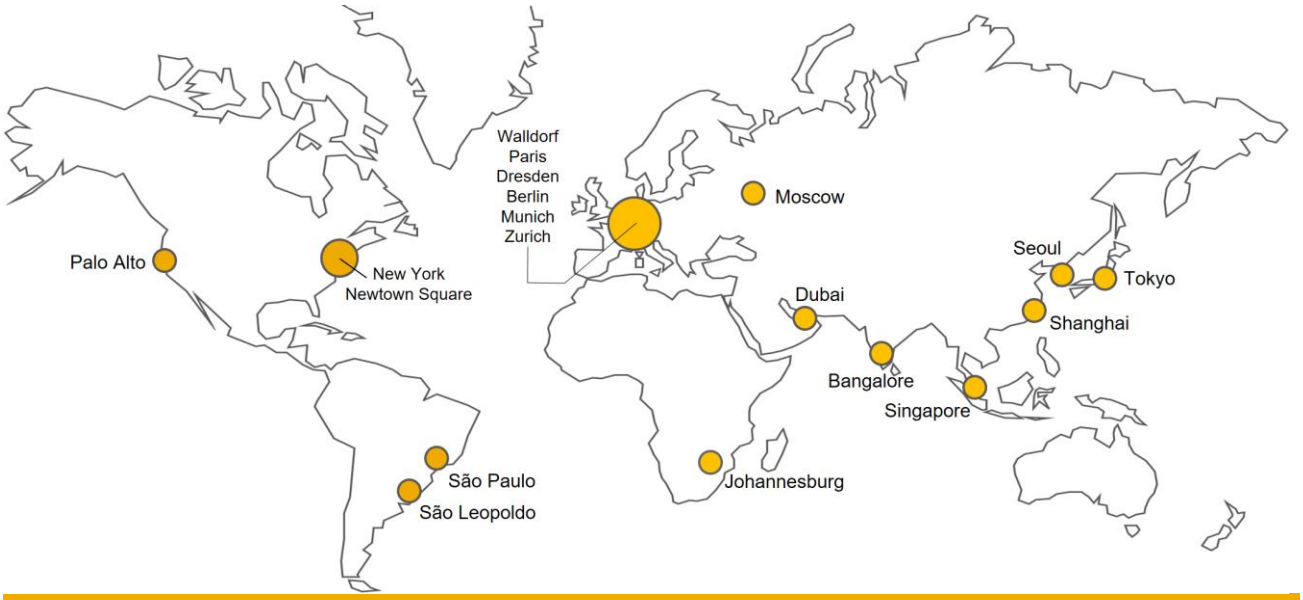
Integration of MATLAB Compiler Runtime in SAP HANA® Extended Application Services, Advanced Model

Technical How-to Guide



Authors: **Jean Zimmermann (statmath GmbH)**
Goran Stoiljkovski (SAP Co-Innovation Labs Network)

SAP Run Simple

statmath
Institut für Data Science & Prognosen

TABLE OF CONTENTS

During the summer of 2017, we at SAP® Co-Innovation Lab were approached by colleagues from the IoT development line of business at SAP with a topic that we all thought was worth brainstorming. They were looking for a way to closely integrate MathWorks' MATLAB with the SAP HANA® platform. MATLAB is a product suite for data analytics (and beyond). It is very popular at universities and research institutes, where it is enjoying widespread adoption among students, scholars, and research staff. SAP HANA is a technology platform for developing and running the data-intense applications that are experiencing accelerated adoption among leading enterprises – globally. The idea was to explore a way to integrate the two in a best possible way from a technology perspective. We decided to do this in a technical proof of concept, which we conducted in the SAP Co-Innovation Lab environment. Right from the beginning, a German solution provider and system integrator was part of the discussions: statmath GmbH. statmath has deep and sound expertise in MATLAB. The company is also an official partner and member of MathWorks' developer community.

