Sonatype

Introduction to Repository Management

What is a Repository Manager?

Introduction

While many developers have adopted Maven, Ant, Ivy and Gradle for their builds, many have yet to understand the importance of maintaining a repository manager both to proxy remote repositories and to manage and distribute software artifacts. Repository managers are a foundational step in a broader trend towards component lifecycle management (CLM). CLM generally, and repository managers specifically, help organizations gain control and improve collaboration in agile, component-based software development.

This document defines repository and repository management, providing context for developers interested in learning how to use Nexus Professional to achieve a more efficient development cycle.

What is a Repository?

Java developers are familiar with the concept of a repository: a collection of binary software artifacts and metadata stored in a defined directory structure, which is used by clients such Apache Maven[1] or Ivy to retrieve binaries during a build process. In the case of the Maven repository, the primary type of binary artifact is a JAR file containing Java bytecode, but there is no limit to what type of artifact can be stored in a Maven repository. For example, one could just as easily deploy documentation archives, source archives, or Ruby libraries to a Maven repository. A Maven repository provides a platform for the storage, retrieval, and management of binary software artifacts and metadata. Maven Repositories are used by virtually all Java development tools, not just Maven.

In Maven, every software artifact is described by an XML document called a Project Object Model (POM). This POM contains information that describes a project and lists a project's dependencies - the binary software artifacts which a given component depends upon for successful compilation or execution.

When Maven downloads a dependency from a repository, it also downloads that dependency's POM. Given a dependency's POM, Maven can then download any other libraries that are required by that dependency. The ability to automatically calculate a project's dependencies and transitive dependencies is made possible by the standard and structure set by the Maven repository.

Build tools such as Maven and Ivy interact with a repository to search for binary software artifacts, model the projects they manage, and retrieve software artifacts on-demand from a repository. When you download and install Maven without any customization, Maven will retrieve artifacts from the Central Repository, which serves millions of Java developers every single day. While you can configure Maven to retrieve binary software artifacts from a collection of mirrors, the best practice is to install a repository manager such as Sonatype Nexus Pro, which can proxy the Central Repository and cache artifacts retrieved from a remote repository on a server in your own network. In addition to the Central Repository, there are a number of major organizations such as Redhat, Oracle, and Codehaus that maintain separate repositories.

[1] Apache Maven is a trademark of the Apache Foundation

While this might seem like a simple, obvious mechanism for distributing artifacts, the Java platform existed for several years before the Maven project created a formal attempt at the first repository for Java artifacts. Until the advent of the Maven repository in 2002, a project's dependencies were gathered in a manual, ad-hoc process and were often distributed with the source code for an open source project. As applications grew more and more complex, and as software teams developed a need for more complex dependency management capabilities for larger enterprise applications, Maven's ability to automatically retrieve dependencies and model dependencies between components became an essential part of software development.

Release and Snapshot Repositories

A repository stores two types of artifacts: releases and snapshots. Release repositories are for stable, static release artifacts and snapshot repositories are frequently updated repositories that store binary software artifacts from projects under constant development. While it is possible to create a repository that serves both release and snapshot artifacts, repositories are usually segmented into release or snapshot repositories serving different consumers and maintaining different standards and procedures for deploying artifacts. Much like the difference between a production network and a staging network, a release repository is considered a production network and a snapshot repository is more like a development or a testing network. While there is a higher level of procedure and ceremony associated with deploying to a release repository, snapshot artifacts can be deployed and changed frequently without regard for stability and repeatability concerns.

Release Artifacts

- A release artifact is an artifact that was created by a specific, versioned release. For example, consider the 1.2.0 release of the commons-lang library stored in the Central Repository. This release artifact, commons-lang-1.2.0.jar, and the associated POM, commons-lang-1.2.0.pom, are static objects that will never change in the Central Repository. Released artifacts are considered to be solid, stable, and perpetual in order to guarantee that builds which depend upon them are solid and repeatable over time. The released JAR artifact is associated with a PGP signature, an MD5 and SHA checksum, which can be used to verify both the authenticity and integrity of the binary software artifact.

Snapshot Artifacts

- Snapshot artifacts are artifacts generated during the development of a software project. A snapshot artifact has both a version number such as "1.3.0" or "1.3" and a timestamp in its name. For example, a snapshot artifact for commons-lang 1.3.0 might have the name commons-lang-1.3.0-20090314.182342-1.jar. The associated POM, MD5 and SHA hashes would also have a similar name. To facilitate collaboration during the development of software components, Maven and other clients interrogate the metadata associated with a snapshot artifact and always retrieve the latest version of a snapshot dependency from a repository.

Repository Coordinates

Repositories and tools like Maven know about a set of coordinates which identify each object, including the following components: groupId, artifactId, version, and packaging. This set of coordinates is often referred to as a GAV coordinate, which is short for "Group, Artifact, Version coordinate". The GAV coordinate standard is the foundation for Maven's ability to manage dependencies. Four elements of this coordinate are described below:

Group Identifier (groupId)

- A group identifier groups a set of artifacts into a logical group. Groups are often designed to reflect the organization under which a particular software component is being produced. For example, software components being produced by the Maven project at the Apache Software Foundation are available under the groupId org.apache.maven.

