

BUILD AND DEPLOY SOA PROJECTS FROM DEVELOPER CLOUD SERVICE TO SOA CLOUD SERVICE

Ashwini Sharma

1 CONTENTS

1.	Introduction	2
2.	Prerequisites	2
3.	Patch the SOA Server Installation	3
4.	Use Oracle Developer Cloud Service to Create a New Project	3
5.	Clone the Project.....	6
6.	Populate the Local Maven Repository	8
7.	Populate the Oracle Developer Cloud Service Maven Repository	9
8.	Maven Deploy to SOA Servers on the Local Machine	11
9.	Maven Deploy to Oracle SOA Cloud Service From Local Machines.....	12
10.	Build Your Project on Oracle Developer Cloud Service and Deploy to Oracle SOA Cloud Service .	13
11.	Issues with Certificates	18

1. INTRODUCTION

In the scenarios below, this document describes how to build and deploy SOA composite projects from a local Maven repository on your network to Oracle SOA Cloud Service and from Oracle Developer Cloud Service to Oracle SOA Cloud Service.

2 PREREQUISITES

Install Git, JDK, and Maven, and set up the environment on a local Linux box within your company network.

1. Download Git from:
<https://git-scm.com/>
2. Download JDK 1.8 or later from:
<http://download.oracle.com/otn-pub/java/jdk/8u151tub12/e758a0de34e24606bca991d704f6dcbf/jdk-8u151-linux-x64.tar.gz>)
3. Download Maven version 3 or later from:
<http://www-us.apache.org/dist/maven/maven-3/3.5.0/binaries/apache-maven-3.5.0-bin.tar.gz>)
4. Download and extract the above files in a directory (for example, /tmp/installed/)
5. Export the following variables:
 - a. `JAVA_HOME=/tmp/installed/jdk1.8.0_151`
(Replace the path with the directory where you extracted the JDK)
 - b. `MAVEN_HOME=/tmp/installed/apache-maven-3.5.0`
(Replace the path with the directory where you extracted maven)
 - c. `PATH=$JAVA_HOME/bin:$MAVEN_HOME/bin:$PATH`
 - d. Use the **which** command to verify that the above variables are set properly:
 - i. `% which java`
(Should display /tmp/installed/jdk1.8.0_151/bin/java on Linux)
 - ii. `% which mvn`
(Should display /tmp/installed/apache-maven-3.5.0/bin/mvn on Linux)
6. Set up the SOA server on the local machine in order to get build files from the local SOA server and upload them to the local Maven repository. Once this is done, sync the local Maven repository with the remote Maven repository. The next section describes how to patch the Oracle Middleware Home directory on the SOA server.
7. After you install SOA, export the following variables:
 - a. `MW_HOME=soaInstallationDirectory/Oracle/Middleware/Oracle_Home`
 - b. `ORACLE_HOME=soaInstallationDirectory/Oracle/Middleware/Oracle_Home`

3 PATCH THE SOA SERVER INSTALLATION

The following patches must be applied based on the SOA version installed. Three patches are provided for the SOA Installation, one each for releases 12.2.1.3.0, 12.2.1.2.0 and 12.2.1.1.0.

12.2.1.3.0 Patch #: 29142661 (12.2.1.3.181223 BP)

12.2.1.2.0 Patch #: 26647800

12.2.1.1.0 Patch #: 26673376

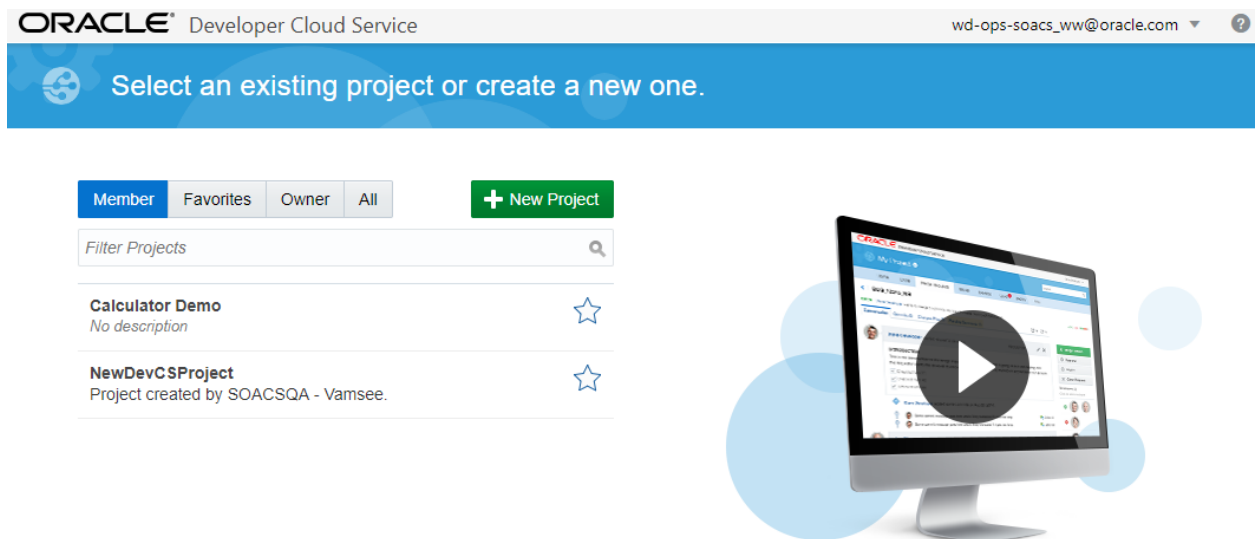
To install the patches, follow the instructions in [Patching Your Environment Using OPatch](#).

