IBM Business Automation Community

Come for answers. Stay for best practices. All we're missing is you.

Ask a question

Search for your f...

Fuel your AI at the ultimate IBM learning event

IBM TechXchange Conference October 21-24, 2024 Mandalay Bay - Las Vegas

Register now

Open Editions

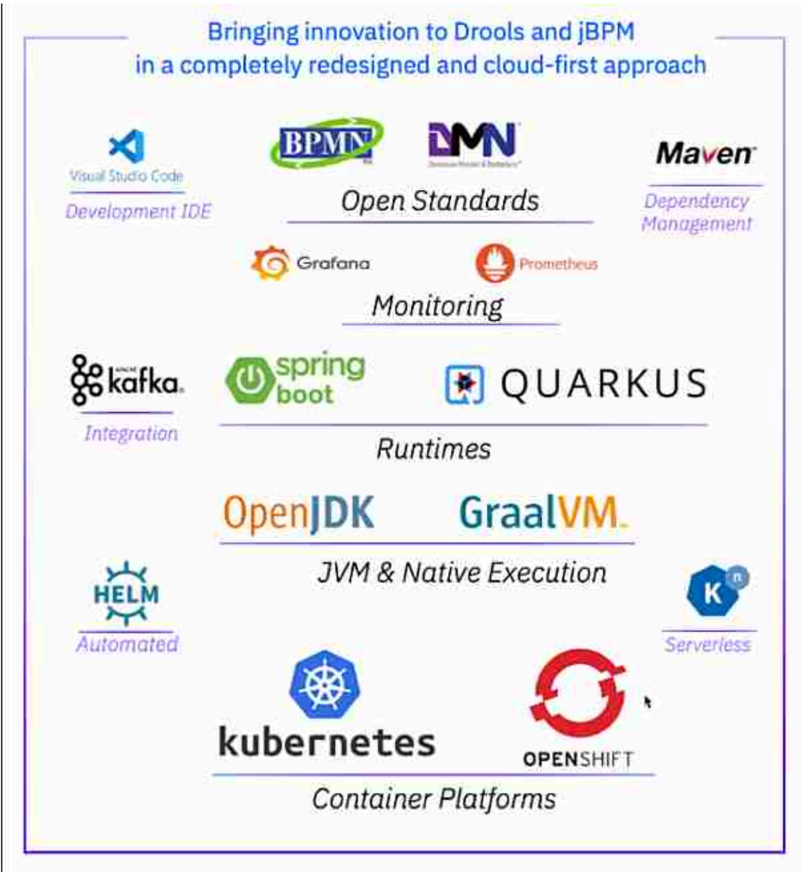
View Only

Quick Workstation Setup for BAMOE 9.0 and getting ready for 9.1

By Tim Wuthenow posted Thu June 06, 2024 04:49 PM

5 Like

Time flies when you're having fun! For me, it's hard to believe it has been almost 2 full years since IBM officially took over Red Hat Process Automation Manager and Red Hat Decision Manager. In that time, we released the initial version of the Enterprise-supported version called IBM Business Automation Manager Open Editions 8.0, designed to be the quick shift for customers with the Red Hat offerings. A year ago, we released version 9 and with it, matching the community direction, a full focus on the cloud native architecture created with the Kogito project. Over the next few weeks I am going to be releasing a series of blogs that cover some of the features in Business Automation Manager Open Editions (BAMOE), but to first do that, let's get our workstation configured so we can run with all of these fun new features!



Cookie Preferences

What are the current requirements for Open Editions?

Great question, there are detailed requirements that I will include in the links to this blog with [the documentation for 9.0.1 here](https://www.ibm.com/docs/en/ibamoe/9.0.x?topic=notes-bamoe-release-version-901#_system_requirements), but my biggest thing is trying to get started with something in a few minutes. So let's save reading some other documentation (though always recommended!), and let's get our environment setup to deal with both the current version, and the very near future so you can easily swap your developer environment from one version to another.

