
49         return list;
50     }
51
52     private void addUserFile(String userId, String filename) {
53         DocumentManagerDAO dmDao;
54     }
55 }

```

Figure 20: EJB 2.x stateless session bean

EJB 3.1 bean classes do not implement `javax.ejb.SessionBean`. Therefore, the lifecycle methods need not be implemented.

The migrated EJB 3.1 session bean (`DocumentManagerBean`) corresponding to the EJB 2.1 stateless session bean is shown in Figure 21. While converting the EJB 2.x bean to the EJB 3 no interface view, there is no need to carry over the EJB 2.x interfaces. Go ahead and delete these two interfaces:

`org.apache.geronimo.samples.document.ejb.DocumentManager`

`org.apache.geronimo.samples.document.ejb.DocumentManagerHome`

```

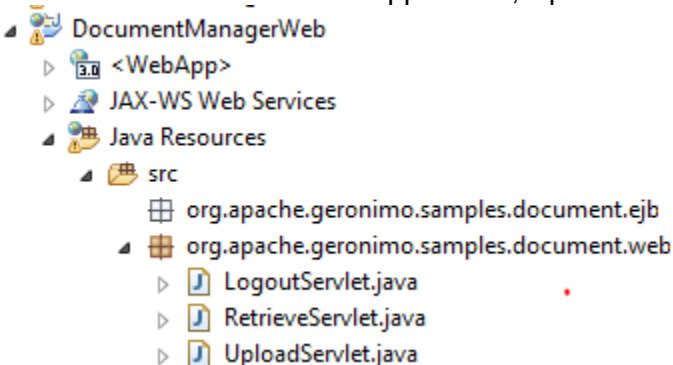
1 package org.apache.geronimo.samples.document.ejb;
2 import java.util.List;
3
4
5
6
7
8
9 @Stateless
10 public class DocumentManagerBean{
11
12     public String upload(String userId, String filename) {
13         addUserFile(userId, filename);
14         return "File successfully uploaded";
15     }
16
17     public List<UserFile> getFilesByUserid(String userid) {
18         List<UserFile> list = null;
19         try {
20             DocumentManagerDAO dmDao = new DocumentManagerDAO();
21             list = dmDao.getUserFilesByUserid(userid);
22         } catch (Exception e) {
23             // TODO Auto-generated catch block
24             e.printStackTrace();
25         }
26         return list;
27     }
28     private void addUserFile(String userId, String filename) {
29         DocumentManagerDAO dmDao;
30
31         try {
32             dmDao = new DocumentManagerDAO();
33             dmDao.addUserFile(userId, filename);
34         } catch (Exception e) {
35             // TODO Auto-generated catch block
36             e.printStackTrace();
37         }
38     }
39 }

```

Figure 21: EJB 3.x stateless session bean with no interface view

Inject session bean into servlets

There are two servlets in this application, UploadServlet and RetrieveServlet.



The changes to these servlets are two-fold.

First, you annotate the servlet class with the `@WebServlet` annotation to indicate classes that are servlets.

Second, you use the `@EJB3` annotation to inject the EJB rather than doing the JNDI lookup of the home interface. Homes are not required in EJB3.

Figure 22 illustrates the changes you made to the `UploadServlet` class. First, you added the `@WebServlet` annotation above the servlet class, added the `@EJB` annotation to inject the session bean into the servlet and removed the JNDI home lookup in the `init()` method. You also removed the call to the `create()` method to create the EJB bean instance. The changes to `RetrieveServlet` are similar.

```

1  package org.apache.geronimo.samples.document.web;
2
3  import java.io.IOException;
4
5  @WebServlet("/UploadServlet")
6  public class UploadServlet extends HttpServlet{
7
8      private static final long serialVersionUID = -2773446199481416101L;
9      @EJB
10     private DocumentManagerBean docManager;
11
12     // public void destroy() {
13     //     docManagerHome = null;
14     //     super.destroy();
15     // }
16
17     public void init() throws ServletException {
18     //     try {
19     //         Context context = new InitialContext();
20     //         docManagerHome = (DocumentManagerHome)context.lookup(DocumentManagerHome.JNDI_NAME);
21     //     } catch (Throwable e) {
22     //         throw new ServletException(e);
23     //     }
24     // }
25
26     protected void doGet(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
27     //     DocumentManager docManager = getDocumentManager();
28     //     String filename = req.getParameter("file");
29     //     String userID = req.getUserPrincipal().getName();
30     //     req.setAttribute("result", docManager.upload(userID, filename));
31     //     req.getRequestDispatcher("jsp/main.jsp").forward(req, res);
32     // }
33
34     protected void doPost(HttpServletRequest req, HttpServletResponse res) throws ServletException, IOException {
35     //     doGet(req, res);
36     // }
37
38     // private DocumentManager getDocumentManager() throws ServletException, IOException {
39     //     try {
40     //         return docManagerHome.create();
41     //     } catch (CreateException e) {
42     //         throw new ServletException(e);
43     //     }
44     // }
45 }

```

Figure 22: Modified Upload Servlet

Add Logout fix to Logout controller

1. In `DocumentManagerWeb` project, `LogoutServlet` class add this code to `Logout` method:

```

private void logOut(HttpServletRequest req, HttpServletResponse res)
throws ServletException, IOException {
    HttpSession session = req.getSession();
    session.invalidate();
}

```

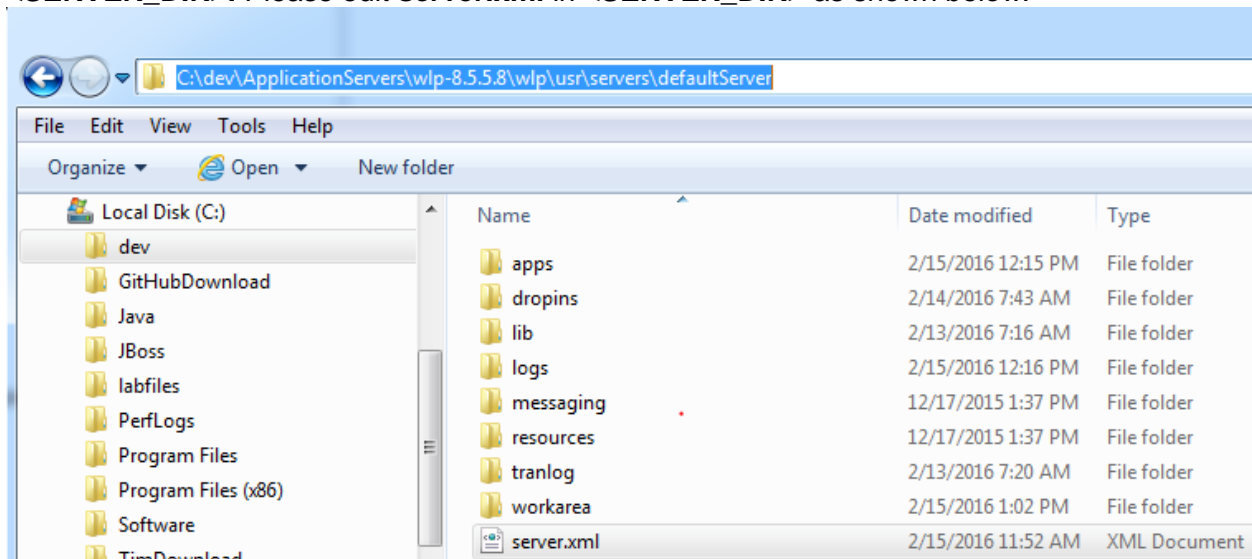
```
req.logout();

res.sendRedirect( req.getContextPath() + "/" );
return;
}
```

Update Liberty Server Configuration

As a lightweight container, Liberty only adds those features that are required by the application. The configuration migration toolkit, cannot determine the features that your application uses, so you will need to add them manually. Navigate to

<LIBERTY_ROOT>/wlp/usr/servers/defaultServer. This directory will be called **<SERVER_DIR>**. Please edit **server.xml** in **<SERVER_DIR>** as shown below.



These features need to be added in server.xml are :

```
<featureManager>
  <feature>javaee-7.0</feature>
  <feature>ejbLite-3.2</feature>
  <feature>appSecurity-2.0</feature>
  <feature>servlet-3.1</feature>
  <feature>jsp-2.3</feature>
  <feature>jdbc-4.1</feature>
  <feature>localConnector-1.0</feature>
</featureManager>
```

Add JDBC driver configuration to server.xml

```
<library id="MySQLDriverLib">
    <fileset dir="${server.config.dir}/lib" id="mysql-fileset"/>
</library>
<dataSource id="DefaultDataSource" transactional="true"
type="javax.sql.ConnectionPoolDataSource">
    <jdbcDriver libraryRef="MySQLDriverLib"/>
    <properties databaseName="ic16_Lab2434" id="mysql-mydb-props"
password="object00" portNumber="3306" serverName="localhost" user="root"/>
    <connectionManager id="mysql-mydb-conMgr" maxPoolSize="10"/>
</dataSource>
```

Copy MySQL JDBC driver file to server

Create **<SERVER_DIR>/lib** directory if it does not exist.

Copy **<CLONED_DIR>/mysql-connector-java-5.1.38-bin.jar** to **<SERVER_DIR>/lib**.

Map Roles to users and groups in Liberty

The JBoss service archive defines the users and groups and stores them in a couple of files. SAR is a proprietary JBoss archive that contains authentication and authorization (e.g. users, groups). In JBoss, there is no way to map the roles that are used in web.xml to the users and groups that are defined in the service archive.

Mapping roles to users and groups in Liberty is a three-step process. First, you define the users and groups (the user registry) in the server.xml file, then you map the roles (defined in the web.xml file) to the groups defined in the ibm-application-bnd.xml file and at that point you can remove the proprietary JBoss SAR project.

In this application, **instead of creating an ibm-application-bnd.xml** file, to make it easy, we are adding the information for mapping the roles to the defined groups in **server.xml**.

Configure a basic user registry with user and editor entries. The group and user information is coming from the j2g_groups.properties and the j2g_users.properties files in the SAR project.

Add basicRegistry to server.xml

```
<basicRegistry id="basic" realm="BasicRealm">
  <user name="user" password="1"/>
  <user name="editor" password="2"/>
  <group name="authenticated">
    <member name="user"/>
    <member name="editor"/>
  </group>
  <group name="uploader">
    <member name="editor"/>
  </group>
</basicRegistry>
```

Figure 23: basicRegistry element

Map security roles to groups using server.xml

Configure the application bindings that map roles to users and groups in the user registry or to special subjects.