4 USE ORACLE DEVELOPER CLOUD SERVICE TO CREATE A NEW PROJECT

The following steps describe how to create a new project in Oracle Developer Cloud Service. For additional information refer to [Using Oracle Developer Cloud Service](#).

Note: If you have already created a project with Oracle Developer Cloud Service and have uploaded the required SOA projects to the repository, then skip to the next section.

1. Log in to the Oracle Developer Cloud Service console and create a new project.



2. Name the project "DeveloperCloudServiceExamples"

New Project ×

< Cancel Details Template Properties Next >

Project Details

* Name

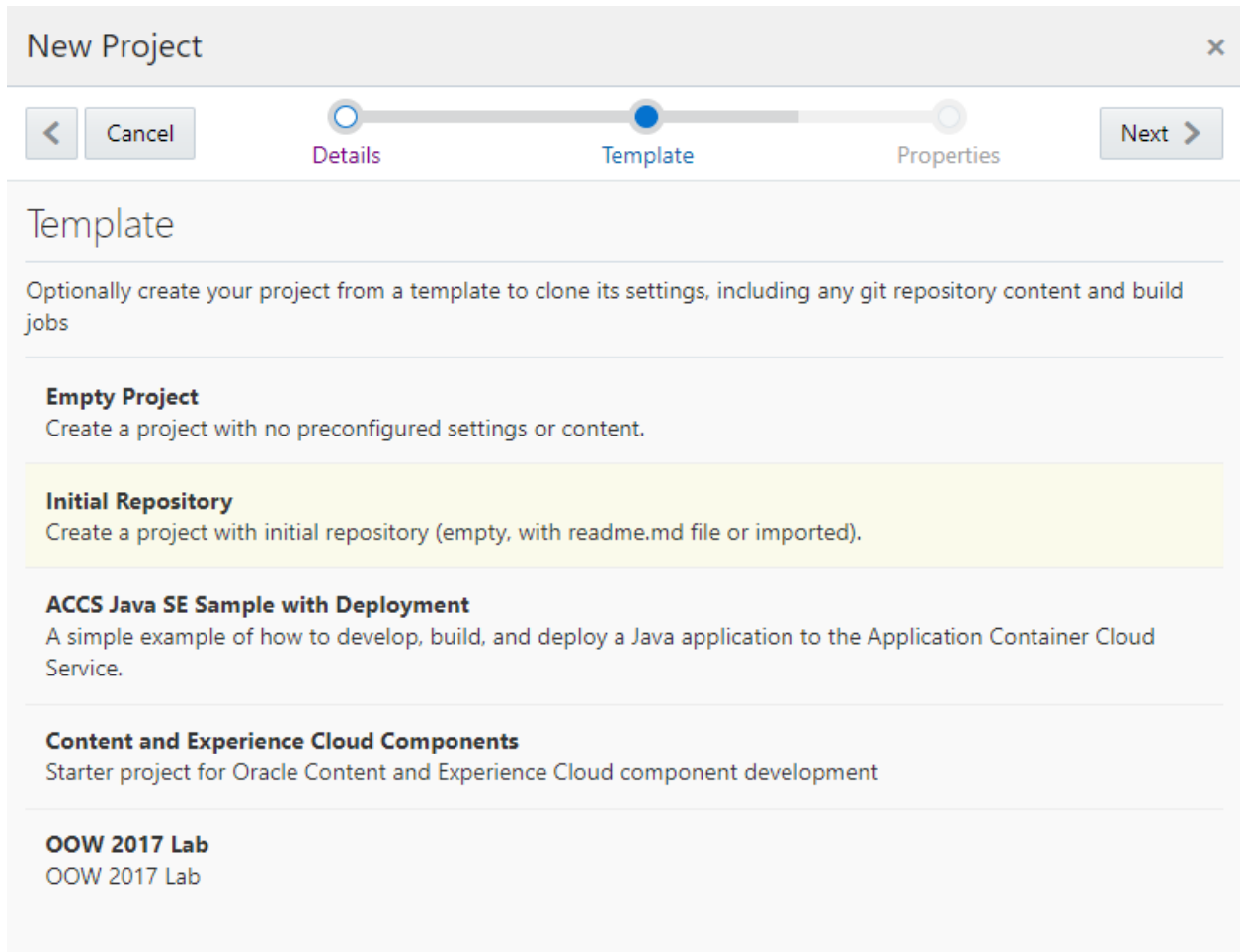
Description

? * Security Private Shared