Java - the first big change coming soon

Currently in both version 8.0 and 9.0, the supported version of Java is version 11. In version 9.1, this is going to jump to version 17. The shift is going to bring a new level of Long Term Support in Java, but also more importantly more efficient development, innovation and API enhancements to make our processes and decisions jump into the future. (**Update 6/18/2024: Changed the Java version to the IBM Semeru versions of Java 11 and 17**)

Base Environment

With Java 11, being a former Red Hatter in IBM, I typically find myself using OpenJDK 11, there are several supported flavors, so choose the one that makes the most sense (and supported for your subscription). I have currently found that a combination of jenv and SDKMAN! have been the best way for me to maintain multiple development setups. If you want to emulate my environment on a Mac, it's pretty easy to get setup using the following commands:

1. Install SDKMAN!

```
curl -s "https://get.sdkman.io" | bash
```

2. Initialize SDKMAN! Close and reopen your terminal, then run:

```
source "$HOME/.sdkman/bin/sdkman-init.sh
```

3. Install Java 11 and Maven 3.8.6 with SDKMAN!

```
sdk install java 11.0.23-sem
sdk install maven 3.8.6
```

4. Install Java 17 and Maven 3.9.7 with SDKMAN!

Cookie Preferences

```
sdk install java 17.0.11-sem
sdk install maven 3.9.7
```

5. Install jenv to help maintain and swap between different Java environments easily.

```
    brew install jenv
```

If you don't have Homebrew installed, first install Homebrew by following the instructions at https://brew.sh

6. Initialize jenv

Add `jenv` to your shell profile. For `zsh`, add the following lines to `~/.zshrc`:

```
 export PATH="$HOME/.jenv/bin:$PATH" eval "$(jenv init -)"
```

For `bash`, add the same lines to `~/.bashrc` or `~/.bash_profile`.

Then, source the file:

```
    source ~/.zshrc  # or source ~/.bashrc
```

7. Add Java versions to `jenv`

```
jenv add $(sdk home java 11.0.23-sem)
jenv add $(sdk home java 17.0.11-sem)
```

8. Set up aliases for switching Java versions and Maven by adding the following aliases to your `~/.zshrc` file (or `~/.bashrc` for bash users):

```
alias refreshSH="source ~/.zshrc"
alias switchJava17="jenv global semeru64-17.0.11 && refreshSH && sdk use maven 3.9.7"
alias switchJava11="jenv global semeru64-11.0.23 && refreshSH && sdk use maven 3.8.6"
```

9. Source your shell configuration
You can do this the standard way first:

```
    source ~/.zshrc  # or source ~/.bashrc
```

After this refresh is done, you will be able to use your new short cut which will refresh the `.zshrc` or `.bashrc` with a shorter command (and one that is used in your *switchJava* commands!

```
    refreshSH
```

10. The last thing to run so your environments more gracefully switch between versions of Maven and Java between the two environments is to enable the jenv plugin for Maven. To do this you would run the following command (this only needs to be done once and is not needed in say your ~/.zshrc file):

```
    jenv enable-plugin maven
```

For your Maven settings.xml, you should be able to use the one I will include in this repository of example files for your workstation setup. The settings.xml just refers to the Red Hat Maven GA for when I want to use things like their version of OpenJDK, Quarkus, etc and the Maven Central repository for pulling things from other Open Source Communities and where IBM publishes our builds of IBM Business Automation Manager Open Editions.

Container Environment

The evolution of jBPM and Drools with the Kogito open-source project to push process and decisions into Cloud Native workloads comes best when deployed as a container. While the services you build will ultimately be able to run as `java -jar` entities; they're much better as a container. In my local environment, I just use `colima` and `docker` together, but choose as you may. One of the main things to note with Kogito is that the solution, when running `mvn quarkus:dev` you will need to use testcontainers through the docker environment, the properties for testcontainers are found at `~/.testcontainers.properties` and the settings I use are below:

```
#Modified by Testcontainers

docker.client.strategy=org.testcontainers.dockerclient.UnixSocketClientProviderStra

testcontainers.reuse.enable=true

ryuk.container.privileged=true
```

Since I am using colima and not Docker Desktop (or other container solutions), I also had to set the environment variable:

Cookie Preferences

```
export DOCKER_HOST=unix:///Users/timwuthenow/.colima/default/docker.sock
```

The last thing t be sure of with your container environment is that you have the capability to use `docker compose`. There are a variety of ways to do this, so this guide won't go into those in detail.