This paper is about the findings of this project – especially the ones we thought would be interesting to the SAP HANA developer community.

1 BRIEF PROJECT DESCRIPTION AND USE CASE

MATLAB is an entire suite of products. It comprises frameworks and toolboxes, ranging from core engines for numerical computations to graphical modeling tools. In this proof of concept, we solely took some core Java libraries and deployed them on Apache Tomcat in SAP HANA extended application services, advanced model. Once you deploy and import these Java libraries into SAP Web IDE, you can use them as reference libraries for developing custom Java Web applications on the advanced model of SAP HANA extended application services.

We will describe the steps and the results in more detail in the sections to come.

Before we start, let us take a short digression and set the stage by explaining some well-known facts about the platform we are targeting.

1.1 Notes on SAP HANA Extended Application Services, Advanced Model

SAP HANA extended application services, advanced model is a platform for developing and running "native" data-intensive applications. These "native" applications are native to SAP HANA, thus taking advantage of its in-memory architecture and parallel processing capabilities. The advanced model of SAP HANA extended application services implements a microservices-oriented architecture design based on Cloud Foundry. One of its major advantages worth emphasizing is found in the following note from the official *Developer's Guide to SAP HANA Extended Application Services, Advanced Model*:

"SAP HANA extended application services, advanced model adds an application platform to the SAP HANA in-memory database. In the cloud, this platform is provided by Cloud Foundry. An SAP-developed runtime environment is bundled with SAP HANA on premise, which provides a compatible platform that enables applications to be deployed to both worlds."

Putting it in a casual way, this note would translate into something like:

- In the cloud[1], the application platform **is** provided by Cloud Foundry.
- In the on-premise case[2], the application platform is provided by SAP but **based on** Cloud Foundry.
- This concept helps ensure the compatibility of the two and allows for deployment of an application in both worlds without code or configuration changes.

1. Read: SAP Cloud Platform / Cloud Foundry
2. Read: SAP HANA extended application services, advanced model running on SAP HANA on premise

Note: We personally expect convergence of these platforms over time towards the Cloud Foundry standard.

1.2 Notes on Java in SAP HANA Extended Application Services, Advanced Model

The advanced model of SAP HANA extended application services is a canonical evolution of the classic model – for example, by introducing new programming models. One of them is Java, including Java SE and Java EE. In the Java EE case, this means there are embedded Tomcat and TomEE servers (provided as build packs) that can be used as runtime environments for developing and running Java-based Web applications.

A classic dilemma of a Java EE **deployment** is how to migrate and port a Java EE-based application across multiple **runtimes**. Please note the emphasis on "deployment" and "runtimes" in the last sentence. One of the major challenges here is to deploy the libraries (JARs) in a way compatible with the class loading of the target runtime environment. Java EE runtimes often have "extended" (proprietary) class loading concepts – extending the standard JVM class loading implementation. In our case, this dilemma is alleviated by the fact that we didn't have different runtimes; in both cases, the runtime (server) is Tomcat.

The major task is to import the existing Java archives (JARs) in SAP Web IDE. This is a task you perform during development. While developing Java code with SAP Web IDE on the advanced model of SAP HANA extended application services, you get build automation for free. SAP Web IDE uses Maven behind the scenes, and Maven manages the code project and its dependencies.

Note: SAP Web IDE expects at least three Maven attributes in a JAR, which are to be imported: **groupId**, **artifactId** and **version**.