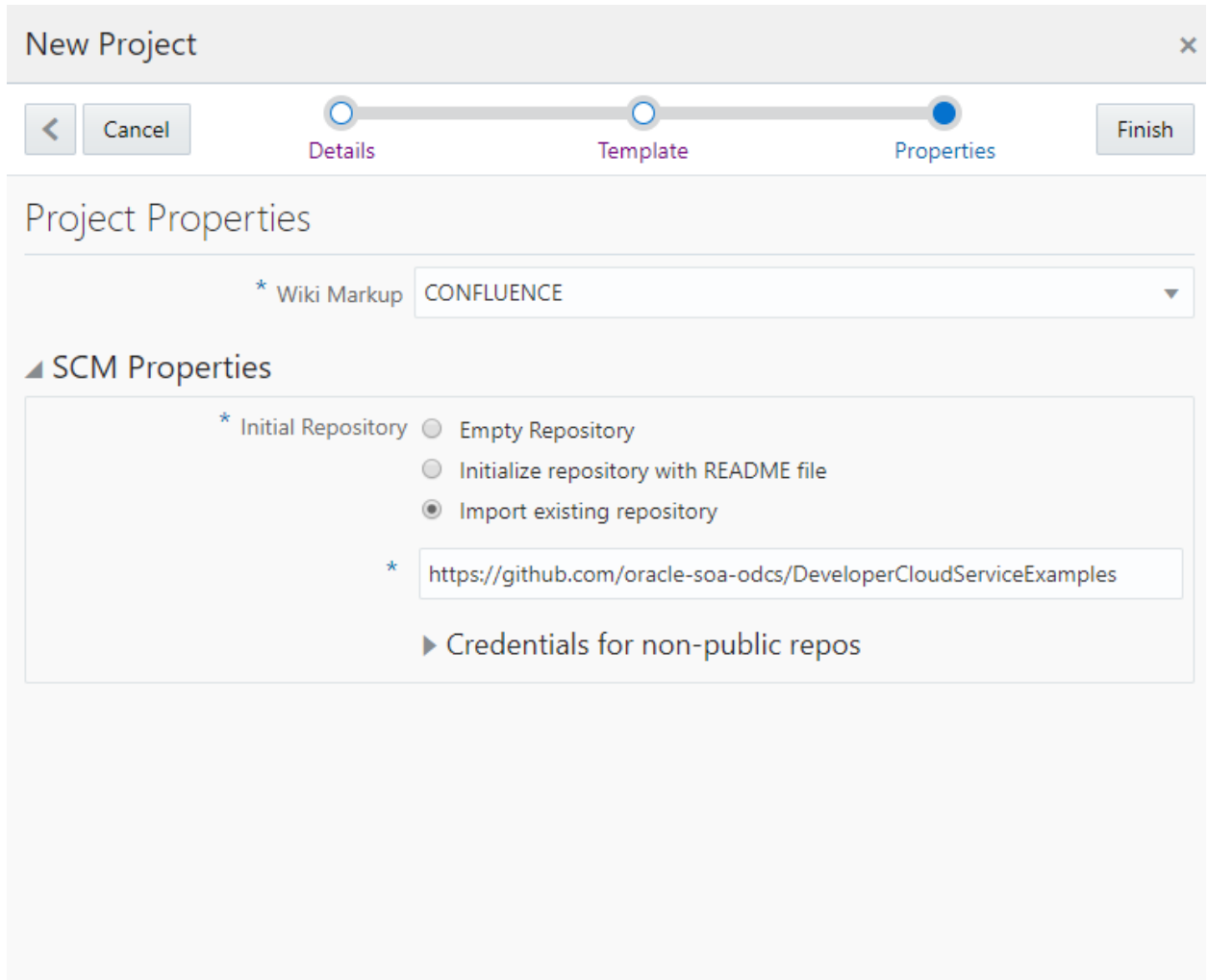
Preferred Language ▼

Click **Next**

3. Choose the **Initial Repository** template. Use an existing Git Repository to bootstrap your project.



4. Enter <https://github.com/oracle-soa-odcs/DeveloperCloudServiceExamples> in the **Import existing repository** field. This repository contains sample SOA projects that you can use for this exercise. You can replace it with your repository URL and import your projects into Oracle Developer Cloud Service.



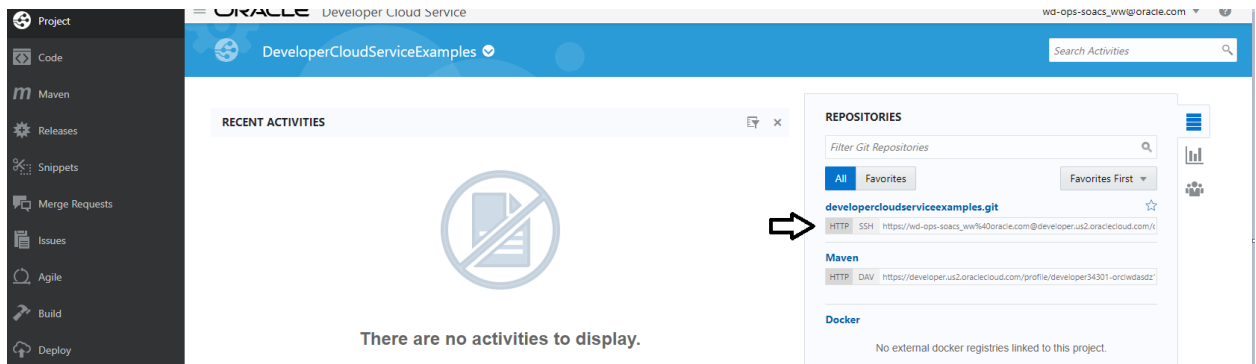
5. Click **Finish** and wait for Oracle Developer Cloud Service to set up the repository. When it finishes you will be at the projects landing page.