```
<application type="ear" id="DocumentManager" name="DocumentManager"
location="${server.config.dir}/apps/DocumentManager.ear">
  <application-bnd>
    <security-role name="authenticated">
      <special-subject type="ALL_AUTHENTICATED_USERS"/>
    </security-role>
    <security-role name="uploader">
      <group name="uploader"/>
    </security-role>
  </application-bnd>
</application>
```

Figure 24: Application security roles

Remove the SAR project from the EAR project. Right click the **DocumentManagerSAR** project and click **Delete**. Select the **Delete project contents on disk check box** and click **OK**.

Migrate Hibernate configuration

Change Transaction Management to Bean and add transaction code

1. In DocumentManagerBean class add this annotation.

```
@TransactionManagement(value=TransactionManagementType.BEAN)
public class DocumentManagerBean {
```

2. In DocumentManagerDAO class, replace all occurrences of `factory.getCurrentSession()` to `factory.openSession()`.
3. In DocumentManagerDAO class, add transaction control statements in `addUserFile` method around `session.save(userfile)` statement.

```
Transaction tx;  
tx = session.beginTransaction();  
session.save(userfile);  
tx.commit();
```

Deploy and test the application on Liberty

Export the DocumentManager project as an EAR file.

Save the DocumentManager.ear in the server's <SERVER_DIR>\apps folder. (Figure 25)



Figure 25: DocumentManagement.ear folder

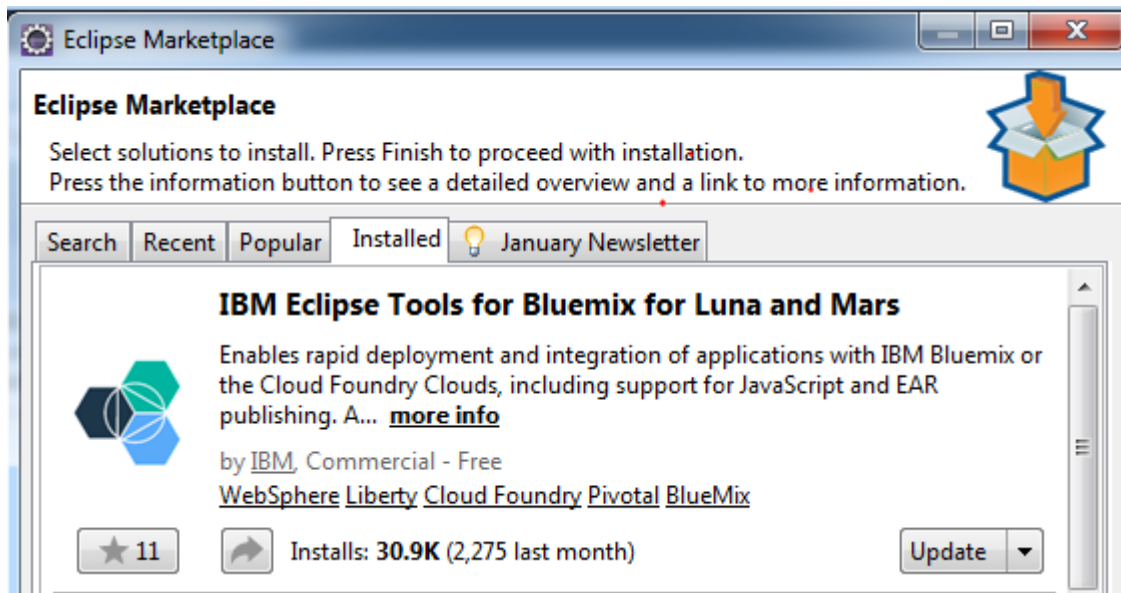
Start the Liberty server. In web browser navigate to the application URL:
localhost:9080/document

Make sure MySQL database is running. After successful login, the application redirects to the Documents page. Observe that only the editor can upload the documents based on security role mappings. After testing, please stop the Liberty Server running in Eclipse.

Step 4: Deploy to Bluemix

Using Eclipse to deploy your application to Bluemix is virtually the same as doing a local deploy to Liberty server. There is just one-time setup to get started, and then your Bluemix account will look just like any other server available to deploy and run your application.

The **IBM Eclipse Tools for Bluemix for Luna and Mars** is already installed in Eclipse as shown below.



Now, you can create a Bluemix server. In the **Servers** tab, right-click, select **New >Server**, and select **IBM > IBM Bluemix**. Click **Next**.

Enter your account information and click **Validate Account**. Follow the prompts to create the server.

We are choosing to use a database in the cloud, so next we need to create the database service to use. This can be done through the IBM Bluemix dashboard.

Go to bluemix.net, Login using IBM id you used to sign up for Bluemix. Click on Catalog Tab and go to Data & Analytics section. Double Click on ClearDB MySQL Database icon. See Figure 26.

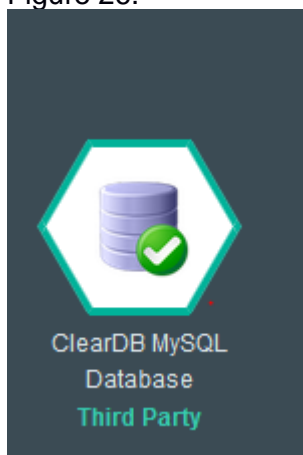
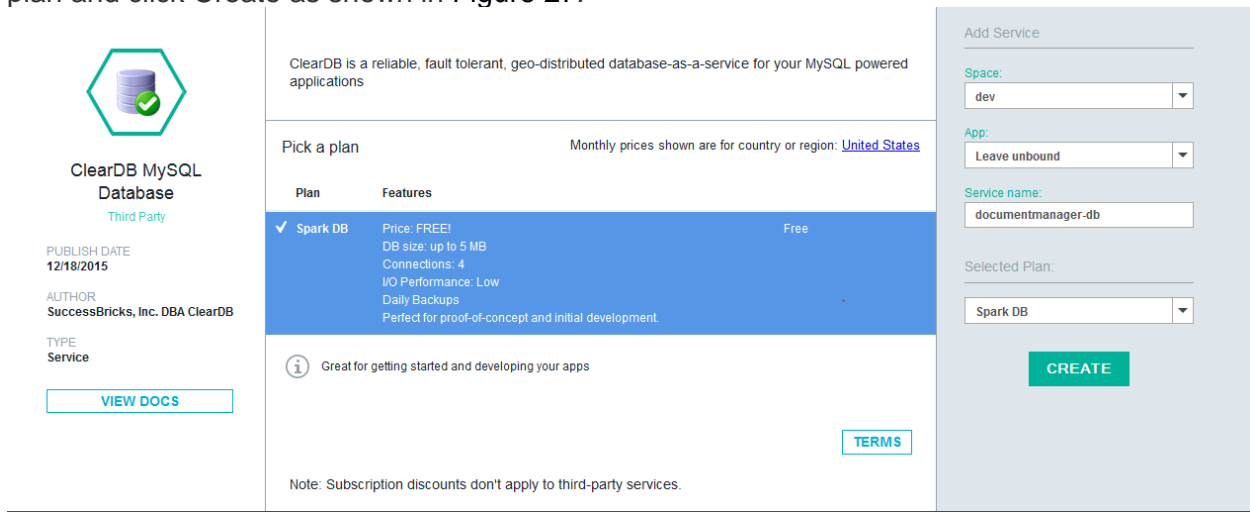


Figure 26: ClearDB MySQL service icon

Leave the database unbound, Name the database as documentmanager-db, Choose SparkDB plan and click Create as shown in Figure 27.



ClearDB is a reliable, fault tolerant, geo-distributed database-as-a-service for your MySQL powered applications

Pick a plan Monthly prices shown are for country or region: [United States](#)

Plan	Features	Price
✓ Spark DB	Price: FREE! DB size: up to 5 MB Connections: 4 I/O Performance: Low Daily Backups Perfect for proof-of-concept and initial development.	Free

Great for getting started and developing your apps

Note: Subscription discounts don't apply to third-party services.

Terms

Add Service

Space:

App:

Service name:

Selected Plan:


CREATE

Figure 27: ClearDB MySQL settings

We need to know the credentials for the ClearDB MySQL instance. To do so, the service needs to be bound to an application.

Click on a Liberty for Java runtime under Runtimes:





Liberty for Java™
IBM

VERSION
2.x

TYPE
Application

[VIEW DOCS](#)

Develop, deploy, and scale Java web apps with ease. IBM WebSphere Liberty Profile is a highly composable, ultra-fast, ultra-light profile of IBM WebSphere Application Server designed for the cloud.

Pick a plan Monthly prices shown are for country or region: [United States](#)

Plan	Features	Price
✓ Default	Run one or more apps free for 30 days (375 GB-hours free)	\$0.07 USD/GB-Hour

This is a service plan for the IBM Bluemix Platform runtime.

[TERMS](#)

Start with a runtime:

Space:

Name:

Host:

Domain:

Selected Plan:

[CREATE](#)

Give a unique name as Host.

Dashboard - IBM Bluemix

<https://console.ng.bluemix.net/?direct=classic/#/resources/appGuid=27e33843-03b9-4f69-9bed-3267a1bcf069&appName=documentmanagersandhya&orgG>

Apps IBM Bluemix - Next...

IBM Bluemix

DASHBOARD SOLUTIONS

Back to Dashboard...

documentmanagersandhya

Overview

Liberty for Java™

Files

Logs

Environment Variables

Start Coding

SERVICES

documentmanagersandhya

Routes: documentmanagersandhya.mybluemix.net

LIBERTY FOR JAVA™

INSTANCES: 1

MEMORY QUOTA: 512

AVAILABLE MEMORY: 4.125 GB

SAVE

RESET





+ ADD A SERVICE OR API

+ BIND A SERVICE OR API

Bind the ClearDB MySQL database that was created earlier, so we can get the service credentials for the database.

Add Service to 'documentmanagersandhya' Application

Select the previously used service instance that you want to add to your application.

	NAME	SERVICE	VERSION
<input type="radio"/>	 dashDB-sl	dashDB	
<input checked="" type="radio"/>	 documentmanage...	cleardb	
<input type="radio"/>	 ForumDB	sqlDb	
<input type="radio"/>	 IoT-hackathon	iotf-service	

ADD

CANCEL

Next, click on **Show Credentials** :

LIBERTY FOR JAVA™

INSTANCES: 1

MEMORY QUOTA: 512 (MB per Instance)

AVAILABLE MEMORY: 4.125 GB

SAVE

RESET

+ ADD A SERVICE OR API

+ BIND A SERVICE OR API

ClearDB MySQL Database

documentma...

spark

Show Credentials +1 Docs

Instantiating Credentials

```
{
  "cleardb": [
    {
      "name": "documentmanager-db",
      "label": "cleardb",
      "plan": "spark",
      "credentials": {
        "jdbcUrl": "jdbc:mysql://us-cdbr-ir",
        "uri": "mysql://b6891886f5b32b:0189",
        "name": "ad_0dc3c9d9ced0dd3",
        "hostname": "us-cdbr-iron-east-03.c",
        "port": "3306",
        "username": "b6891886f5b32b",
        "password": "0189934e"
      }
    }
  ]
}
```

Your service credentials will vary based on your instance. This database instance has been populated for you. If you are using this instance, then skip the next step of populating the database.

Populate ClearDB MySQL database using MySQL command line client:

Add "C:\Program Files\MySQL\MySQL Server 5.7\bin" to PATH.

Execute mysql.exe with hostname, username, password and database name as shown below.

```
mysql.exe -h us-cdbr-iron-east-03.cleardb.net -u b6891886f5b32b -p0189934e
ad_0dc3c9d9ced0dd3
```

Note: there is no space between the letter p and the password.

```

Command Prompt - mysql.exe -h us-cdb-iron-east-03.cleardb.net -u b6891886f5b32b -p0189934e ad_0dc3c9d9ced0dd3
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\tivouser>echo %PATH%
C:\ProgramData\Oracle\Java\javapath;C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;C:\Windows\System32\WindowsP
overShell\w1.0\;C:\Java\jdk1.8.0_66\bin;C:\Java\apache-ant-1.9.6\bin;C:\Program Files (x86)\CloudFoundry;C:\Program File
s\nodejs\;C:\Program Files\Git\cmd;C:\Users\tivouser\AppData\Roaming\npm;C:\dev\Docker\DockerToolbox

C:\Users\tivouser>set PATH=C:\Program Files\MySQL\MySQL Server 5.7\bin;%PATH%

C:\Users\tivouser>echo %PATH%
C:\Program Files\MySQL\MySQL Server 5.7\bin;C:\ProgramData\Oracle\Java\javapath;C:\Windows\system32;C:\Windows;C:\Window
s\System32\Wbem;C:\Windows\System32\WindowsPowerShell\v1.0\;C:\Java\jdk1.8.0_66\bin;C:\Java\apache-ant-1.9.6\bin;C:\Prog
ram Files (x86)\CloudFoundry;C:\Program Files\nodejs\;C:\Program Files\Git\cmd;C:\Users\tivouser\AppData\Roaming\npm;C:\d
ev\Docker\DockerToolbox

C:\Users\tivouser>mysql.exe
ERROR 1045 (28000): Access denied for user 'ODBC'E'localhost' (using password: NO)

C:\Users\tivouser>mysql.exe -h us-cdb-iron-east-03.cleardb.net -u b6891886f5b32b -p0189934e ad_0dc3c9d9ced0dd3
mysql: [Warning] Using a password on the command line interface can be insecure.
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 558598586
Server version: 5.5.45-log MySQL Community Server (GPL)

Copyright (c) 2000, 2015, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> _

```

Let's create the tables and populate the data. In the MySQL command line client, source <CLONED_DIR>\Lab2434_ClearDB.sql as shown below

```

mysql>
mysql>
mysql> source C:\GitHubDownload\MigrateEnterpriseAppFromJBosstoLiberty-master\Lab2434_ClearDB.sql
Query OK, 0 rows affected (0.22 sec)

Query OK, 1 row affected (0.02 sec)

Query OK, 1 row affected (0.02 sec)

Query OK, 1 row affected (0.02 sec)

Query OK, 1 row affected (0.02 sec)

Query OK, 1 row affected (0.02 sec)

Query OK, 1 row affected (0.03 sec)

mysql> select * from files;
+----+-----+-----+-----+
| id | username | filename | created |
+----+-----+-----+-----+
| 22 | user | UserProposal.doc | 2016-02-14 22:19:22 |
| 32 | user | UserPlanning.doc | 2016-02-14 22:19:22 |
| 42 | user | UserExecution.doc | 2016-02-14 22:19:23 |
| 52 | editor | EditorProposal.doc | 2016-02-14 22:19:23 |
| 62 | editor | EditorPlanning.doc | 2016-02-14 22:19:23 |
| 72 | editor | EditorLookup.doc | 2016-02-14 22:19:23 |
+----+-----+-----+-----+
6 rows in set (0.04 sec)

mysql> _

```

To verify that database is populated, execute select * from files; command.

Create manifest.yml

A best practice in Cloud Foundry is to use what's called a *manifest.yml* file when deploying your applications.

Create manifest.yml file in DocumentManager project in your Eclipse workspace, referred to as <ECLIPSE_WORKSPACE>

