Java Integration

HPCC Systems Platform 4.x and greater has the ability to integrate with Java directly. This page takes you through a few steps that you have to implement to configure Java correctly. In this particular case we will see how to make it work on a Ubuntu 13.04 system

Configuring HPCC for Java Integration

1. **Download and Install the HPCC Systems platform with plugins 4.x and greater**

   Follow the installation instructions [here] for downloading HPCC & installing it. Specific instructions for installing the package with plugins are on page 9 of Installing & Running the HPCC Systems® Platform

   For the RPM based system you will need the distro beginning with **hpccsystems-platform_community-with-plugins-**, NOT hpccsystems-platform_community-

   You must install the packages that have the plug-ins using the --nodeps option. For example: sudo rpm -Uvh --nodeps <rpm file name>

   For Debian based systems there is only the hpccsystems-platform_community- package available and it includes plugin support.

2. **Install OpenJDK 1.7 or greater**

   In some cases you may have to install the default-jdk package as well.

Check that Java Plugins are Working

1. **If you haven't run java on the cluster before, verify that Javaembed is functioning correctly on the cluster**
   a. Run sample java like the following. JavaCat is installed on the HPCC Cluster by default.

```
IMPORT java;
integer add1(integer val) := IMPORT(java, 'JavaCat.add1:(I)I');
output(add1(10));
```

If you get output, you have installed java successfully.

If you get a "unable to load libjvm.so" error you should reinstall java, or try a different Java package.

If you get an error accompanied by "Warning: /usr/bin/ld: cannot find -ljavaembed", the HPCC install does not include the plugin feature and needs to be reinstalled from a distro with the plugin feature.

Deploy your Java Jar File

1. **Verify that your java class was not compiled with a more recent version of java than is on the cluster.**
   a. You can check this by running "rpm -qa|grep java" on one of the cluster nodes.
2. **Copy the Java jar or class file to all of the THOR nodes on a cluster.**
   a. The default location for java files is /opt/HPCCSystems/classes.
   b. This can be done manually or by running something like:

      ```
      for x in 'seq 8';do scp myjava.jar 10.173.147.$x:/opt/HPCCSystems/classes/ ;done
      ```

      The /opt/HPCCSystems/classes folder is owned by root user. The HPCC user (hpcc) doesn't have permission to write. By default, the HPCC installation only sets ssh key pairs for the HPCC user which allow ssh/scp between the hosts without prompting password. So, to run above script you need one of following methods:

      1) run as root and provide password for each host

      2) add root ssh public key to authorized file (/root/.ssh/authorized_keys and run as root

      3) change /opt/HPCCSystems/classes permission or ownership to allow HPCC user has write permission and run the script as HPCC user.
3. **Set the classpath in the HPCC Systems configuration file**
   a. The JAR file itself can be physically located anywhere. You can add the JAR file to the classpath by adding it to /etc/HPCCSystems /environment.conf, or by adding it to the Java global classpath environment variable.
   b. Edit the environment.conf in your favorite editor and add your java class/jar to the classpath entry
      i. If you are adding a jar file, the jar file itself has to be added to the classpath. For example:

      ```
      classpath=/opt/HPCCSystems/classes:/opt/HPCCSystems/classes/myjava.jar
      ```

4. **Restart the thor cluster for the classpath changes to take effect**

   sudo service hpcc-init start => command for starting

   sudo service hpcc-init stop => command for stopping

   sudo service hpcc-init restart => command for restarting

Call your Java from ECL

Currently, you can only pass primitive data types to and from a Java plugin (String, long, etc.)

Define the interface for the java class you're calling. For example, if you're calling a static method SegmentText in the class Segmenter with the definition of

```
public static String SegmentText(String input, String config)
```

You would place the following code in your ecl:

```
import java;

STRING segment() := IMPORT(java,'org/hpccsystems/Segmenter.SegmentText:(Ljava/lang/String;Ljava
/lang/String;)Ljava/lang/String;');
```

You must include the entire path to the class being called within the jar to the IMPORT statement.

To pass complex data from a Java class, the best practice is to pass it as an xml or otherwise encoded/delimited string, and then break the xml/delimited fields into separate fields for processing. Check the simple Java Integration sample below for an example of how to turn an xml string returned from a java class into a dataset. An additional example can be found at https://github.com/hpcc-systems/TextAnalytics/blob/master/hpcc/ecl/medex/BWR_MedexDrugSignatures.ecl

the Java JVM starts up once at Thor startup, and continues running until Thor is restarted. Any classes you call remain loaded until Thor restart. Call static class methods, or use a Factory method pattern in any java code you're calling. When possible, incorporate all dependencies, configuration files and other resources within the jar to help avoid any classpath/dependency issues that could arise when running the JAR on the cluster.