5 CLONE THE PROJECT

After you create your project, the source code of SOA projects resides within Oracle Developer Cloud Service's Git source repository. In order to work on it locally, use the following steps to download the contents of the remote Git repository into your local Git repository.

Note: The version of the design time for SOA must match the version of the runtime. If you use Oracle SOA Cloud Service 12.2.1.2, then the design time must also be 12.2.1.2. The following examples are for the 12.2.1-3-0 release. If you use an older release, you must modify the value of the `com.oracle.soa.plugin-version` attribute in the `pom.xml` file located in the `DeveloperCloudServiceExamples/version/HelloWorldExample/common/` directory.

1. From the project's landing page in Oracle Developer Cloud Service console, find the Git URL.



2. Clone the above repository into your local machine. Run the following command to make the clone, but change the URL to reflect your Oracle Developer Cloud Service project's repository URL:

```
% git clone https://wd-ops-soacs\_ww%40oracle.com@developer.us2.oraclecloud.com/1-orclwdasdzt14soa/s/developer34301-orclwdasdzt14soa/developercloudserviceexamples\_21100/scm/developercloudserviceexamples.git
```

You should see the following output after you run the **clone** command. (You will be prompted for a password.):

```
Cloning into 'DeveloperCloudServiceExamples'...
remote: Counting objects: 231, done
remote: Finding sources: 100% (231/231)
remote: Getting sizes: 100% (108/108)
Receivremote: Total 231 (delta 107), reused 229 (delta 107)
Receiving objects: 100% (231/231), 134.44 KiB | 0 bytes/s, done.
Resolving deltas: 100% (107/107), done.
```

3. If you are behind a proxy, use one of the following commands to inform Git about the proxy information:

```
git config --global http.proxy myproxy.mycompany.com:80
[git config --global proxyUrl:proxyPort]
```

4. The **clone** command creates a subdirectory called `DeveloperCloudServiceExamples` that acts as your local repository. Any change that you make in this repository must be pushed to the remote repository on Oracle Developer Cloud Service in order for the change to be reflected and be buildable or deployable from Oracle Developer Cloud Service.

6 ENABLE CONFIGURE OVERWRITE PROPERTY FOR RELEASE ARTIFACTS

You must enable the **Configure Overwrite Property for Release Artifacts** option in the Oracle Developer Cloud Service project before execution of the maven-sync plugin. See the following documentation to enable the **Configure Overwrite Property for Release Artifacts** option for the Maven Repository:

<https://docs.oracle.com/en/cloud/paas/developer-cloud/csdcs/use-projects-maven-repository.html#GUID-64CF3DE2-1552-440E-A704-DBAC39387523>

7 POPULATE THE LOCAL MAVEN REPOSITORY

There are multiple ways in which the local maven repository can be populated with JAR files from the *MW_HOME* directory of the local SOA Server. The Maven repository is the location that contains the JAR files required for Maven to compile and package your source code.

This section describes how to populate the local Maven repository from the *MW_HOME* directory of your SOA Installation.

A new Oracle Developer Cloud Service Project contains an empty repository (remote repository). You must populate the local Maven repository and then sync the local Maven repository with the remote Maven repository. This is a one-time setup that populates all the required files and JARs in the remote repository. Once the remote repository is populated, developers can use the Oracle Developer Cloud Service repository to get the binaries required for compilation in their local machines

Before populating the local Maven repository, please ensure that the SOA installation has been patched as described in the previous step.

Use the Oracle Maven Synchronization Plug-in to synchronize the Maven repository with *MW_HOME*. This plug-in introspects the *MW_HOME* directory and uploads the `pom.xml` file and referenced binary into the repository. To perform this operation, follow these steps:

1. Ensure that the Maven `settings.xml` file contains proxy configuration settings. Running the Maven command for first time downloads several artifacts from the Maven central repository. Your system may require a proxy to connect to Maven central. For this update the `$MAVEN_HOME/conf/settings.xml` file contains proxy information.
2. Use the following command to install the Oracle Maven Synchronization Plug-in from *MW_HOME*. Note that the version might be different, 12.2.1 is used here. Also note the directory change; you want to be in a directory without a POM or with a bland POM otherwise Maven attempts to evaluate it.

```
% cd calculator-demo/  
% mvn install:install-file \
```

```
-DpomFile=$MW_HOME/oracle_common/plugins/maven/com/oracle/  
maven/oracle-maven-sync/12.2.1/oracle-maven-sync-12.2.1.pom  
-Dfile=$MW_HOME/oracle_common/plugins/maven/com/oracle/  
maven/oracle-maven-sync/12.2.1/oracle-maven-sync-12.2.1.jar
```

3. Once the plug-in is installed, run it with the following command:

For release 12.2.1.2:

```
% mvn -s $MAVEN_HOME/conf/settings.xml com.oracle.maven:  
oracle-maven-sync:12.2.1-2-0:push -DoracleHome=$MW_HOME
```

For release 12.2.1.3:

```
% mvn -s $MAVEN_HOME/conf/settings.xml com.oracle.maven:  
oracle-maven-sync:12.2.1-3-0:push -DoracleHome=$MW_HOME
```

Note: Execute the following command from a directory that doesn't contain `pom.xml`. Also, change the version number as appropriate for your environment.