2 SYSTEM SETUP

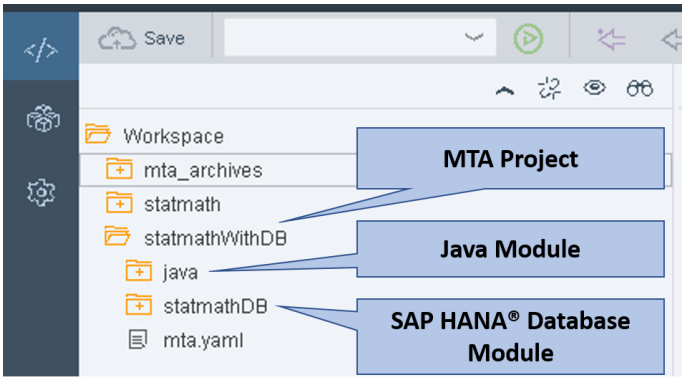
We used an on-premise instance of **SAP HANA 2.0 SPS02** hosted in the computing center at SAP Co-Innovation Lab as our target deployment and development platform. This means we had full access to the SUSE Linux OS instance. Keep in mind that you won't normally get this access in a cloud environment such as SAP Cloud Platform.

We had the core MATLAB Java libraries (version 2017b) that we wanted to deploy on SAP HANA extended application services, advanced model. Those libraries also have native parts performing Java Native Interface (JNI) calls, and we needed to make sure that the native libraries were also in place. These libraries are also known as the **MATLAB Compiler Runtime (MCR)**.

2.1 Setting Up the Development Project in SAP Web IDE for SAP HANA

In our case, we had defined a multitarget application (MTA) project (named **statmathWithDB**) with two modules in it:

- An SAP HANA database module (**statmathDB**)
- A Java Web application module (**java**) as a Tomcat application



3 INSTALLING AND IMPORTING THE MATLAB LIBRARIES

3.1 Installing the MATLAB Compiler Runtime

With the installation of the MCR, you install the native libraries and obtain the Java class libraries. From the MathWorks Web site (https://de.mathworks.com/products/compiler/mcr.html), we downloaded the release **R2017b (9.3)** for Linux 64bit (MATLAB version 2017b).

Note: You get a version that corresponds to the Java Development Kit (JDK) version of Tomcat in SAP HANA extended application services, advanced model. Only 2017b runs on JDK 1.8, which is used by the current Tomcat delivery in the advanced model.

We unzipped the downloaded file and installed the runtime into a directory. Please keep in mind that we had access to the underlying Linux file system. We issued the `install` command in this directory:

```
./install –mode silent –agreeToLicence yes –destinationFolder /usr/sap/VH1/MCR/
```

This creates the folder MCR and installs all the needed components of MATLAB. All these steps are performed with the `<sid>adm` user. This is the Linux user, under which SAP HANA extended application services, advanced model and SAP HANA are running.

3.2 Deploying the MATLAB Libraries

We basically had two sets of libraries to deploy: native libraries and Java class libraries.

The native libraries are deployed through a system environment variable.

The Java class libraries are deployed through SAP Web IDE. As previously mentioned, SAP Web IDE uses Maven to manage third-party libraries as well as custom application code. The issue we had was that the core MATLAB Java libraries were not Maven-compliant. This meant that at least three didn't have required Maven properties. But you can "attach" those properties to the JARs and then import them into SAP Web IDE.

3.2.1 Setting Up the LD_LIBRARY_PATH Environment Variable

We set the LD_LIBRARY_PATH environment variable (as `sid<adm>`) to the installed MCR:

```
LD_LIBRARY_PATH:
/usr/sap/VH1/MCR/v93/runtime/glnx64:/usr/sap/VH1/MCR/v93/bin/glnxa64:/usr/sap/VH
1/MCR/v93/sys/os/glnxa64
```

Now we needed to set a configuration of SAP HANA extended application services, advanced model to this environment variable, so that the build and deployment service of the advanced model can bind this variable to the corresponding application. This can be done in two different ways:

- You can use a command in the command line interface (CLI) of the advanced model for the corresponding application. The drawback of this approach is that this has to be done for each redeployment, which is quite unfavorable during development.
- A much better approach is setting a property (the LD_LIBRARY_PATH) in the application's `mta.yaml`. Here is the relevant excerpt from this file:

```
ID: statmathWithDB
_schema-version: '2.0'
version: 0.0.1

modules:
 - name: statmathDB
   type: hdb
   path: statmathDB
```

```
   requires:
    - name: hdi-container

 - name: java
   type: java.tomcat
   path: java
   properties:
      LD_LIBRARY_PATH:
/usr/sap/VH1/MCR/v93/runtime/glnxa64:/usr/sap/VH1/MCR/v93/bin/glnxa64:/usr/sap/VH1/MCR/v9
3/sys/os/glnxa64
      JBP_CONFIG_RESOURCE_CONFIGURATION: "['tomcat/webapps/ROOT/META-INF/context.xml' :
{'service_name_for_DefaultDB' : '~{hdi-container-name}'}]"
...
```

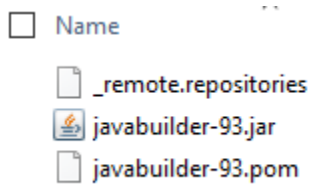
3.2.2 Attaching Maven Properties to the MATLAB Java Class Libraries

A prerequisite for this step is to have a Maven installation on any machine in your control. We used Maven 3.5.0 on a local machine. We then took our MATLAB Java class libraries and issued the command. Here is an example for the `javabuilder.jar`):

```
mvn install:install-file -Dfile=<PATH_TO_YOUR_JAR>/javabuilder.jar -
DgroupId=mathworks -DartifactId=javabuilder -Dversion=93 -Dpackaging=jar
```

This way the `javabuilder.jar` gets the properties (and the values) **groupId=mathworks**, **artifactId=javabuilder** and **version=93** attached. The local Maven installation has a local repository, where you will find the Maven artifacts of the `javabuilder` class library file. Here are some details – exemplary for the `javabuilder.jar`:

In a standard installation for Windows, the repository usually resides under `C:\Users\<user>\.m2\repository`. There you will find a folder structure like `~/<groupId>/<artufactId>/<version>` (in this case `~/mathworks/javabuilder/93`). Under that resides the “Maven compliant” `javabuilder jar` file as well the corresponding `pom.xml`.



Note: You do the same with all JARs you want to import in SAP Web IDE.

3.2.3 Importing the MATLAB Java Class Libraries into the SAP Web IDE Project

The import of the (now Maven-compliant) JARs into the SAP Web IDE project is now easy. The only thing you still need to take care of is the definition of a “local” Maven repository.

Note: Usually the dependencies are being checked against a repository, which in most cases is available online. In this case, it was sufficient to have a local repository, which we have built, simply by defining a folder structure in SAP Web IDE (which is then persisted on the machine running SAP HANA extended application services, advanced model).

In the `pom.xm` of the Java module, we defined the local repository to:

- Reside directly under the project directory of the Java module (this is the property `project.basedir` in the URL element)

```
<repositories>
  <repository>
    <id>lib</id>
    <url>file:${project.basedir}/lib</url>
  </repository>
</repositories>
```

- Set its name to `lib`

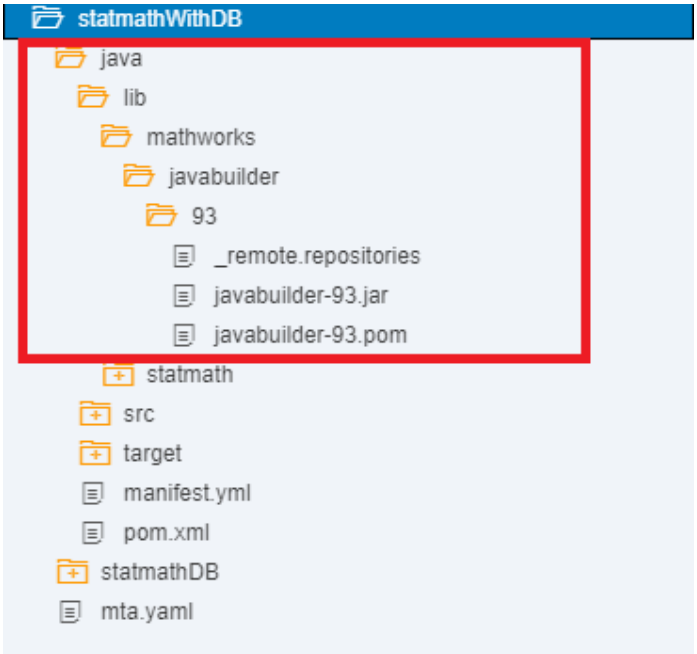
At the end, the project structure in SAP Web IDE looks like the screenshot to the right.