```
applications:
- name: documentmanager
  path: C:\dev\ApplicationServers\wlp-8.5.5.8\wlp\usr\servers\defaultServer
  buildpack: liberty-for-java
  memory: 512M
  host: documentmanagersandhya
  domain: mybluemix.net
  services:
  - documentmanager-db
```

Figure 28: manifest.yml

1. path parameter should point to your local server directory.
2. host name should be unique to deploy the app to Public Bluemix.
3. services should contain the name of your ClearDB service name.

We will use Auto wiring feature of Liberty buildpack, to bind the application to ClearDB MySQL instance, without requiring an update to the application code.

Go to **<ECLIPSE_WORKSPACE>/DocumentManager** directory where manifest.yml file lives.

Run deployment command as shown below:

cf push documentmanager -p **<SERVER_DIR>**

You can now test the application by invoking the URL:

documentmanagersandhya.mybluemix.net/document

Exercise 2:

Integrating Cloud Applications with On-Premise Resources

This exercise will show how to connect a Java Enterprise Edition (JEE) application running in Bluemix to a relational database running outside of Bluemix. Typically these would be an enterprise application hosted in Bluemix connecting to an enterprise database of record hosted in a private data center (a.k.a. on-premise).

Introduction

The solution we'll build in this tutorial is intentionally quite simple. The solution connects an application running in Bluemix to an enterprise database of record running outside of Bluemix. The focus is on how to configure and use Secure Gateway, not on the sophistication of the application or the database.

The reader can perform this tutorial using their computer and a Bluemix account. Rather than require that the reader have access to an existing data center hosting a production enterprise database of record, this tutorial simulates one using a MySQL database running in a virtual machine hosted by Bluemix. This enables the user to experience installing Secure Gateway without requiring access to a data center.

The techniques shown in this tutorial using this architecture can easily be applied to true production Java applications and enterprise databases running in private data centers. The

existing Java application is migrated to Bluemix for the operational efficiencies of cloud computing. Meanwhile, the enterprise database remains in the data center so that other existing applications can continue to access it locally. Secure Gateway is installed in Bluemix and in the data center and configured to connect the Java application to the enterprise database. This tutorial shows how to perform these steps.

Figure 29 shows the architecture of the solution that we will build. The sample application is a Java program that enables the user to manipulate the data in an SQL relational database, which is typical functionality of many existing Java applications. This program runs in Bluemix in a Liberty for Java runtime. The database stores the Java application's data. It is a SQL database, as is typically used by many existing Java applications. For this tutorial, the data center is simulated by a virtual machine and the SQL database is a MySQL server. The Secure Gateway service in Bluemix will be used to connect the Java runtime to the MySQL database.

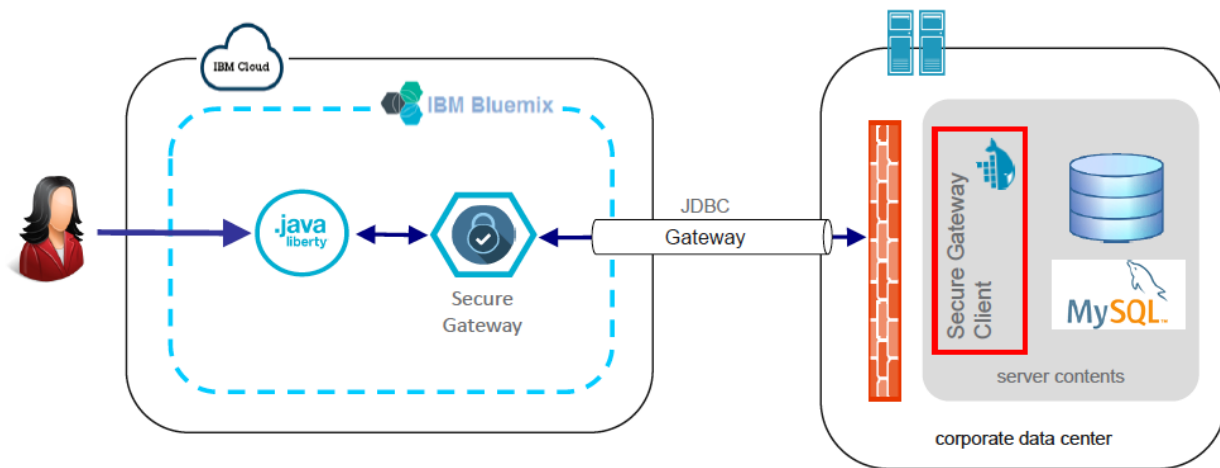


Figure 29: Architecture of the solution

Let's get started with the tutorial.

Prerequisites

To perform this exercise, you'll need an account in the Bluemix public cloud. If you do not already have a Bluemix account, see the instructions in the lab's "" section for how to create a free trial account.

In this exercise, you will use the following development tools:

- Cloud Foundry CLI — Used to deploy the sample application to Bluemix.
- Eclipse — Used to load the source code for the sample application and edit the server configuration.
- Liberty — Used to test the sample application locally and to package the server to deploy it to Bluemix.
- Git Bash or PuTTY — Used in Windows to generate a key pair and SSH into the VM hosted in Bluemix.

These tools have already been installed in the client VM used to perform this lab. For more information, see “” in the appendices on page 65.

Overview

This exercise consists of the following steps:

- — Create a simulation of a data center hosting an enterprise database of record.
- — Get the application running locally connecting directly to the remote database.
- — Get the application running locally connecting to the remote database via a Secure Gateway.
- — Get the application running in Bluemix (connecting to the remote database via the Secure Gateway).

This is the basic approach you can use to get an application working locally, add Secure Gateway, and get it working in Bluemix.

Step 1: Set up environment

Before we can show how to use Secure Gateway, we need to set up a sample application and sample database. This section leads the reader through setting up these required assets:

- *Virtual machine* — The tutorial uses a virtual machine running MySQL to simulate an enterprise database hosted in a data center. The tutorial explains how to create the VM, install Docker, and install MySQL initialized with the database needed for the Java application.
- *Create sample app* — The tutorial needs a sample application to connect to the database. We'll use one already included in Bluemix, from the Java DB Web Starter boilerplate.
- *Import sample app* — Once we've downloaded the starter code, we'll load the project into Eclipse.
- *Deploy sample app* — We'll deploy the Eclipse project to our local Liberty server to see the application from Bluemix run locally.

Create simulated data center

To demonstrate Secure Gateway, we'll need a system of record running in a private data center. One of those is difficult to download and install! So for the purposes of these exercises, we will simulate one using a MySQL database running in a virtual machine. We'll create a virtual machine, install Docker, install MySQL running in a Docker container, and initialize the database with some sample data the Java application will need.

Before we get started with that, we need to log into Bluemix and create a space to work in. As part of doing so, we'll need to switch regions. The default Bluemix region is the US South region. However, the IBM Virtual Machines BETA environment is available only in the United Kingdom region. So you need to log into Bluemix switch to the UK region, and create a space.

In a web browser:

1. Log into Bluemix.

If you do not have a Bluemix account, see “” in the prerequisites on page 6.

2. In the Bluemix console, in the upper right corner, click on the Account and Support button. See Figure 30.

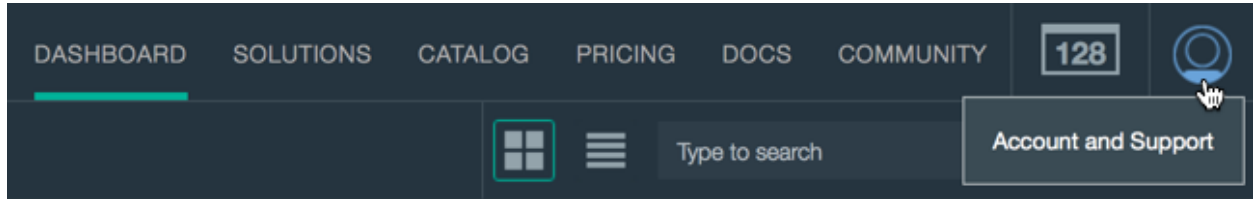


Figure 30: Bluemix account and support

3. Use the Account and Support menu to change your active region to the United Kingdom region. See Figure 31.

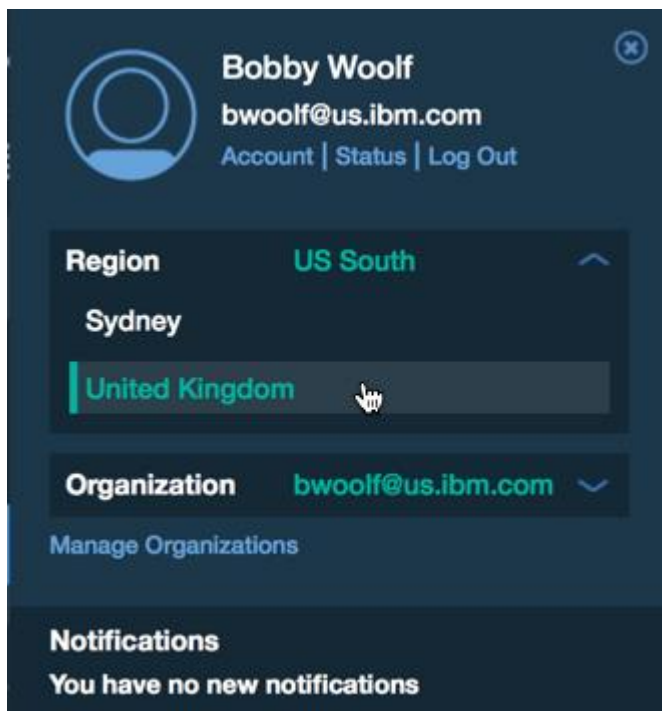


Figure 31: Change Bluemix region

4. Create a new space. Name it **hybrid-cloud**.

You are now logged into Bluemix with your console set to the United Kingdom region and your hybrid-cloud space.

All of your Bluemix artifacts—a virtual machine, a Java for Liberty runtime, and a couple of services instances—will be created in your space. When you have completed this lab and want

to delete these artifacts to make their resources available for other uses, you can simply delete this space.

Create VM

We'll create a virtual machine to simulate a private data center. That VM can run anywhere as long as it has a public IP address (in any cloud provider, on your local computer, etc.). For this exercise, we'll use the Virtual Machines capability in Bluemix.

In a terminal window or tool:

1. Create an SSH keypair, as documented in "Creating web applications: Creating a virtual machine: Configuring an SSH security key in a VM: Creating an SSH security key to access a VM" in the Bluemix documentation:

https://www.ng.bluemix.net/docs/virtualmachines/vm_index.html#vm_create_ssh_key

Specifically:

- We'll call ours **todo**.
 - Private key file: **todo.key**
 - Public key file: **todo.key.pub**
- In Unix/Linux: Run `ssh-keygen -t rsa -f todo.key`
- In Windows:
 - Git Bash: To run Linux commands on Windows, use Git Bash.
 - PuTTY: Use PuTTYgen to generate SSH-1 (RSA) keys.
 - These instructions use Unix/Linux commands.

In the Bluemix dashboard:

2. Create a VM on Bluemix, as documented in "Creating web applications: Creating a virtual machine: Creating a VM in a public cloud" in the Bluemix documentation:

https://www.ng.bluemix.net/docs/virtualmachines/vm_index.html#vm_create_public_cloud

Specifically:

- As of this writing, VMs are only supported in the United Kingdom region, so ensure your Bluemix console is switched to that region.
- Select **Run Virtual Machines** to go to the Create a Virtual Machine page.
- Any Linux image will work. It should be the latest release and support Docker. Docker is pretty easy to install on either Ubuntu 14.04 or Debian 8.0.
- To create the VM, use the settings shown in Table 1. For the settings that have default values, use those default values.
- When you name the VM group "AAA_To_Do," change "AAA" to your initials to make the name unique.
- To specify the security key, select **Add Key** to import your **todo** key. (In Unix, run **cat <public key file>** to display the key so you can copy and paste it into Bluemix.)

Table 1: Virtual machine creation settings

Property	Value	Default
VM Cloud	IBM Cloud Public	default
Initial instances	1	default
Assign public IP addresses	Select (yes)	
VM image	Ubuntu 14.04 or Debian 8.0	default
VM group name	AAA_To_Do	
VM size	m1.small	default
Security Key	todo	
Network	private	default

- Once the VM is created, make note of its public IP address, whose form is 129.xxx.xxx.xxx. (The other IP address, 192.168.xxx.xxx, is private.) We'll refer to this public IP address as *<virtual machine's IP address>*.
- Log into the virtual machine. The image has a user predefined for logging in remotely; it's ibmcloud. Use the authentication key specified when creating the VM.

In Linux: **ssh -i todo.key ibmcloud@<virtual machine's IP address>**

In Windows: Use Git Bash to run the Linux commands.

You now have a running Linux VM and you can log into it.

Install Docker

As we'll see later, Secure Gateway includes a client that must be installed in the data center, and the simplest client implementation to install is the one that IBM has packaged as a Docker container (known as the Docker client for Secure Gateway). Therefore, our VM needs the Docker runtime installed so that we can later install the Secure Gateway client. To use the Docker client in your data center, you will need to install Docker on a host in your data center, the host that will run the Secure Gateway client.

Later, we will also install a MySQL database. To simplify that installation, we'll use MySQL that's already installed in a Docker container. So another reason we need our VM to run the Docker runtime is so that it can run the MySQL container.

For directions for installing Docker:

- “Install Docker Engine” explains how to install Docker on various platforms:
- Bluemix also documents installing the Docker runtime. See “Services: Secure Gateway: Docker” in the Bluemix documentation:

Log into your VM using SSH, as described above, and perform the following:

1. Follow the directions for installing Docker, both the Prerequisites section as well as the Install Docker section.

Note: While performing the prerequisites in Debian, you may get the following error:
E: The method driver /usr/lib/apt/methods/https could not be found.
N: Is the package apt-transport-https installed?
To fix this problem, run this command:
\$ sudo apt-get install apt-transport-https

2. When Docker is installed, run this command:

```
$ sudo docker run hello-world
```

When hello-world runs correctly, part of the output should say:

```
Hello from Docker.
```

This message shows that your installation appears to be working correctly.

When you can run hello world successfully, your VM has the Docker runtime installed and running correctly.

Install and config MySQL

To simulate an enterprise database of record, we'll use a MySQL database with a small, simple data set. To initialize that database, we'll need a schema file and a data file.

Create the database files

Still logged into your VM using SSH:

1. Create the directory for the database initialization files:
\$ mkdir ~/liberty-sql
\$ cd ~/liberty-sql
2. Using your favorite Linux text editor (such as nano or vi), create the file **todo-schema.sql** and insert the contents shown in Figure 32.

```
DROP SCHEMA IF EXISTS todo;  
CREATE SCHEMA todo;  
USE todo;  
  
CREATE TABLE `TODOLIST` (  
  `L_ID` INT(8) DEFAULT NULL,  
  `C_NAME` VARCHAR(254) DEFAULT NULL  
) ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

Figure 32: Contents of todo-schema.sql

3. Create the file **todo-data.sql** and insert the contents shown in Figure 33.

```
USE todo;
```

```
INSERT INTO `TODOLIST` (`L_ID`,`C_NAME`) VALUES (1, "sample entry #1");
INSERT INTO `TODOLIST` (`L_ID`,`C_NAME`) VALUES (2, "sample entry #2");
INSERT INTO `TODOLIST` (`L_ID`,`C_NAME`) VALUES (3, "sample entry #3");
```

Figure 33: Contents of todo-data.sql

4. Confirm you've got the files in the right directory. It should look like this:

```
ibmcloud@aaa-to-do-12345678:~$ ls -l /home/ibmcloud/liberty-sql/
-rw-r--r-- 1 ibmcloud ibmcloud 227 Jan 1 12:00 todo-data.sql
-rw-r--r-- 1 ibmcloud ibmcloud 190 Jan 1 12:00 todo-schema.sql
```

Note: You can also **cat** each file to make sure its contents look correct.

You now have the schema and data file that will be needed to initialize the database when creating its Docker container.

Install the database

Still logged into your VM using SSH:

1. Create the MySQL container instance and load the sample data from the two initialization files with the command below:

```
$ sudo docker run -d --name mysql-tutum -p 3306:3306 -v /home/ibmcloud:/home/ibmcloud -e
MYSQL_PASS=passwd -e STARTUP_SQL="/home/ibmcloud/liberty-sql/todo-schema.sql
/home/ibmcloud/liberty-sql/todo-data.sql" tutum/mysql
```

Where:

- `-d` runs the container in the background, not interactively
 - `mysql-tutum` is the name to give the container created from the image
 - `3306:3306` forwards the MySQL port to make it accessible from the host OS's IP address
 - `/home/ibmcloud:/home/ibmcloud` binds the directory to make the directory in the host OS available within the container
 - `MYSQL_PASS` sets the password of the database's main user, in this example to `passwd`
 - `STARTUP_SQL` tells the container to run the SQL files in the order specified via the space-separated list
 - `tutum/mysql` is the name of the Docker image to create the container from
2. Verify that the MySQL container is running with this command:

```
$ sudo docker ps
```

CONTAINER ID	IMAGE	STATUS	NAMES
dd1234567890	tutum/mysql	Up 28 seconds	mysql-tutum

You now have a running Docker container named `mysql-tutum`. That container has a MySQL database server running in it, bound to port 3306. The database server contains a database named `todo` that contains a table named `TODOLIST` that contains the sample data for a set of items in a To Do list.

Create sample app

This tutorial needs a sample application, a Java app that connects to a SQL database. Rather than create one from scratch, we'll use one that's built into Bluemix, created by the Java DB Web Starter boilerplate.

In the Bluemix dashboard:

1. Select your space named **hybrid-cloud**.
2. Create a new instance of the **Java DB Web Starter** boilerplate.
 - a. Select **Catalog**.
 - b. In the Boilerplates section of the catalog, select **Java DB Web Starter**.
 - c. On the boilerplate's Create page, specify the name as **AAA-java-sql**, where AAA is your initials or some other ID to make the application's name unique.
 - d. Leave the other settings with their default values and press **Create**.
3. Once the boilerplate's application runtime has started, run the application.
 - a. Select the link shown next to "Your app is running," such as .
 - b. A new browser window opens displaying the Java DB Web Starter home page. See Figure 34. Experiment with adding, changing, and removing items in the To Do list. These items are stored in the SQL database.



Figure 34: To Do application

4. Back in the Dashboard, it should still display the page for your application with Start Coding selected.
 - a. Select **Download Starter Code** to download **AAA-java-sql.zip** to your local computer.
 - b. If you needed to install the CF CLI, you could do so here: Select **Download CF Command Line Interface** to download and install the CF CLI.

You've now seen the Java DB Web Starter app run in Bluemix and have downloaded its code. For good measure, we've also installed the Cloud Foundry CLI, which later we'll use to upload our modified application back into Bluemix.

Import sample app

The starter code from Bluemix is conveniently packaged as an Eclipse project. Import that project into Eclipse so that you can review the sample app's code.

In Eclipse:

1. Import the **Java DB Web Starter** project, as documented in "Importing existing projects" in *Eclipse documentation - Current Release*:

Specifically:

- Specify Select archive file.
- Select the starter code file you downloaded in "," AAA-java-sql.zip.

This creates the project **JavaDBApp**.

Deploy sample app to Liberty server

Not only can we edit the sample app's code in Eclipse, we can also use Eclipse to deploy the sample app to our Liberty server and run the sample app.

In Eclipse:

1. Add the sample app's project to the Liberty server, as documented in "Adding projects to a server" in *Eclipse documentation - Current Release*:

Specifically:

- The project to add is **JavaDBApp**.
 - The server to add it to is **wlp-8.5.5.8 defaultServer at localhost**.
2. The Servers view now shows the project associated with the server.

Documentation: For details, see "Servers view" in *Eclipse documentation - Current Release*:

At this point, we could start the server and start the app in the server, then open the app's GUI and test it. There's not much point in doing that yet, however, because the app isn't connected to a database, so the section of the web page that's supposed to display the To Do list in the database instead will simply say "Error."

Step 2: Connect directly to database

Now we'll set up the local Liberty server to connect to MySQL and test the app using that connection.

Add the MySQL driver jar

First: Download MySQL's JDBC driver, which they call Connector/J.

1. Go to the MySQL Download Connector/J page.
2. Download the archive for the latest version of the platform independent driver (not the Microsoft Windows driver, since we will upload this driver to Bluemix), unpack it, and get the jar file. In this example, we downloaded v5.1.35, so the driver jar is:

mysql-connector-java-5.1.35-bin.jar

Second: In the directory structure of the Liberty server, add the MySQL driver jar, mysql-connector-java-5.1.35-bin.jar, to the server's lib directory.