This command takes a while to run because it copies JAR files from `MW_HOME` into the local Maven repository.

4. Once the sync completes, you should be able to run the following command:

```
% cd DeveloperCloudServiceExamples/12.2.1.3/HelloWorldExample/  
HelloWorldApp/Project1  
% mvn clean compile
```

The next section describes how you can populate the remote Maven repository. The Oracle Developer Cloud Service remote Maven repository has to be populated only once, after that the developer can use the Oracle Developer Cloud Service remote repository to download the JARs required to build and deploy on local machines.

8 POPULATE THE ORACLE DEVELOPER CLOUD SERVICE MAVEN REPOSITORY

There is more than one way to populate the Oracle Developer Cloud Service remote Maven repository, this document describes how to use the previously installed Oracle Maven Synchronization Plug-in to populate the remote repository.

To use the Oracle Maven Synchronization Plug-in with remote servers, you must first define the `<repository>` and `<server>` tags in `settings.xml`.

Include a `<server>` tag in the Maven `settings.xml` file. A sample is listed below:

```

<server>
  <id>MyMavenRepository</id>
  <username>clara.coder@oracle.com</username>
  <password>f@nt@5t1c</password>
</server>

```

You can enter the password as it is or encrypt it using Maven encryption. The recommended approach is to use Maven to encrypt it.

Add a profile to the Maven `settings.xml` file. This profile is used to populate the remote Maven repository. The following is a sample of the profile:

```

<profile>
  <id>RemoteMavenRepository</id>
  <activation>
    <activeByDefault>true</activeByDefault>
  </activation>

  <repositories>
    <repository>
      <id>MyMavenRepository</id>
      <name>Remote Maven Repository for your DevCS project</name>
      <!-- CHANGE ME! -->
      <url>URL to your ORACLE DEVELOPER CLOUD SERVICE Maven Repository. This can be
      obtained from Oracle Developer Cloud Service Landing Page</url>
      <layout>default</layout>
    </repository>
  </repositories>

  <pluginRepositories>
    <pluginRepository>
      <id>MyMavenRepository</id>
      <name>Remote Maven Plugin Repository for your DevCS project</name>
      <!-- CHANGE ME! -->
      <url>URL to your ORACLE DEVELOPER CLOUD SERVICE Maven Repository. This can be
      obtained from Oracle Developer Cloud Service Landing Page</url>
      <layout>default</layout>
    </pluginRepository>
  </pluginRepositories>
</profile>

```

Remember that the `<id>` used in `<repository>` and `<pluginRepository>` should match that of the `<id>` used in the `<server>` tag.

Now you can run the Oracle Maven Synchronization Plug-in again using the profile described above:

For release 12.2.1.2:

```

% mvn -s $MAVEN_HOME/conf/settings.xml com.oracle.maven:oracle-maven-
sync:12.2.1-2-0:push -DoracleHome=$MW_HOME -P RemoteMavenRepository -
DserverId=MyMavenRepository

```

For release 12.2.1.3:

```
% mvn -s $MAVEN_HOME/conf/settings.xml com.oracle.maven:oracle-maven-  
sync:12.2.1-3-0:push -DoracleHome=$MW_HOME -P RemoteMavenRepository -  
DserverId=MyMavenRepository
```

Note: Be sure not to execute the following command from a directory that contains a `pom.xml` file.

This command can take many hours to complete.

The Oracle Developer Cloud Service Maven repository can be accessed without a proxy if your company's network policy permits. Accessing Oracle Developer Cloud Service without using a proxy can enhance performance while populating the Maven remote repository. To access the repository without a proxy, add the Oracle Developer Cloud Service Maven repository URL to `<nonProxyHosts>` in `settings.xml`. For example:

```
<proxy>  
  <id>optional</id>  
  <active>>true</active>  
  <protocol>http</protocol>  
  <host>YOUR_ORGANIZATION_PROXY_HOST</host>  
  <port>PROXY_PORT</port>  
  <nonProxyHosts>DEV_CS_MAVEN_REPOSITORY_URL</nonProxyHosts>  
</proxy>
```

9 MAVEN DEPLOY TO SOA SERVERS ON THE LOCAL MACHINE

Prerequisites:

1. SOA must be installed on the local machine.
2. The SOA server must be up and running.
3. Copy the file `cacerts` from `$JAVA_HOME/jre/lib/security/` and put it into the `DeveloperCloudServiceExamples/12.2.1.3/HelloWorldExample/HelloWorldApp/certificates/` folder. You will use this file as a trust store to store certificates required to deploy to Oracle SOA Cloud Service (in case Oracle SOA Cloud Service requires a certificate be used in the client side to access Oracle SOA Cloud Service).