Troubleshooting

1. **You can run the sample java, but get a class not found error when you try to run yours.**

   This is because HPCC can't find your jar or class in its classpath. First things to check:

   a. Verify that the entry in /etc/HPCCSystems/environment.conf is pointing to the correct jar file
   b. Verify that your jar file is located where expected
   c. Verify that the class package location in the jar matches the path defined in the ECL IMPORT Statement
   d. Verify that the jar is correctly deployed on all thor nodes in the cluster.
   e. Verify that the jar file is world readable and executable.
   f. Verify that the thor cluster was restarted after you added your entry to the classpath.
   g. if you are using a pre 5.0 hpcc build, the environment.conf's classpath variable was limited to 1024 characters; check if this is the problem.

   To determine exactly what classpath HPCC is using, you can create the following java class. Call it from the command line and from ecl IDE and see what path is being used in each case.

```
public class TokenizerTest{
   public static void main(String[] args){
      System.out.println(GetThePath());
   }
   public static String GetThePath(){
      return System.getProperty("java.class.path");
   }
}
```

   If you do see this entry in the HPCC logs and your jar is included in the classpath, the error is either a) in the import statement syntax in ECL or b) an internal error in your java code causing the problem. See #2 for more on debugging this.

2. **You get unexpected errors when running your java in ECL: java.lang.ExceptionInInitializerError, etc.**

Put a Main in your java class that runs the same code as is being run in ECL. Try calling the jar file from the command line on the HPCC Cluster. (for example, java -cp /opt/HPCCSystems/classes/myjava.jar /org/hpcc/MyClass). Chances are you'll see the same error. If you do, debug the java from here; if the error only occurs when running from ECL, raise an issue at http://track.hpccsystems.com

3. You get a java.lang.NoClassDefFoundError error when running your Java from ECL; the error doesn't occur when running the same java from the command line.

In one case, where java code was creating an instance of Jboss Drool's specific classes, this error popped up because the contextClassLoader was null. Adding the following code to the Java class fixed the error.

```
if(Thread.currentThread().getContextClassLoader() == null)
   { Thread.currentThread().setContextClassLoader(String.class.getClassLoader()); }
```

More details are in this ticket: ~~HPCC-10204~~ - **java.lang.NoClassDefFoundError** RESOLVED

4. You try running your java, and get an out of memory or heapspace error.

In HPCC, the JVM is started when Thor starts and continues to run until Thor is restarted. Any java classes loaded into memory during this time remain there.

You can increase your maximum memory for the JVM by adding the following line to /etc/HPCCSystems/environment.conf: jvmoptions=-Xmx1024m

Alternately, if your java plugin requires a large amount of data loaded in memory (such as lexicons, etc. needed by Natural Language Processing java jars), you can add a method to your java class to clear out cached data after a process is run so that it does not continue to take up space in the Java heap. For example:

```
import java;
STRING segment() := IMPORT(java,'org/hpccsystems/Segmenter.SegmentText:(Ljava/lang/String;Ljava/lang/String;)
Ljava/lang/String;');
STRING clearcache() := IMPORT(java,'org/hpccsystems/Segmenter.ClearCache:(Ljava/lang/Boolean)Ljava/lang
/String;');

SEQUENTIAL(output(segment('text to segment')),
           output(clearcache(true))
)
```

You can also view additional java-related HPCC issues that have occurred in JIRA or raise an issue at http://track.hpccsystems.com to get help resolving a java issue.

A simple Java Integration example

The idea is to create a Java class that acts as a consumer of external data (think Kafka consumer). For sanities sake let us create a simple implementation of a class with a static method that returns a string. Making this a true Kafka consumer will be material for another Wiki page.

The Java Consumer Class

```
package org.hpccsystems.streamapi.consumer;

public class DataConsumer {

        public static String consume() {
                return "<dataset><rows><row>sample row1</row><row>sample row2</row></rows></dataset>";
        }

}
```

Now, assuming that you have Java configured correctly (if not, read the setting up Java wiki), the sample ECL code to call the Java class will look like:

The ECL Script

```
IMPORT java;

STRING consume() := IMPORT(java,
        'org/hpccsystems/streamapi/consumer/DataConsumer.consume:()Ljava/lang/String;');


messages := consume();

OUTPUT(messages);

messagesDS := DATASET([{messages}], {STRING line});

ExtractedRow := RECORD
  STRING value;
END;

ExtractedRows := RECORD
  DATASET(ExtractedRow) values;
END;

ExtractedRows RowsTrans := TRANSFORM
  SELF.values := XMLPROJECT('row', TRANSFORM(ExtractedRow, SELF.value := XMLTEXT('')));
END;

parsedData := PARSE(messagesDS, line, RowsTrans, XML('/dataset/rows'));

OUTPUT(parsedData);
```

The calling of the Java consume method is really accomplished in the first three lines. The rest of the code is used to extract the XML content into something more meaningful.

Give it a try and see how easy it is to extend HPCC using Java libraries. HPCC provides you the framework to perform Big Data analytics. This example shows how you can easily extend ECL to perform advanced tasks like streaming data, text extraction, sentiment analysis etc., using Java libraries.

Additional Examples of Java Plugin usage:

https://github.com/hpcc-systems/TextAnalytics/tree/master/hpcc/stanfordnlp
https://github.com/hpcc-systems/kafka-integration/blob/master/ecl/DataCollection_Scheduler.ecl