1. Go to the directory where your Liberty server is installed on disk, `<LIBERTY_ROOT>`.
2. Go to the subdirectory `wlp/usr/servers/defaultServer..`
3. Go to the subdirectory `lib`. (If `defaultServer/lib` doesn't already exist, create it.)
4. Copy or move the MySQL driver jar, `mysql-connector-java-5.1.35-bin.jar`, into `defaultServer/lib`.

The server's libraries now contain the database driver.

Configure server for MySQL

Modify the server configuration of the Liberty server, **WebSphere Application Server Liberty Profile**, to add a data source for the MySQL database and to set the app's context root.

First, let's review the application dependency that requires the data source.

In Eclipse:

1. In the Enterprise Explorer, navigate to your project, `JavaDBApp`.
 - In that project, the `Java Resources` folder contains the application source code.
2. Navigate to `persistence.xml` and open the file.
 - It's typically in the path `src/main/resources/META-INF/` or `src/META-INF/`.
3. Notice the line defining a JTA data source, shown in Figure 35.

```
<jta-data-source>java:comp/env/jdbc/mydbdatasource</jta-data-source>
```

Figure 35: Defining a JTA data source

This line makes an important declaration:

- It binds the application's persistence unit, openjpa-todo, to a data source whose JNDI name is jdbc/mydbdatasource.

Therefore the server needs to define a data source registered in JNDI as jdbc/mydbdatasource.

Second, let's add that data source to the server configuration.

In Eclipse:

1. In the Servers view, open the **Server Configuration [server.xml]** file associated with the server.
2. In the Server Configuration view, use either the Designer tab or the Source tab to add the Data Source and Shared Library shown in Figure 36.
 - Notice that the library's fileset is configured to locate the MySQL driver jar.

```
<dataSource jndiName="jdbc/mydbdatasource">
  <jdbcDriver libraryRef="mysql-connector"/>
  <properties URL="jdbc:mysql://<hostname>:<port>/todo?relaxAutoCommit=true"
    user="<user>" password="<password>"/>
</dataSource>
<library id="mysql-connector" name="MySQL Connector" description="MySQL JDBC
Driver">
  <fileset id="mysql-connector-jar" dir="${server.config.dir}/lib" includes="mysql-
connector-java-*.jar"/>
</library>
```

Figure 36: Data Source and Shared Library configuration

In the URL property's value, substitute these variables:

- *<hostname>* — the IP address of the virtual machine hosting the MySQL database: *<virtual machine's IP address>*
 - *<port>* — set when we created the MySQL container: 3306.
 - *<user>* — the default user in the MySQL container: admin
 - *<password>* — set when we created the MySQL container: passw0rd
3. To make the app accessible via the root directory (and not a subdirectory like liberty-IRDS), add the Web Application configuration shown in Figure 37.

```
<webApplication id="liberty-IRDS" name="liberty-IRDS" location="liberty-IRDS.war"
context-root="/">
```

Figure 37: Web application configuration

4. As explained in the Console view, the URL for accessing the app is .

The server is now configured with a data source for accessing the database, and the application's context root is set.

Test app using direct connection

Now let's run the app and confirm that it connects to MySQL correctly.

In Eclipse:

1. In the Servers view, confirm the server and app are started and synchronized. If not, start the server and confirm the app starts as well.

The items that should show as started and synchronized are:

- wlp-8.5.5.8 defaultServer at localhost
 - JavaDBApp(liberty-IRDS)
2. The app runs a web GUI. To open the GUI, in Eclipse or in an external web browser, go to .
 3. The app's **Java DB Web Starter** web page opens, as shown earlier in Figure 34 on page 52. The area where the To Do list will be displayed says "Please wait while the database is being initialized ..." while the database is accessed.
 4. If the app connects to the database successfully, the To Do list then fills in. The default items are sample entry #1, #2, and #3.
 - If the app cannot connect and load the data successfully, the To Do area displays "Error."

The app is running locally using the MySQL database in the virtual machine.

Step 3: Connect via Secure Gateway

Now let's configure a Secure Gateway that connects to the MySQL database, and configure the app to use that.

Configure a gateway and destination

We will create an instance of the Secure Gateway service, then use it to configure a gateway and configure a destination in that gateway. The destination will be the MySQL database running in the virtual machine.

For details:

- See "Services: Secure Gateway: Creating a Secure Gateway by using the Bluemix UI" in the Bluemix documentation:

https://www.ng.bluemix.net/docs/services/SecureGateway/sg_022.html#sg_009

In the Bluemix dashboard:

1. Select your space, **hybrid-cloud**.
2. In the main view for the space, select **Use Services or APIs**.

3. In the Service Catalog, select **Secure Gateway**.
4. In the creation page for Secure Gateway, keep the default settings and select **Create**.
5. In the Secure Gateway page, select **Add Gateway**.
6. In the Add Gateway page, set the name to **My Data Center**. Deselect the options to use tokens. See Figure 38.

☒ *Name It
☐ Connect It
☐ Add Destinations

What would you like to name this new gateway?

My Data Center

☐ Require clients to provide the security token on connection. ⓘ

☐ Token Expiration:

90

 days ⓘ

What would you like to do next?

CONNECT IT

ADD DESTINATIONS

I'M DONE

Figure 38: Add gateway

7. Select Add Destinations.
8. In the Add Destinations page, add a destination with the settings shown in Table 2. See Figure 39.
 - Fill in the values and then press the plus sign (+) on the right.
 - The IP address and port are the same settings you specified in the data source's URL in the Liberty server's configuration.

Table 2: Secure Gateway destination settings

<i>Property</i>	<i>Value</i>
Destination name	To Do Database
Hostname or IP Address	<virtual machine's IP address>
Port	3306
Transport	TCP

✔ *Name It
○ Connect It
● Add Destinations

Let's Add some destinations to your gateway...

Create Destinations

To Do Database

159.122.111.222

3306

TCP

+

▶ Advanced

What would you like to do next?

CONNECT IT

I'M DONE

Figure 39: Add destination

9. A destination named To Do Database is now listed. Optional: Select destination info (the button looks like the letter i with a circle around it) to display the destination's configuration, as shown in Table 3.

Table 3: Destination info

Property	Value
Name	To Do Database
Destination ID	aaaA0AAAA00_vfY
Cloud Host : Port	cap-sg-prd-1.integration.ibmcloud.com:15000
Destination Host : Port	<virtual machine's IP address>:3306
Created by	John Doe at 7/1/2015, 12:05:00 PM
Last modified by	John Doe at 7/1/2015, 12:05:01 PM

The gateway is a proxy that maps the cloud host and port to the destination host and port:

- The destination's host and port are the values you specified when you created the destination. This is the target the gateway will connect to, the MySQL database.
- The cloud host and port are the values an application will use to access the destination remotely.

You have now configured a gateway and destination for accessing the database.

Configure the gateway client

The gateway needs a client deployed in the data center hosting the database. The gateway client needs network access to both the database running in the data center and to the gateway running in Bluemix. Typically, this means that the gateway client should be installed on a host that is connected to the same network segment (that is, subnet or VLAN) as the host of the database, and that is connected to the Internet (or some other network connection to Bluemix). The gateway client can run on a host natively, in a Docker container, or in IBM DataPower. In this example, we'll use the Docker container. The virtual machine simulating our data center already has the Docker runtime installed (we installed it in "Install Docker" on page 5), so we just need to deploy the Docker container to that Docker runtime.

For details:

- We've already installed the Docker runtime in this tutorial (see "" on page 49). For additional details on installing the Docker runtime, see "Services: Secure Gateway: Docker" in the Bluemix documentation:

https://www.ng.bluemix.net/docs/services/SecureGateway/sg_021.html#sg_003

- Installing the Docker container for the Secure Gateway client is documented as part of setting up Secure Gateway. See "Services: Secure Gateway: Creating a Secure Gateway by using the Bluemix UI" in the Bluemix documentation:

https://www.ng.bluemix.net/docs/services/SecureGateway/sg_022.html#sg_009

Starting the gateway client

First, get the Docker connect command for the gateway client. In the Bluemix dashboard:

1. Navigate to the Secure Gateway Dashboard for your space's Secure Gateway service instance.
2. One gateway is listed, named My Data Center. Optional: Select gateway info (the button looks like the letter i with a circle around it) to display the gateway's configuration, as shown in Table 4.

Table 4: Gateway info

<i>Property</i>	<i>Value</i>
Name	My Data Center
Gateway ID	aaaA0AAAA00_prod_ng
Created by	John Doe at 7/1/2015, 12:00:00 PM
Last modified by	John Doe at 7/1/2015, 12:00:01 PM

3. Click on the gateway's label to go to the Gateway page.

4. In the Gateway page, notice that one destination is listed, To Do Database. Select **Connect Gateway**. The page displays: How would you like to connect this new gateway? See Figure 40.