The certificate has to be installed in the keystore that you use (as mentioned above). You will push these changes to Oracle Developer Cloud Service.

In

`DeveloperCloudServiceExamples/12.2.1.3/HelloWorldExample/common/pom.xml`, there are profiles called `soaDeployEnv_DEV`, `soaDeployEnv_TEST`, and `soaDeployEnv_PROD`. As you can guess by their names, these profiles contain deployment information for various environments. Modify and use `soaDeployEnv_DEV` in this example.

The file

`DeveloperCloudServiceExamples//12.2.1.3/HelloWorldExample/common/pom.xml` also contains several properties that you might have to change to reflect the correct values.

Here are a few of them:

`com.oracle.soa.plugin-version`: Should have a value of `12.2.1-3-0` (for the `12.2.1-3-0 Release`)

`soaServerDeployProtocol`: Should be set to `HTTP` or `HTTPS` according to the SOA Server configuration.

In the profile section, modify the values described in the profile `soaDeployEnv_DEV` and set the values for the local server:

`soaServerHost`: The host where the Oracle SOA Cloud Service server is running. The host where OTD is running if OTD is present.

`soaServerPort`: If the deploy protocol is `HTTP`, then use port `80` on the SOA managed server, if the protocol is `HTTPS` use `443`. If you are deploying to a SOA server on a local machine, you can use port `7002`.

`soaServerUsername`: The user name used to log in to Oracle Enterprise Manager Fusion Middleware Control Console.

`soaServerPassword`: The password used to log in to Oracle Enterprise Manager Fusion Middleware Control Console.

The following commands deploy the `Project1` composite into the local SOA server Installation:

```
% Cd DeveloperCloudServiceExamples/12.2.1.3/HelloWorldExample/  
HelloWorldApp/Project1  
  
% mvn clean pre-integration-test -P soaDeployEnv_DEV
```

10 MAVEN DEPLOY TO ORACLE SOA CLOUD SERVICE FROM LOCAL MACHINES

Please read the above section and modify the values for the `soaDeployEnv_TEST` profile with the values replaced for Oracle SOA Cloud Service.

Execute the following command:

```
% mvn clean pre-integration-test -P soaDeployEnv_TEST  
-Djavax.net.ssl.trustStore=path to your cacert file
```

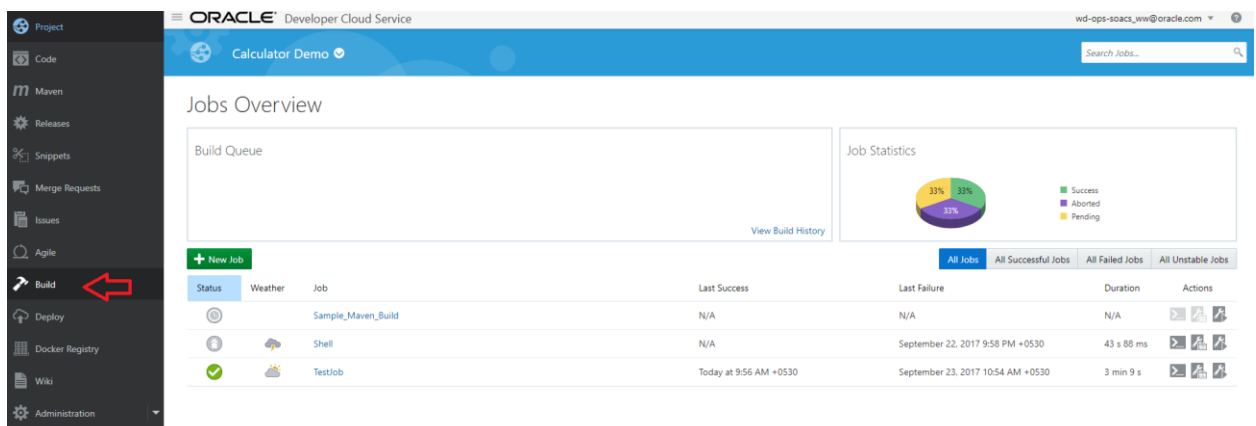
The `-Djavax.net.ssl.trustStore` parameter should only be added when Oracle SOA Cloud Service is accessible over `HTTPS` and requires a certificate.

Please refer to Section 11 for information about installing a certificate to a trust store.