You can see the whole structure of the lib folder/directory for the class library `javabuilder`.

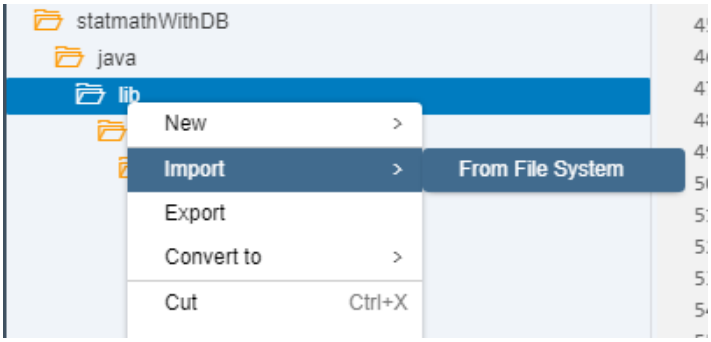
Important note: You have to manually create the folder structure for each JAR (class library).

Please keep in mind that:

- **mathworks** is the groupId.
- **javabuilder** is the artifactId.
- **93** is the version.



Finally, you can import the artifacts (the actual JAR file along with its `pom.xml` and the Maven-specific file `_remote_repositories`) into the project folder/directory `lib` using the import wizard of SAP Web IDE.



As a last step in this process, you have to declare the dependencies of your imported MCR JARs in the “dependencies” section of the `pom.xml`.

Note: Please keep in mind that in the screenshot to the right, you see the result of a process not only for the JAR file `javabuilder` but also for other non-MCR JARs.

```
18  <dependencies>
19    <dependency>
20      <groupId>mathworks</groupId>
21      <artifactId>javabuilder</artifactId>
22      <version>93</version>
23    </dependency>
24    <dependency>
25      <groupId>statmath</groupId>
26      <artifactId>matlabExample</artifactId>
27      <version>1</version>
28    </dependency>
29    <dependency>
30      <groupId>statmath</groupId>
31      <artifactId>statmathVar</artifactId>
32      <version>1</version>
33    </dependency>
34    <dependency>
35      <groupId>statmath</groupId>
36      <artifactId>statmathVarToolbox</artifactId>
37      <version>1</version>
38    </dependency>
```

4 BUILD AND DEPLOYMENT

You can finally build the MTA archive and deploy it to the on-premise instance of SAP HANA extended application services, advanced model. The Java class libraries are then automatically deployed, since they are part of the MTA archive.

5 RESULTING SOLUTION ARCHITECTURE

At the end, we have the MCR deployed as a Java module running in SAP HANA extended application services, advanced model. The Java module is of type `java.tomcat` (see the `mta.yaml` excerpt above).