Artifact Identifier (artifactId)

- An artifact is an identifier for a software component. An artifact can represent an application or a library; for example, if you were creating a simple web application, your project might have the artifactId "simple-webapp", and if you were creating a simple library, your artifact might be "simple-library". The combination of groupId and artifactId must be unique for a project.

Version (version)

- The version of a project follows the established convention of Major, Minor, and Point release versions. For example, if your simple-library artifact has a Major release version of 1, a minor release version of 2, and point release version of 3, your version would be 1.2.3. Versions can also have alphanumeric qualifiers, which are often used to denote release status. An example of such a qualifier would be a version like "1.2.3-BETA" where BETA signals a stage of testing meaningful to consumers of a software component.

Packaging (packaging)

- Maven was initially created to handle JAR files, but a Maven repository is completely agnostic when it comes to the type of artifact it is managing. Packaging can be anything that describes any binary software format including ZIP, SWC, SWF, NAR, WAR, EAR, SAR.

Addressing Resources in a Repository

Tools that are designed to interact with the Maven repository translate these coordinates to a URL, which corresponds to a location in a Maven repository. If a tool such as Maven is looking for version 1.2.0 of the commons-lang JAR in the group org.apache.commons, this request is translated into: http://repo1.maven.org/maven2/org/apache/commons/commons-lang/1.2.0/commons-lang-1.2.0.jar

Properties of the Central Repository

The Central Repository contains over 400,000 software artifacts occupying terabytes of disk space. You can look at the Central Repository as an example for how Maven repositories operate. Here are some the properties of release repositories such as the Central Repository:

Artifact Metadata

- All software artifacts added to the Central Repository require proper metadata including a Project Object Model (POM) for each artifact which describe the artifact itself, and any dependencies that software artifact might have.

Release Stability

- Once published to the Central Repository, an artifact and the metadata describing that artifact never change. This property of release repositories guarantees that projects that depend on releases will be repeatable and stable over time. While new software artifacts are being published to the Central Repository every day, once an artifact is assigned a release number on the Central Repository, there is a strict policy against modifying the contents of a software artifact after a release.

Artifact Security

- The Central Repository contains cryptographic hashes and PGP signatures that can be used to verify the authenticity and integrity of software artifacts served from the Central Repository or one of the many mirrors of the Central Repository.

What is a Repository Manager?

Put simply, a repository manager provides two core features:

- The ability to proxy a remote repository and cache artifacts saving both bandwidth and time required to retrieve a software artifact from a remote repository, and;
- The ability to host a repository providing an organization with a deployment target for software artifacts.

In addition to these two core features, a repository manager also allows you to manage binary software artifacts through the software development, quality assurance, and production release lifecycle. A repository manager can also search software artifacts, audit development and release transactions, and integrate with external security systems such as LDAP. A repository manager is a powerful tool that encourages collaboration and provides visibility into the workflow that surrounds binary software artifacts.

A richer, more detailed description of the features of a repository manager include:

Management of Software Artifacts

- A repository manager is able to manage packaged binary software artifacts. In Java development, this would include JARs containing bytecode, source, or javadoc. In other environments, such as Flex, this would include any SWCs or SWFs generated by a Flex build.

Management of Software Metadata

- A repository manager should have some knowledge of the metadata that describes artifacts. In a Maven repository this would include project coordinates (groupId, artifactId, version, classifier) and information about a given artifact's releases.

Proxying of External Repositories

- Proxying an external repository yields more stable builds as the artifacts used in a build can be served to clients from the repository manager's cache even if the external repository becomes unavailable. Proxying also saves bandwidth and time as checking for the presence of an artifact on a local network is often orders of magnitude faster than querying a heavily loaded public repository.

- Proxying an external repository such as the Central Repository is also an act of good citizenship; reducing the bandwidth burden on the Central Repository helps to preserve a valuable public resource.

Deployment to Hosted Repositories

- Organizations that deploy internal snapshots and releases to hosted repositories have an easier time distributing software artifacts across different teams and departments. When a department or development group deploys artifacts to a hosted repository, other departments and development groups can develop systems in parallel, relying upon dependencies served from both release and snapshot repositories. Finding an efficient way to distribute the binary software artifacts during the development cycle is essential for an organization that needs to scale system complexity and number of developers.

- Once you start using Nexus Pro as a sharing mechanism across development teams, each team can then focus on smaller, more manageable systems. The web application team can focus on the code that directly supports the web application while it depends on the binary software artifacts from a team managing an Enterprise Service Bus.