11 BUILD YOUR PROJECT ON ORACLE DEVELOPER CLOUD SERVICE AND DEPLOY TO ORACLE SOA CLOUD SERVICE

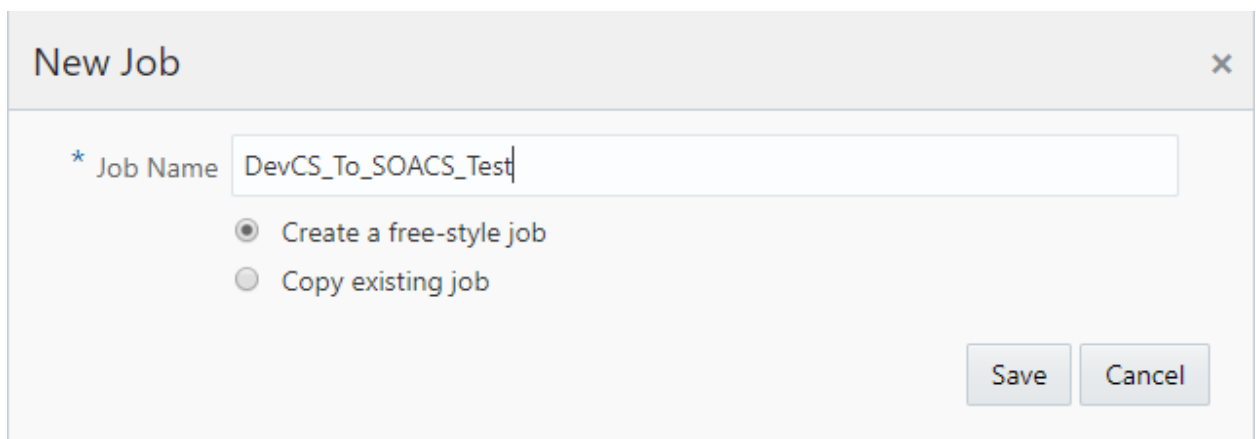
This step requires you to configure a build job in the Oracle Developer Cloud Service console. The following steps guide you thru this configuration.

1. From the Oracle Developer Cloud Service landing page, navigate to the **Build** tab



Status	Weather	Job	Last Success	Last Failure	Duration	Actions
🕒	☁️	Sample_Maven_Build	N/A	N/A	N/A	🔍 🗑️ 🔄
⚠️	☁️	Shell	N/A	September 22, 2017 9:58 PM +0530	43 s 88 ms	🔍 🗑️ 🔄
✅	☁️	TestJob	Today at 9:56 AM +0530	September 23, 2017 10:54 AM +0530	3 min 9 s	🔍 🗑️ 🔄

2. Click **New Job** to create a new job. Name it something memorable such as “DevCS_To_SOACS_Test”.



New Job

* Job Name

Create a free-style job

Copy existing job

Save Cancel

3. Click **Configure**.
4. In the **Main** tab select **JDK 8** from the dropdown list.

The screenshot shows the 'Configure build job' page for a job named 'DevCS_To_SOACS_Test'. The 'Main' tab is selected, and the 'JDK' dropdown is set to 'JDK 8'. There are three unchecked checkboxes: 'Disable build', 'Execute concurrent builds if necessary', and 'Discard old builds'.

5. In the **Build Parameters** tab, check the **This build is parameterized** checkbox and then select **String Parameter** from the **Add Parameter** dropdown menu. Enter **MW_HOME** as the name and **\${MIDDLEWARE_HOME_SOA_12_2_1}** as the value.

The screenshot shows the 'Configure build job' page in the 'Build Parameters' tab. The 'This build is parameterized' checkbox is checked. A 'String Parameter' is being added with the name 'MW_HOME' and the default value '\${MIDDLEWARE_HOME_SOA_12_2_1}'.

This value is required by `common/pom.xml`.

6. In the **Source Control** tab, click **Git** and choose the only repository from the dropdown menu. Click **Add** on the right side of the **Branches** section. You should see a branch called "master" on the **Branch Specifier** dropdown menu.

Main Build Parameters **Source Control** Triggers Environment Build Steps Post Build Advanced

To integrate the Build System with Source Control, select an option below and then configure the required settings.

None
 Git

Repositories Add

* Repository calculator-demo.git x

▶ Advanced Repository Settings

Branches Add

Branch Specifier master x
Leave blank for default


▶ Advanced Git Settings

7. In the **Triggers** tab, keep the default settings. You will manually trigger this job for the time being:

Main Build Parameters Source Control **Triggers** Environment Build Steps Post Build Advanced

When these jobs are built
 Based on this schedule
 Based on SCM polling schedule
 When Maven dependencies have been updated by Maven 3 integration

8. In the **Environment** tab keep the default settings.

 DeveloperCloudServiceExamples ▼

[Jobs Overview](#) | [Test](#) | Configure build job

Main Build Parameters Source Control Triggers **Environment** Build Steps Post Build Advanced

Start Xvfb before the build, and shut it down after
 Abort the build if it's stuck
 Add Timestamps to the Console Output
 Connect Oracle Maven Repository
 Use NodeJS version

- In the **Build Steps** tab, click **Add Build Step** and select **Invoke Maven 3**. Enter the information as it appears below:

The screenshot shows the Jenkins 'Configure build job' page for a job named 'Invoke Maven 3'. The 'Build Steps' tab is active. The configuration includes:

- Maven 3** (Bundled)
- Goals:** clean pre-integration-test
- Properties:** (empty)
- POM File:** 12.2.1.3\HelloWorldExample\HelloWorldApp\pom.xml
- Repositories:** Private repository, Private temporary directory, Offline
- Profiles:** RampartMavenRepository, useDeployEnv_TEST, UseProxy
- Verbose:** Show Errors
- Verbosity:** NORMAL
- Checksum Mode:** NORMAL
- Snapshot Updates:** NORMAL
- Recursive:** (checked)
- Projects:** Project1
- Resume From:** (empty)
- Fail Mode:** NORMAL
- Make Mode:** NONE
- Threading:** (empty)
- JVM Options:** -Djavax.net.debug=off -Djavax.net.ssl.trustStore=C:\Users\jenkins\workspace\developer\4301-ocrid\demo\1404_calculator_demo_20323\TestJob\CalculatorApplication\Project1\certificates\cacerts

The JVM option that specifies `javax.net.ssl.trustStore` can be set according to the location of `cacert` file in your project.