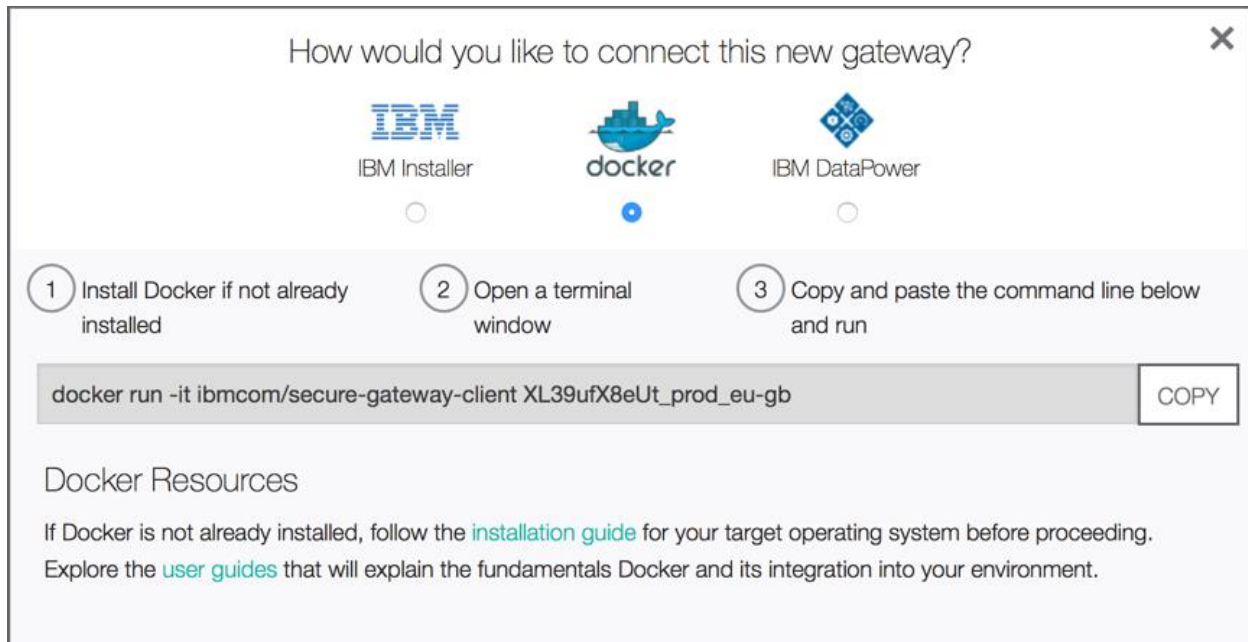


Figure 40: Connect gateway

5. With Docker selected, the page displays the command to install and run the gateway's Docker client, as shown in Figure 41.
 - o Notice that the parameter is the gateway's ID, shown in Table 4. This tells Bluemix what gateway the client needs to connect to.
 - o Make a note of this Docker command. You'll need to run it once you're logged into the VM.

```
docker run -it ibmcom/secure-gateway-client aaaA0AAAA00_prod_ng
```

Figure 41: Command to run a gateway's Docker client

Second, install and run the gateway client in the virtual machine.

1. Log into the virtual machine. Use the authentication key specified when creating the VM.
From a Unix/Linux or Git Bash shell:
`$ ssh -i todo.key ibmcloud@<virtual machine's IP address>`
2. Verify that the Docker runtime is running correctly.
`$ sudo docker run hello-world`
3. Run the command Bluemix gave us to install and run the gateway Docker client.

```
$ sudo docker run -it ibmcom/secure-gateway-client aaaA0AAAA00_prod_ng
```

The gateway client is now running. It says it's connected, and the Gateway page says the gateway is connected. The shell used to run the client is now attached to the container and running the gateway client's shell, which displays when connections are open and closed.

Note: For more details about how to use the CLI, see "" in the appendices.

Configure the gateway client

The gateway client is running, but by default, it blocks all traffic. If a client tries to use the gateway to connect to any of the gateway's destinations, it gets a "Communications link failure" error.

The gateway client is configured with an access control list (ACL) that specifies which hostname-port combinations to allow and which to deny. By default in a newly created gateway client, the ACL is empty, which means that all hostname-port combinations are blocked. This is a security feature; even if an administrator of the gateway in Bluemix decides to start accessing a new destination, the administrator of the gateway client has to agree and add that destination's hostname-port combination to the client's ACL.

Configure the gateway client's ACL to allow the gateway to connect to the destination:

1. When you started the Docker container for the gateway client, the command line is attached to the client's shell. Press <enter> to get the prompt for the client's CLI.

```
<enter>
cli>
```
2. Show the current ACL configuration. It shows that all ports are blocked.

```
cli> show acl
[2016-01-01 12:00:00.000] [INFO] (Client PID 1) There are no Access Control List
entries, the ACL Deny All flag is set to: true
```
3. Add the destination's hostname and port to the ACL. This is the same hostname and port you used to configure the gateway's destination. See "" on page 56.

```
cli> acl allow 159.122.240.100:3306
cli> show acl
```

```
-----
-- Secure Gateway Client Access Control List --
hostname                port    value
<virtual machine's IP address> 3306    Allow
-----
```

The ACL in the gateway client is now configured to allow the gateway to connect to the destination at <virtual machine's IP address>:3306.

Configure application server for Secure Gateway

Now that we have our Secure Gateway service instance configured with a gateway and destination to connect to our MySQL database, we need to modify our app to use that gateway

destination. Recall that the destination's configuration, shown in Table 3 on page 58, includes the value for Cloud Host : Port, which in this example is:

cap-sg-prd-1.integration.ibmcloud.com:15000

This is the URL the app will use to access the database via the gateway destination.

In Eclipse—as we did in “” on page 54—modify the server configuration of the Liberty server to update the data source to use the gateway destination's URL.

1. In the Servers view, open the **Server Configuration [server.xml]** file associated with the server.
2. In the Server Configuration view, use either the Designer tab or the Source tab to edit the URL in the data source, this time with these values:
 - `<hostname>` — the gateway destination's Cloud Host: cap-sg-prd-1.integration.ibmcloud.com.
 - `<port>` — the gateway destination's Cloud Port: 15000.
 - `<user>` — unchanged: admin
 - `<password>` — unchanged: passw0rd

As the Console shows, the Liberty server automatically updates with the new configuration.

Test app using gateway connection

Now let's run the local app and confirm that it correctly connects to MySQL via the gateway.

1. To run the app, go to .
2. The app's **Java DB Web Starter** window opens and the To Do list data is displayed.
3. In the shell logged into the virtual machine, the logging in the gateway client's CLI shows that a new connection was opened.

This confirms that the local app is able to connect to the database via the gateway.

Step 4: Deploy to Bluemix

Now that we have the local app working correctly using the gateway, let's deploy it to Bluemix and confirm that it works correctly from there as well.

Deploy server and app to Bluemix

We need to not only deploy the app (its JavaDBApp.war file), but also its server configuration (the server's server.xml file and the database driver, mysql-connector-java-5.1.35-bin.jar). How can the server configuration be deployed along with the app?

Packaging a Liberty server

To deploy not just an app but its server configuration as well, we package the server with the app and deploy the package. There are two procedures for doing this:

- Command line

- Eclipse tools

Command line

This procedure is documented in “Creating web applications: Creating apps with Liberty for Java: Options for pushing Liberty applications” in the Bluemix documentation, under “Packaged Server”:

<https://www.ng.bluemix.net/docs/starters/liberty/index.html#optionsforpushinglibertyapplications>

The procedure uses a Liberty server command to package the server into an archive. That procedure is documented in “Packaging a Liberty profile server from the command line” in the *WebSphere Application Server (Distributed and IBM i operating systems), Version 8.5.5* documentation:

That procedure also uses the CF CLI. Deploying applications using the CF CLI is documented in “Managing applications: Deploying applications by using the cf command” in the Bluemix documentation:

https://www.ng.bluemix.net/docs/manageapps/deployingapps.html#dep_apps

Eclipse tools

The IBM Eclipse Tools for Bluemix can also be used to deploy a packaged server. This procedure is documented in “CLI and Dev Tools: Deploying apps with IBM Eclipse Tools for Bluemix: Packaged server support” in the Bluemix documentation:

<https://www.ng.bluemix.net/docs/manageapps/eclipsetools/eclipsetools.html#packagedserversupport>

Perform the package and push

Here, we’ll show how to package and push the server using the command line. This procedure shows more clearly the steps that the Eclipse tools perform for you.

In the directory structure of the Liberty server, package the server and then push it to Bluemix.

1. Go to the directory where your Liberty server is installed on disk, `<LIBERTY_ROOT>`.
2. Before you package the server, if the server is running, stop it.

```
$ wlp/bin/server stop defaultServer
```

3. Run the package command to package the server.

```
$ wlp/bin/server package defaultServer --include=usr
```

4. If you haven’t already, log in the CF CLI to your Bluemix account, org, and space. Run the following command:

```
$ cf login -a <API_URL> -u <username> -o <org> -s <space>
```

Where:

- `<API_URL>` — the URL of the Cloud Foundry provider, which is Bluemix:
 - US South:
 - United Kingdom:
 - `<username>` — your email address.
 - `<org>` — your email address (assuming your org in Cloud Foundry is named after your user)
 - `<space>` — the space in Cloud Foundry you want your app deployed to: hybrid-cloud
5. Then use the CF CLI to push the packaged server to Bluemix.

```
$ cf push <appname> -p wlp/usr/servers/defaultServer/defaultServer.zip
```

Where:

- `<appname>` — the name you want your app to have in Bluemix. Needs to be unique so that its route will also be unique.

It will typically take the app 1-2 minutes to upload and several minutes to start. The deployment has finished when the CF CLI says the app has started and when the Bluemix dashboard says the app is running.

Review the deployed app

Optional: Once the app is running in Bluemix, we can review it to see that our server configuration has been uploaded successfully.

In the Bluemix dashboard:

1. Navigate to the page for your application, `<appname>`, and select **Files**.
2. Navigate to `app/wlp/usr/servers/defaultServer/server.xml` and select the file.

Confirm that the `server.xml` file contains the customizations we made to the server configuration back in “” on page 54 and “” on page **Error! Bookmark not defined.**, such as:

- A data source with the JNDI name `jdbc/mydbdatasource`.
 - That the data source's URL property is in part `cap-sg-prd-1.integration.ibmcloud.com`.
 - That the data source links to a library, `mysql-connector`, with a fileset for matches to `mysql-connector-java-*.jar`.
 - A web application whose context root is the root directory.
3. Navigate to `app/wlp/usr/servers/defaultServer/lib`. Confirm that this directory contains the MySQL driver, such as `mysql-connector-java-5.1.35-bin.jar`.

This confirms that not only was the app deployed to Bluemix, but that its server configuration was also deployed.

Test app in Bluemix

Now let's run the app in Bluemix and confirm that it correctly connects to MySQL via the gateway.

1. To run the app, go to `http://<your_app>.ng.mybluemix.net`.
2. The app's **Java DB Web Starter** window opens and the To Do list data is displayed.
3. In the shell logged into the virtual machine, the gateway client shows that a new connection was opened.

This confirms that the Bluemix app is able to connect to the database via the gateway.

Appendix A Useful knowledge

Here is some additional information that is not required to perform the lab, but that may help you understand better how the lab procedures work. You may be able to apply this additional understanding in your future efforts.

The additional sections are:

-
-

Please see these sections below.

Bluemix development environment

The local environment on the laptop for this lab has several tools installed. If you'd like to reproduce the environment on your own computer, you'll need to install the tools described in the following sections.

Java Development Kit

Download the Java Development Kit (JDK) from Oracle:

-

This lab used JDK 8 v1.8.0_66.

WebSphere Liberty

You have two options for installing Liberty with Eclipse:

1. Download Liberty, install it, and connect to it from Eclipse
 -
2. Download and install Liberty using Eclipse, which creates the connections
 -

The second approach assumes you already have the Eclipse IDE installed. To do that, see the next section, “.”

This lab used the Liberty v8.5.5.8 release.

Eclipse with tools

To download and install Eclipse, go to the Downloads and either run the Eclipse Installer or download the Eclipse Package and install it manually. The Eclipse downloads page is here:

-

This lab used the Eclipse Mars.1 (4.5.1) release.

We also added several tools to Eclipse that are developed by IBM and helpful to using Eclipse with WebSphere products and Bluemix. To install any tool into Eclipse, you have three options:

1. *Eclipse Marketplace dialog* — You can use the Eclipse Marketplace feature in your Eclipse IDE to find and install tools. To open this feature, from the menu bar, select **Help > Eclipse Marketplace**.
2. *Eclipse Marketplace web site* — You can find the page for the tool on the Eclipse Marketplace web site. Each page has an Install button you can drag onto your Eclipse IDE. The web site's home page is:

-

3. *Vendor web site* — IBM has a page on its web site for each of the tools. The page lists several installation options: Download, Install button, and Install New Software dialog.

Here is the list of tools that we added to Eclipse and its page on the IBM web site:

1. WebSphere Developer Tools for Eclipse Mars – for WAS Liberty Profile V8.5.5
 - https://developer.ibm.com/wasdev/downloads/#asset/tools-WebSphere_Developer_Tools_for_Eclipse_Mars
2. IBM Eclipse Tools for Bluemix
 - https://developer.ibm.com/wasdev/downloads/#asset/tools-IBM_Eclipse_Tools_for_Bluemix
3. WebSphere Application Server Migration Toolkit
 - https://developer.ibm.com/wasdev/downloads/#asset/tools-WebSphere_Application_Server_Migration_Toolkit

You now have Eclipse installed with several WebSphere and Bluemix tools.

For the sample JBoss application we use for the Migration exercise, we also installed the JBoss Tools into Eclipse. See the “” section below.

Cloud Foundry CLI

To install the CF CLI, follow the directions in “Installing the cf Command Line Interface”:

-

This lab used CF version v6.14.0.

Git client

Download the Git tool here:

-

This lab used git version 2.6.4.windows.1.

PuTTY

Download PuTTY here:

-

This lab used PuTTY release 0.63.

JBoss

The sample application requires an older version of JBoss. Download older versions of JBoss Application Server here:

-

The sample application requires JBoss v5.1, so download JBoss AS 5.1.0.

JBoss Tools

Download the JBoss Tools for Eclipse Mars 4.5 here:

-

This lab used JBoss Tools 4.3.0.Final.

MySQL

Download MySQL Community Server here:

-

This lab used MySQL Community Server 5.7.10.

PostgreSQL

Download PostgreSQL here:

-

This lab used PostgreSQL version 9.4.

Derby

Download Apache Derby here:

-

This lab used version 10.11.1.1.

Using the Secure Gateway client CLI

Optional: This section is a brief detour to look at the gateway client CLI. There are no tutorial steps in this section.

The command to run the gateway client container also connects the user to the gateway client CLI. The CLI shows logging information, such as when connections from the gateway are open and closed. See Figure 42.

```
[2015-01-01 12:00:00.000] [INFO] Connection #1 is being established to 129.40.111.222:3306
[2015-01-01 12:01:00.000] [INFO] Connection #1 to 129.40.111.222:3306 was closed
[2015-01-01 12:02:00.000] [INFO] Connection #2 is being established to 129.40.111.222:3306
[2015-01-01 12:03:00.000] [INFO] Connection #2 to 129.40.111.222:3306 was closed
```

Figure 42: Gateway client logging information

The CLI also has commands for managing the client. For the list of commands, run the help command:

```
cli> help
```

Documentation: These commands are also documented in “Services: Secure Gateway: Secure Gateway client interactive command-line interface” in the Bluemix documentation: https://www.ng.bluemix.net/docs/services/SecureGateway/sg_022.html#sg_010

When you log out of the virtual machine, the gateway client keeps running so that applications can continue to connect to destinations. How do you know that the client is still running? On the Bluemix end, in the Bluemix dashboard, the gateway view shows whether the gateway is connected.

On the client end, use Docker commands to see what the client container’s status is.

Documentation: Docker commands are documented in “Using the command line” in the Docker documentation:

To see if the gateway client is running, run Docker’s process status command, as shown in Figure 43.

```
$ docker ps
```

CONTAINER ID	IMAGE	STATUS	NAMES
aa000000000a	ibmcom/secure-gateway-client	Up 4 days	gateway_client
a000aa00aaaa	tutum/mysql	Up 2 weeks	mysql-tutum

Figure 43: Docker process status command

This example shows a couple of details of interest:

- The container for the gateway client is the one named gateway_client. We know this because the container gateway_client was created from the image ibmcom/secure-gateway-client.
- The container gateway_client is running--its status says “Up.”

To see all of the Docker containers, whether they're running or stopped, use the `all` option, as shown in Figure 44.

```
$ docker ps -a
```

CONTAINER ID	IMAGE	STATUS	NAMES
aa000000000a	ibmcom/secure-gateway-client	Exited (0) 7 seconds ago	gateway_client
a0a0000a0aa0	hello-world	Exited (0) 2 weeks ago	serene_newton
a000aa00aaaa	tutum/mysql	Up 2 weeks	mysql-tutum

Figure 44: Docker process status `all` command

This example shows a couple of details of interest:

- The container `gateway_client` is stopped--its status says "Exited."
- The `gateway_client` container stopped cleanly--its exit code is 0.

To start an existing container, run Docker's `start` command:

```
$ docker start gateway_client
```

To stop a running container, run Docker's `stop` command:

```
$ docker stop serene_newton
```

Notice that example shows stopping the `hello-world` container; it doesn't show using the `stop` command to stop the gateway client's container. Stopping the gateway client by stopping its container is the equivalent of running `kill -9` on the client process, which stops the client immediately and doesn't allow the client to stop cleanly. The result is a 137 exit code, as shown in Figure 45.

```
$ docker start gateway_client
```

```
gateway_client
```

```
$ docker stop gateway_client
```

```
gateway_client
```

```
$ docker ps -a
```

CONTAINER ID	IMAGE	STATUS	NAMES
aa000000000a	ibmcom/secure-gateway-client	Exited (137) 2 seconds ago	gateway_client
a0a0000a0aa0	hello-world	Exited (0) 2 weeks ago	serene_newton
a000aa00aaaa	tutum/mysql	Up 2 weeks	mysql-tutum

Figure 45: Docker `stop` causes error code 137

To stop a gateway client cleanly, use its CLI. If you've lost the CLI connection to the gateway client, you can connect to the CLI again by running Docker's `attach` command:

```
$ docker attach gateway_client
```

Figure 46 shows how to start the gateway client's container, connect to the client's CLI, and cause the client to shut down cleanly.

```
$ docker start gateway_client
```

```
gateway_client
```

```
$ docker attach gateway_client
```

```
cli> quit
```

```
[2015-01-01 12:00:00.000] [FATAL] About to exit with code: 0
```

Figure 46: Cleaning stopping a gateway client

Now we've seen how to manage the gateway client using Docker commands and the gateway client CLI.