Searching an Index of Artifacts

- When you collect software artifacts and metadata in a repository manager, you gain the ability to create indexes and allow users and systems to search for artifacts. With a Nexus index, an IDE such as Eclipse has almost instantaneous access to the contents of all proxy repositories (including the Central Repository) as well as access to your own internal and third party artifacts. If a user needs to search for a particular artifact, they can use the built-in auto-completion capabilities of Eclipse, and the IDE will perform a query against an index of the repository. If you need to update a library to the latest version, click on the POM editor and use the auto-complete feature in m2eclipse. If you need to search for all artifacts that contain a specific class name, you can use m2eclipse to search an index of Maven repository artifacts by class name.

- While the Central Repository transformed the way that software is distributed, the Nexus index format brings the power of search to massive libraries of software artifacts.

Infrastructure for Artifact Management

- A repository manager should also provide the appropriate infrastructure for managing software artifacts and a solid API for extension. In Nexus Pro, Sonatype has provided a plugin API that allows developers to customize both the behavior and functionality of the tool.

- Here are just some of the features that are available as Nexus Plugins in Nexus Pro: Release Audits and Compliance, Support for Workflow and Process, Integration with External Security Providers.

Enterprise Repository Management

- Once you adopt the core features of a repository manager, you start to view a product like Sonatype Nexus Pro as a tool that enables more efficient collaboration between development groups. Nexus Pro builds upon the foundations of a repository manager and adds capabilities such as Procurement and Staging.

Managing Project Dependencies

- Many organizations require some level of oversight over the open source libraries and external artifacts that are let into an organization's development cycle. An organization could have specific legal or regulatory constraints, which require every dependency to be subjected to a rigorous legal or security audit before it is integrated into a development environment. Another organization might have an architecture group which needs to make sure that a large set of developers only have access to a well-defined list of dependencies or specific versions of dependencies. Using the Procurement features of Nexus Pro, managers and architecture groups have the ability to allow and deny specific artifacts from external repositories.

Managing a Software Release

- Sonatype Nexus Pro adds some essential workflow to the process of staging software to a release repository. Using Nexus Pro, developers can deploy to a staging directory, which can trigger a message to a Release Manager or to someone responsible for QA. Quality assurance (or a development manager) can then test and certify a release having the option to promote a release to the release repository or to discard a release if it didn't meet release standards. Nexus Pro's staging features allow managers to specify which personnel are allowed to certify that a release can be promoted to a release repository giving an organization more control over what software artifacts are released and who can release them.

Integration with LDAP

- Nexus Pro integrates with an LDAP directory, allowing an organization to connect Sonatype Nexus Pro to an existing directory of users and groups. Sonatype Nexus Pro authenticates users against an LDAP server and provides several mechanisms for mapping existing LDAP groups to Sonatype Nexus Pro roles.

Advanced Security

- Using Nexus Pro, an organization can define a master User Password Encryption Key. Each user will be given a separate Maven settings file with an encrypted password using the Maven Nexus plugin. When users interact with Nexus Pro, Nexus uses the User Password Encryption Key to decrypt a user's Nexus Pro credentials avoiding the need to send an easily compromised plain-text password over the network.

Settings Templates

- Sonatype Nexus Pro allows you to define Maven settings templates for developers. Developers can then automatically receive updates to Maven settings (~/.m2/settings.xml) using the Maven Nexus plugin. The ability to define Maven settings templates and to distribute customized Maven settings files to developers makes it easy for an organization to change global profiles or repository configuration without relying on developers to manually install a new settings file in a development environment.

p2 Repository Support

- Sonatype Nexus Pro supports the p2 repository format used by the new Eclipse provisioning platform. You can use the p2 plugin to consolidate, provision, and control the plugins that are being used in an Eclipse IDE. Using Sonatype Nexus Pro procurement, repository groups, and proxy repositories to consolidate multiple plugin repositories, an organization can use Sonatype Nexus Pro to standardize the configuration of Eclipse IDE development environments.

Sonatype Nexus Pro and Sonatype CLM

Sonatype Nexus Pro is a core foundation of a component lifecycle management strategy. Nexus Pro provides increased control of components in the repository and enables effective collaboration among teams. Many organizations wish to extend component management across the entire software development lifecycle. Sonatype CLM builds on Nexus Pro to enable visibility and control from design, through development and build, an in to production.

For more information about Sonatype Nexus Pro or Sonatype CLM, please visit www.sonatype.com

About Sonatype

Sonatype has been on the forefront of creating tools to manage, organize, and better secure components since the inception of the Central Repository and Maven in 2001. Today, over 70,000 companies download over 8 billion components every year from the Central Repository, demonstrating the explosive growth in component-based development. Today's software ecosystem has created a level of complexity that is increasingly hard to manage. Partnering with application developers, security professionals and the open source community, Sonatype has introduced a way to keep pace with modern software development without sacrificing security. We call it Component Lifecycle Management (CLM), the new platform for securing the modern software supply chain.

We believe that to achieve application security, the approach has to be simple to use, integrated throughout the lifecycle and ensure sustaining trust. With CLM we're improving the visibility, management and security of component-based development across the entire lifecycle. Together with our customers, we're ushering in a new era of application security.

Sonatype

12501 Prosperity Drive • Suite 350
Silver Spring, MD 20904
1.877.866.2836 • www.sonatype.com