Define the profile "UseProxy" on

DeveloperCloudServiceExamples/12.2.1.3/HelloWorldExample/common/pom.xml:

```
<profile>
  <id>UseProxy</id>
  <properties>
    <http.proxyHost>${env.HTTP_PROXY_HOST}</http.proxyHost>
    <http.proxyPort>${env.HTTP_PROXY_PORT}</http.proxyPort>
    <https.proxyHost>${env.HTTPS_PROXY_HOST}</https.proxyHost>
    <https.proxyPort>${env.HTTPS_PROXY_PORT}</https.proxyPort>
    <http.nonProxyHosts>${env.NO_PROXY}</http.nonProxyHosts>
  </properties>
</profile>
```

- In the **Post Build** tab, keep the default settings.

Main Build Parameters Source Control Triggers Environment Build Steps **Post Build** Advanced

- Aggregate downstream test results
- Build other jobs
- Archive the artifacts
- Record fingerprints of files to track usage
- Publish Javadoc
- Publish JUnit test result report
- Archive Maven 3 artifacts
- Record fingerprints of Maven artifacts
- Git publisher
- Notify that Maven dependencies have been updated by Maven 3 integration
- Oracle Cloud Service Deployment

11. In the **Advanced** tab, keep the default settings.

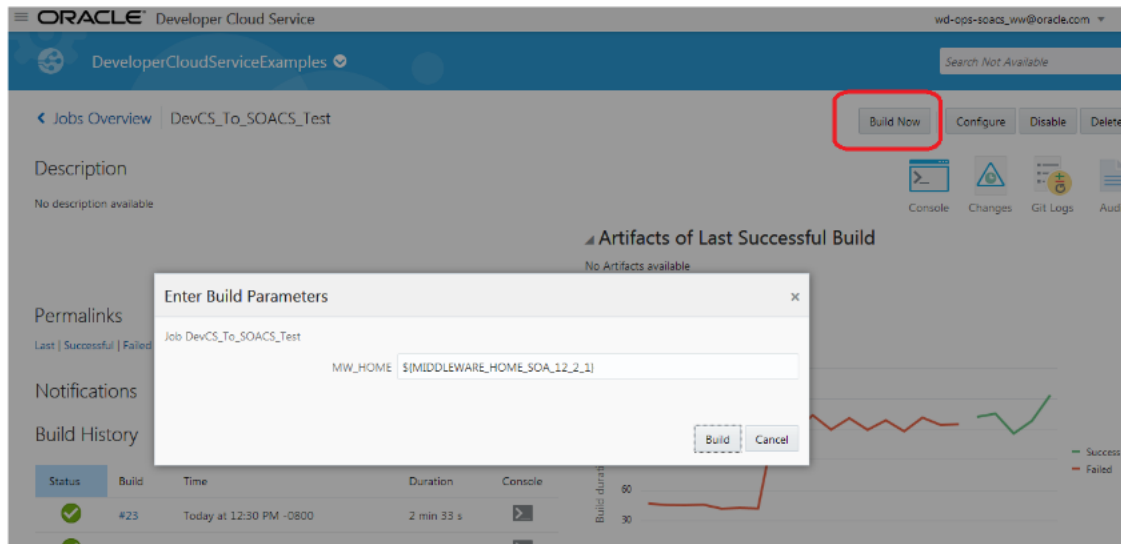
Main **Build Parameters** Source Control Triggers Environment Build Steps Post Build **Advanced**

- Quiet period
- Retry count
- Block build when upstream job is building
- Block build when downstream job is building

12. Click **Save**.

13. Make sure all the changes made in the local repository are pushed to the remote repository on Oracle Developer Cloud Service, then click **Build Now**.

14. When prompted to confirm **MW_HOME** is `${MIDDLEWARE_HOME_SOA_12_2_1}`, click **Build**.



12 ISSUES WITH CERTIFICATES

Currently there is an issue whereby deploying to Oracle SOA Cloud Service requires a certificate. If the Oracle SOA Cloud Service server uses HTTPS and requires a certificate to be installed at the client side, then it has to be done in a `cacert` file that's located at `DeveloperCloudServiceExamples/12.2.1.3/HelloWorldExample/HelloWorldApp/certificates/cacert`

The certificate from the Oracle SOA Cloud Service server has to be imported into the `cacert` file and then the file is pushed to the Oracle Developer Cloud Service Git repository.

The absolute path to this file has to be provided as a parameter to the key `javax.net.sslTrustStore`.

The following command installs a certificate to the trust store:

```
% keytool -import -alias SOACSCertificate -file certificateFileName -keystore cacerts -storepass changeit
```