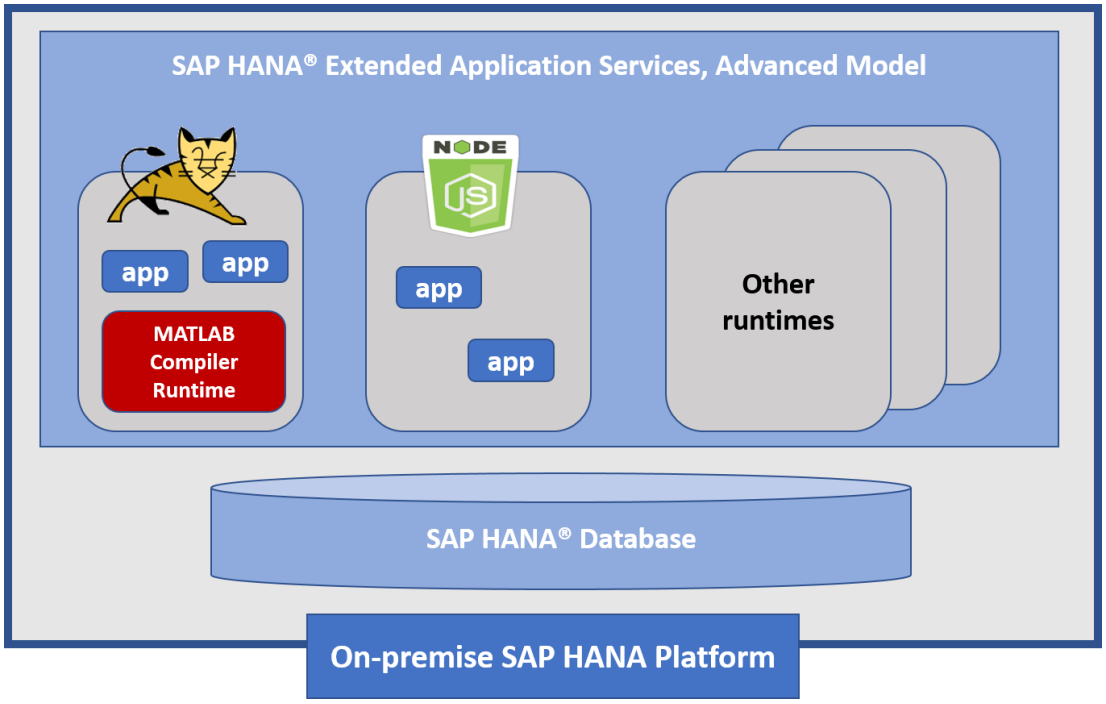


Figure 1: Multitarget Application (MTA) on SAP HANA® Extended Application Services, Advanced Model

6 CONCLUSION AND FINAL REMARKS

The main goal of the technical proof of concept was to evaluate the options of integrating the MATLAB Compiler Runtime with SAP HANA. The reason for this move is obvious: if you can tightly integrate such a powerful data management platform like SAP HANA with a just as powerful data science platform like MATLAB, the result can be quite useful for both communities and beyond. Our approach was to deploy some basic MATLAB libraries (the MATLAB Compiler Runtime) as Java-class libraries on the Tomcat server of SAP HANA extended application services, advanced model, running on the very same box as the SAP HANA database – as a framework so to speak. You can deploy any arbitrary solution based on MCR on top of this foundation. You can also build custom solutions – for example, **SAP HANA native solutions** running on the engine of the advanced model using the MCR.

From the perspective of a data center operator, this option is very maintenance friendly. Once packaged and in the corresponding release, an admin can take the “MCR mtar” and deploy it to many arbitrary SAP HANA instances by performing minor configuration steps (basically following the steps described in sections 2 and 3 of this document). The release cycle of SAP HANA extended application services, advanced model (especially in regards to JDK and Tomcat) is not in sync with the release cycle of MATLAB’s MCR. This has to be taken into account.

You can develop a native custom SAP HANA solution using the MCR API, since the MCR is yet another module in SAP HANA extended application services, advanced model. From a developer’s perspective, such an application simply uses a module available on the advanced model.

Final remark: Generally speaking, we have described the process of integrating Java native and class libraries in a Java module running on the Tomcat server of SAP HANA extended application services, advanced model. This concept is not MATLAB-specific but of rather a universal nature. This means that you can apply this concept to any arbitrary set of native and especially Java-class libraries.

www.sap.com/contactsap