Learning Days Setup Guide

## Deployment

The application is currently deployed on Microsoft Azure hosting.

Virtual Machine DNS Name: ldays.cloudapp.net

Heartbeat URL: <http://ldays.cloudapp.net/ic-war/contentservice/v1/system>

Database Server:

## Things to install:

### Git – SCM client to access GitHub where our source code is located

**Download link:** <http://git-scm.com/download/win>

### Java jdk1.8 – Java development kit

**Download link:** <http://download.oracle.com/otn-pub/java/jdk/8u40-b25/jdk-8u40-windows-x64.exe>

### Tomcat 8 – Apache Tomcat Servlet Container

**Download link:** <http://mirror.sdunix.com/apache/tomcat/tomcat-8/v8.0.20/bin/apache-tomcat-8.0.20.exe>

### Maven (unzip into c:\Program files\Apache Software Foundation\)

**Download link:** <http://apache.mesi.com.ar/maven/maven-3/3.2.5/binaries/apache-maven-3.2.5-bin.zip>

### Intellij Professional 13

**Download link:** <http://download.jetbrains.com/idea/ideaIU-13.1.6.exe>

### Putty – (optional) ssh client for managing the Azure virtual machine.

**Download link:** <http://the.earth.li/~sgtatham/putty/latest/x86/putty-0.64-installer.exe>

For Javascript developing:

### JetBrains Webstorm:

**Download Link**: <https://www.jetbrains.com/webstorm/>

## NPM/Node.js

npm is the package manager for javascript. It comes as part of the node.js msi. You don’t really need node.js if you are using webstorm but you need the package manager to download the javascript libraries.

**Download Link:** <http://nodejs.org/dist/v0.12.1/x64/node-v0.12.1-x64.msi>

## Environment Variables:

After installing all of the above software create the following environment variables.

* M2\_HOME=C:\Program Files\Apache Software Foundation\apache-maven-3.2.5
* JAVA\_HOME=C:\Program Files\Java\jdk1.8.0\_40

## Azure Database Server Configuration:

You will need to add the Microsoft JDBC Driver to Maven in order to use it as a Maven dependency. Instructions are here: <http://claude.betancourt.us/add-microsoft-sql-jdbc-driver-to-maven/>

We’ll also need to get the IP addresses of any machines that will be connecting to the Azure SQL Server so that we can add them to the firewall configuration on the Azure portal.

## GitHub Configuration

You will need to create an account on Github. Go to <https://github.com/> and create a new account. Once you have an account created, follow these instructions to set up an ssh key for your GitHub account. <https://help.github.com/articles/generating-ssh-keys/>

(\*Note: when you installed Git, it installed a shell tool called gitbash. It is what you will use for the ssh instructions)

Once you have created your GitHub account <fill in information on how to find/watch LearningDays repo>

## Intellij Configuration

While there are many Java IDE’s available, my personal favorite is Intellij. It is a professional grade development environment and we have a license server available to us for using it. While it’s not an absolute must to use this tool, it is the only one I will have configuration information on.

|  |  |
| --- | --- |
| **License Activation**  Intellij Ultimate will prompt for license server the first time you start it up. Use the floating license server maintained by the neutron team. |  |
| **Intellij Quick Start**  The source for our project is up on GitHub. We will want to “Checkout” the project from there. This will actually just clone the GitHub repository to your local disk. If your GitHub account is not already configured you will be asked for your GitHub login information. |  |
| **Getting the Code**  Select the LearningDays.git url on my GitHub account and a location for the code on your local machine. Press the “Clone” button to bring the repository down to your local machine. |  |
| **Maven Tool Window**  Since we’re using Maven, we want to use the maven build lifecycles to build and package our artifacts. Intellij has a built in maven integration so just make sure the tool window is open in your IDE. (skip this if it already is there) |  |
| **Maven Lifecycles**  Under the Maven Projects we want to build the root project (ic-content). Each of the different lifecycles has a specific set of steps that they take and each lifecycle going down on the list includes everything in the one before it (eg: "test" encompasses "clean" "validate" and "compile"). For more information on Maven lifecylces go here: <http://maven.apache.org/guides/introduction/introduction-to-the-lifecycle.html>  For our purpose we will normally use either package or install. I tend to use install as it creates the artifacts that are ready for deployment and then my run configuration takes those and deploys them properly into my local tomcat environment. I manually update the Azure instance so there is no need to use the deploy lifecycle.  Double click on the lifecycle you want to build. |  |
| **Run Configurations**  In Intellij you need to create a configuration that will be used for your application to run. To create a run configuration it is under the Run menu. |  |
| **Tomcat Run Configuration**  Since our application is a servlet we’ll need to set up a Tomcat run configuration. The little green plus sign at the top left is how you add a new configuration, select “Tomcat Server” and then “Local” |  |
| **Configure Tomcat Local Server**  You’ll need to configure the Application server. Press the Configure button next to the Application Server drop down and navigate to your tomcat installation (default is at c:\program files\apache software foundation\tomcat 8\)  Set your browser start page to be the system endpoint for our application.  Everything else should just use the defaults. |  |
| **Configure Tomcat Local Deployment**  Check to make sure that the deploy at the server startup is set to deploy the ic-war:war artifact. This is the .war file that is generated by the maven build. If it is not there, press the green plus sign to the right of the control to add it.  Make sure the application context is set to “/ic”. By default it is set to / and tomcat doesn’t like things deployed at / so your application will not show up. |  |

## Code Description:

#### Package ic-application:

The application package is just the basics needed to get a servlet instance loaded into the tomcat container. It also contains the REST API (WebService.java) and the request/response objects for talking to the REST API.

#### Package ic-storage:

The storage package contains the java classes needed to talk to the persistent storage engine. Right now there are two different classes for supporting different storage back ends – Azure SQLServer and Berkeleydb.

#### Package ic-war:

The war package is just there for creating the war file. It is configuration only and has no source code.

#### Package ic-services

Services package is the application logic. Manages distribution of calls to satisfy workload level transactions.