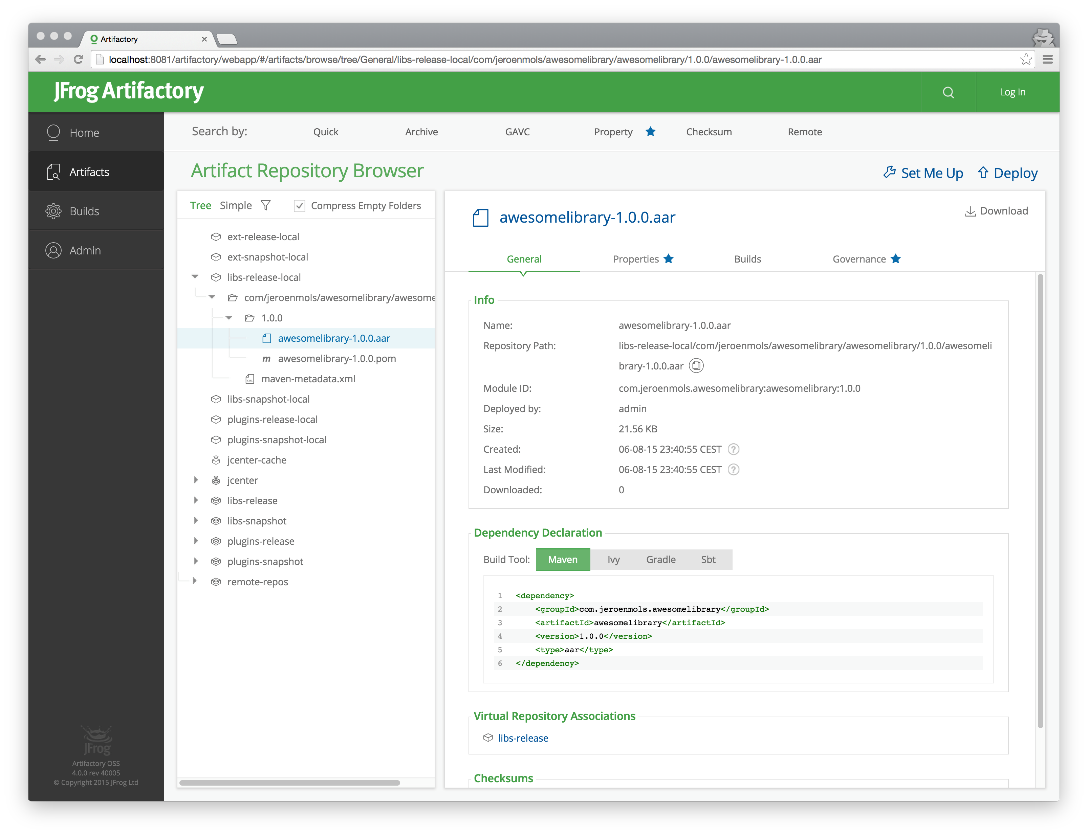
# Deploying Shared Components

## Vendor-agnostic Deploying Shared Components

Managing shared components across projects requires for a solid shared component management solution. As a best practice, it is a function of organization’s IT department to maintain a private shared component repository of all approved and certified third-party and in-house-developed dependencies and components. There are various shared component management platforms on the market, but commonly used are JFrog Artifactory or Maven private repository for Java and NodeJS projects. For Microsoft .NET projects, Team Foundation Server provides ability to manage shared components and integrate into any Visual Studio.NET project via NuGet or similar package management add-ons.

To deploy a shared component, first the component needs to be properly packaged into a format readable by the dependency management system. In Java, that format is a .jar file of a shared component. In Microsoft.NET, it is an assembly .dll file. Finally, in NodeJS, it is a NodeJS folder ready with its package.json and enabled for importing into the dependency management system.

Once the shared component is properly packaged, it is ready for deployment into a dependency management platform. There are two ways to deploy, depending on whether there is a need to auto-deploy updated modules via a CI/CD pipeline script or whether the deployment is done manually. For manual deployments, there is usually a graphical interface like the one shown below for JFrog Artifactory. The “Deploy” link shown on the top-right-hand corner of the interact under “Artifacts” tab allows an Artifactory admin to deploy a given module or component with a new version.



The same dependency management systems always have a command-line interface (CLI), which can be used either manually to deploy new shared components or triggered via CI/CD stage script. As an example, a new shared component version is scripted via Jenkins to automatically get build and then deployed into Artifactory via CLI.

Once the new version of the shared component is deployed into the dependency management system, it is recommended for the owner of the change to publish new release notes via Readme.md or similar into the source code repository of that shared component, and to manually notify other developers on the team that a new shared component version is now available for consumption.

At this point, a shared component new version is published into Artifactory or similar dependency management system, the code is checked into the source code repository, and the release notes are updated for the new version of the component.

## MuleSoft-specific Shared Component Deployment

#### Shared API Commons Library

Shared libraries are deployed similarly to how Java shared component JAR files are deployed. A packaged Mule shared library JAR file is deployed to one of these target locations to become available to other projects for use:

* Mulesoft Exchange
* Maven private repository
* Artifactory or similar dependency management repository

Once the shared library is packaged and deployed to the dependency management platform or to Exchange, it becomes available to be added as a dependency in Maven pom.xml of any Mule application project.

#### Shared Connectors and Modules

Once the connector is created and published to Exchange, any Mule project that needs to leverage this connector can now add it to the project, by reviewing the list of connectors on the organization’s Exchange dashboard. Here is a image of the Exchange dashboard listing default and custom connectors:

