What is a Maven Repository?

medium.com/@chris.shellenbarger/what-is-a-maven-repository-6fa22991fad9

If you've just joined the software engineering workforce at a Java shop, or have recently become a Java developer, you may be asking yourself, "What is Maven? Why do I need it?".

Asking your co-workers, they might respond with something along the lines of, "A Maven Repository is where we store all of our artifacts". Artifacts? Are we some type of archaeologist?

This article assumes you're relatively new to the professional world of the Java ecosystem. It will answer the following questions which will surely arise in your first couple of weeks at your new job:

- What is an Artifact?
- What is Maven?
- What is a Maven Repository?
- What about Private Maven Repositories?

Jump ahead to the question that is most relevant to you, or read the entire article to gain the most knowledge. Either way, we hope this article will provide you with a helpful intro to the world of Artifacts and Maven.

Before we get started, we'd like to make clear other assumptions that hold for the remainder of this article:

- We are discussing Java software development practices. All references to anything software related will refer to the Java world.
- Maven, Artifacts, and other related topics discussed here are used by other programming languages that run on the Java Virtual Machine (JVM), such as Scala (Simple Build Tool — SBT), Groovy (Ant & Ivy), and Clojure (Leiningen & Boot). The concepts discussed here apply similarly to these technologies.

What is an Artifact?

In Java, an artifact can be any type of file that is used in the software development process. The most common of these are Java libraries, also known as ‘JAR files’. Software distribution files, packages, documentation bundles, machine learning models, and any other type of file you can think of can all be artifacts in the Java world.

Artifacts are used in a Java program for many different purposes. When a JAR file artifact is used at compile time it is typically used to bring in Java library code so that code can be reused.

Some artifacts may be packaged with the software and not used until runtime. These can include artifacts which hold data of some sort: images, machine learning models, documentation, language packs, etc.

TLDR: Artifacts are files used by Java programs. They can also be Java programs in the case of self executing archives or other type of executable Java file.

The most common type of artifact you’ll encounter is a dependency—ie. a java library. This brings us to your new friend (and our old friend), Maven.

What is Maven?

According to the Maven Homepage: “Apache Maven is a software project management and comprehension tool.”.

Hmm? Yeah, that’s not really clear to someone who’s just getting started.

Let’s take a quick minute and discuss what the Java development process is when you don’t have Maven:

Let’s say you start a new project for your Facebook-disrupting new app. You open your editor and start writing your Java program. You come to a part in the code where you want to insert data into a database so you have two choice:

- Research the database protocol and write code that talks directly to the database, or..
- Find a library which already has implemented the database

connectivity.

Which one will you choose? If you have any hope of shipping that new disruptive app, you had better use the library. Why? Because writing database code is non-trivial and will suck down most of your energy before you even get to your actual app's code.

Software Engineering 101: "Don't Reinvent the Wheel"—ie. find a reliable library and use it. One of the reasons why Java is popular today is because of the wealth of existing libraries that exist for you to use today. Other than app specific business logic, most utility code that you will need has been written and is waiting for you to use—you just have to find it!

Okay, so once we find a library that contains the code we need (how you do this could be a completely different article), how do we add it to our program? In Java, we can add libraries to our programs by downloading and adding the JAR files to the Java Classpath. If you're using an Integrated Development Environment (IDE) the GUI will guide you. If you're a hard core glutton for punishment and are using a basic text editor and using the command line javac and java commands to run your program, you'll need to add the -cp or -classpath argument to your invocations.

If you only have to add a single library to your classpath it might not be such a big deal, but what if that library you are using requires yet another library (and that one requires another, and so on and so on). You can quickly end up in what is referred to as Maven Dependency Hell where you will have to download and specify dozens, if not hundreds of different libraries.

It's madness, and it sucks, and it's something that we used to have to do all the time for all of our projects before we started to use Maven.

Using Maven, you no longer directly manipulate the classpath or download jars. Using an XML configuration file, known as a Project Object Model (POM) or POM file, you specify the dependencies your project needs and then let Maven do the rest. When Maven runs, it will look at the list of declared dependencies and download all of them, including any dependencies that may be implicitly needed, also known as ‘transitive dependencies’.

Once Maven has completed, you’ll be able to run your program (with Maven or through an IDE) and your classpath will include all the Jars that Maven downloaded for you. It’s really simple, relatively straightforward (if you can get past the XML verbosity), and has helped many development teams manage their dependencies in a declarative, repeatable manner (in the old days we’d check in dependencies to version control, yikes!).

So, that’s Maven….well, one of the most commonly used things that it does. Maven can also build your project, bundle up your application, publish it, and do many different things all driven by various plugins that have been written over the years. Just like java libraries, there is usually a Maven Plugin available for anything you want to do, you just have to find it!

If you were paying attention, you might have wondered where Maven downloads all of these dependencies from. Well, the answer is quite simple: Maven Repositories.

Just like artifacts, Maven Repositories can be called by many different names: Maven Artifact Repositories, Maven Package Repositories, Maven Package Managers, Maven Repository Managers, Binary Repositories, the list goes on and on!

Maven Repositories are web servers which provide simple HTTP endpoints which allow GET and PUT requests for publishing and retrieving Maven Artifacts by Maven itself. That might seem like it’s a little bit too technical, but we’re all software engineers here after all and HTTP is something we all are familiar with in 2018, we hope!

Java is known for its wealth of open source libraries and most of these libraries are available through Maven Repositories. In particular, the big massive Maven Repository that holds most of the world’s open source artifacts is call the Maven Central Repository.

Maven is configured to check the Maven Central Repository by default so you won't have to configure your POM files to retrieve them — simply declare your open source dependencies and the Maven command line will take care of the rest!

What about Private Maven Repositories?

We've covered the Maven Central Repository, the place where Maven pulls its publicly available, open source dependencies from, but what about the dependencies that contain our company's proprietary, private code?

This is where Private Maven Repositories come in.

Private Maven Repositories are just like any other Maven Repository except that they contain a company's private artifacts.

Typically, a Private Maven Repository will implement access controls, or will be isolated on an internal network, in order to prevent people outside of the company from accessing private artifacts. Historically, most private maven repositories have been hosted inside a company's data center or firewall, however with everything moving to the cloud, new cloud based maven repository manager have been developed.

Private Maven Repositories are not exclusively for private artifacts. They can also be used by companies that wish to publish certain artifacts to the public but wish to maintain control over the distribution of these artifacts.

How does CloudRepo fit in?

CloudRepo provides both Private and Public Maven Repositories built on a cloud based, highly available architecture. CloudRepo provides access controls so that you can restrict access to only specific users but also allows for publicly available repositories so that anyone in the public can connect and download artifacts.

CloudRepo also provides the ability to search across your Maven Repositories for any of your Maven Artifacts. A robust user portal is also available for browsing the contents of your maven repositories.