Using VS Code and the BAMOE Extensions and BAMOE Canvas

The community direction with the tools associated with BAMOE are through a combination of using built for purpose extensions for VSCode or through the Enterprise version of KIE Sandbox called BAMOE Canvas. Canvas offers a web client based hosting of the editors for DMN and BPMN with a DMN runner to test your decisions in real time.

For detailed instructions for the tools, please refer to [the official BAMOE documentation](https://www.ibm.com/docs/en/ibamoe/9.0.x?topic=installation) for the installation instructions for either set of tools. If you want to deploy Canvas on OpenShift, the instructions can be ran as a shell script found with my repositories [here](https://github.com/timwuthenow/bamoe-canvas-quick-deploy)and if you are looking at building Docker environments locally, I have a few different methods found within this [repository](https://github.com/timwuthenow/ibamoe-docker-compose).Let's do some quick examples to validate these changes!

First let's do something in BAMOE 9.0, which will be Quarkus 2, Java 11 and using DMN just as a quick getting started. Let's use our new command to start our terminal in Java 11 with Maven 3.8.6:

```
switchJava11
```

When you run this, you can validate your environment reflects what you expected.

Cookie Preferences

```
timwuthenow@Tims-Work-MacBook-Pro-4 ~ % switchJava11

Using maven version 3.8.6 in this shell.
timwuthenow@Tims-Work-MacBook-Pro-4 ~ % mvn --version
Apache Maven 3.8.6 (84538c9988a25aec085021c365c560670ad80f63)
Maven home: /Users/timwuthenow/.sdkman/candidates/maven/3.8.6
Java version: 11.0.23, vendor: IBM Corporation, runtime:
/Users/timwuthenow/.sdkman/candidates/java/11.0.23-sem
Default locale: en_US, platform encoding: UTF-8
OS name: "mac os x", version: "14.5", arch: "aarch64", family: "mac"
timwuthenow@Tims-Work-MacBook-Pro-4 ~ % java --version
openjdk 11.0.23 2024-04-16
IBM Semeru Runtime Open Edition 11.0.23.0 (build 11.0.23+9)
Eclipse OpenJ9 VM 11.0.23.0 (build openj9-0.44.0, JRE 11 Mac OS X aarch64-64-Bit
20240522_620 (JIT enabled, AOT enabled)
OpenJ9   - b0699311c7
OMR      - 254af5a04
JCL      - 3d1045c3ea based on jdk-11.0.23+9)
```

After this, you can run the commands to create a new Kogito project:

```
mvn io.quarkus:quarkus-maven-plugin:2.16.7.Final:create \
    -DprojectGroupId=com.ibm \
    -DprojectArtifactId=first-kogito \
    -DprojectVersion=1.0.0-SNAPSHOT \
    -DplatformVersion=2.16.7.Final \
    -DplatformGroupId=io.quarkus.platform \
    -DplatformArtifactId=quarkus-bom \
    -Dextensions=kogito-quarkus,dmn,resteasy-reactive-jackson,quarkus-smallrye-
openapi,quarkus-smallrye-health
```

Your output should be similar to the below:

```
[**INFO**] Scanning for projects...
[**INFO**]
[**INFO**] **------------------< org.apache.maven:standalone-pom >----------
---------**
[**INFO**] **Building Maven Stub Project (No POM) 1**
[**INFO**] **--------------------------------[ pom ]---------------------------
-----**
[**INFO**]
[**INFO**] **---** quarkus-maven-plugin:2.16.7.Final:create **(default-cli)** @
standalone-pom **---**
[**INFO**] -----------
[**INFO**] selected extensions:
- org.kie.kogito:kogito-quarkus-decisions
- io.quarkus:quarkus-smallrye-openapi
- io.quarkus:quarkus-smallrye-health
- io.quarkus:quarkus-resteasy-reactive-jackson
- org.kie.kogito:kogito-quarkus

[**INFO**]

applying codestarts...

[**INFO**] 📚  java

🔨  maven

📦  quarkus

📝  config-properties

🔧  dockerfiles

🔧  maven-wrapper

🚀  kogito-dmn-codestart

🚀  resteasy-reactive-codestart

🚀  smallrye-health-codestart

[**INFO**]
-----------
[SUCCESS] ✅  quarkus project has been successfully generated in:
--> /Users/timwuthenow/process-future/first-kogito
-----------
[**INFO**]
```

```
[**INFO**]
==========================================================================

[**INFO**] Your new application has been created in
**/Users/timwuthenow/process-future/first-kogito**
[**INFO**] Navigate into this directory and launch your application with **mvn
quarkus:dev**
[**INFO**] Your application will be accessible on **http://localhost:8080**
[**INFO**]
==========================================================================
[**INFO**]
[**INFO**] **------------------------------------------------------------------
-----**
[**INFO**] **BUILD SUCCESS**
[**INFO**] **------------------------------------------------------------------
-----**
[**INFO**] Total time:  0.953 s
[**INFO**] Finished at: 2024-06-05T09:42:04-04:00
[**INFO**] **------------------------------------------------------------------
-----**
```

You can open your project by changing to the new directory and then opening the directory in VS Code.

```
cd first-kogito
code .
```

This will open your VSCode editor (if you have the shortcut to do so otherwise, you need to open VSCode, press `Command Shift and P` on Mac, type shell and select `Shell Command: Install 'code' command in PATH`, Windows should do this automatically in the installation, Linux will depend on the distribution).

Within the project itself, the command will create a Quarkus project with a DMN, endpoints, and some more components. If you want to test out the service, you should be able to run the sample service as long as you have a Docker environment actively running based on the previous sections (especially the DOCKER_HOST variable otherwise it will fail to start). With that said, let's go ahead and start this service using our new environment!

```
mvn quarkus:dev
```

After a short bit, your application will start and will be available at http://localhost:8080/q/dev for the development console to take a browse around. With this sample application, you can browse things like the Swagger-UI for the various endpoints that are autogenerated from the DMN model and more.

Cookie Preferences

If you have any issues with the build/start, validate the following:
1. Make sure the Java you're currently running is 11 - `java --version` should yield a Java 11 version if your commands are setup appropriately on your Mac (I will write a blog soon on Windows and Linux).
2. This also matters on the Maven version, validate that when you're running the sample command above, you're using Maven 3.8.6. The `switchJava11` command I provided should also show the Maven version that is being used to provide extra reassurance.
3. Make sure the `DOCKER_HOST` variable is setup, this was the one that I don't always put in my `~/.zshrc` because I swap environments a lot, but might make sense for you. If you get an error complaining about Docker environment not available, this is more than likely the reason *if you have a Docker environment installed and running*. If you're worried before you start if Docker is running or not, a quick `docker run hello-world` is always the best bet.

The next entries we will be doing much more cooler integrations and tests, but I wanted to be sure that the base layer is there for you. The next blog will be connecting your local environment to an IBM Event Streams instance and testing a DMN decision. After that, we will be showing process connected to Event Streams, so come back often! If you can't wait until next time, feel free to use some of the exercises at https://bamoe.university to gain more experience in the mean time!

1 comment **35 views**

Permalink

https://community.ibm.com/community/user/automation/blogs/tim-wuthenow/2024/06/(

Comments

Naresh Thakur Wed July 03, 2024 01:34 AM

NT

Dear Tim

I hope this message finds you well. I have been following your blog for some time now and have found your guides incredibly helpful. Currently, I am facing challenges setting up IBM BAM OE (Business Automation Workflow) 9 or 8 on my Windows laptop. Despite my best efforts over the past week, I have a low code no code background and it has made this task quite daunting.

Given your expertise in the field, I would greatly appreciate it if you could post a detailed **guide on installing IBM BAM OE 9 or 8 for Windows**, tailored for beginners like myself. Any tips, step-by-step instructions, or best practices you

could share would be immensely valuable as I work on my current project, which heavily relies on building workflows with IBM BAM OE.
Thank you in advance for considering my request. Your assistance would not only help me but also many others who might be in a similar situation.

Cheers

Naresh