Concepts and Benefits of Repository Management

By Manfred Moser
Community Advocate, Author & Trainer, Sonatype, Inc.

WHITE PAPER

ABOUT MANFRED MOSER

Manfred Moser has been dabbling with computers ever since getting a Commodore 64 in the 80s. He started using Linux and the Internet in the 90s and has been developing software professionally since before the Y2K bug frenzy. Manfred has an engineering background, an eye for detail and a desire for doing it right, while also wanting to get the software released and used. This led him to pursue agile software development methodologies before everybody was talking about it. He has a passion for any tools and infrastructure that help developers and development teams and loves mentoring others and sharing his experience and wealth of knowledge.

He is a professional trainer for Apache Maven and Nexus Repository Manager, author of books such as *The Hudson Book, Repository Management with Nexus* and the Nexus IQ Server documentation. As community advocate at Sonatype, he helps developers with their component usage on a daily basis. He is the project lead for the Android Maven Plugin and is involved in a number of other open source projects as well as local user groups. With this background he has been presenting at conferences such as AnDevCon, OSCON, DevOpsDays, JavaOne and user group meetings around the world for a number of years.

Manfred lives in Victoria, BC with his wonderful wife and three little sons. You can follow him on twitter or G+.

TABLE OF CONTENTS

OVERVIEW

Since much of today's software is assembled using open source, proprietary or 3rd party components, many organizations rely on repository management to efficiently source, store, share and deploy these components. The volume and velocity of component parts used in your software development process creates a 'software supply chain' and, in that context, a repository manager serves as your official parts warehouse. The repository manager can also provide critical insight into component quality so development teams make better choices up front, and avoid downstream technical debt and unplanned/unscheduled work.

Today, 80-90 percent of a typical application is comprised of a variety of component formats and types, such as libraries, frameworks, modules, packages, assemblies and other parts. As development teams move toward micro-services and containers, component usage increases even more.

This report explains the concepts and terminology of repository management. It then describes the inefficiencies caused by poor component management processes and the capabilities and benefits of a repository manager such as:

- Increasing developer productivity and collaboration with dedicated local storage for all components - open source, proprietary or 3rd party.
- Accelerating continuous and DevOps goals with a single repository to manage all assets related to development and delivery.
- Improving performance and stability for builds and other component users.
- Improving component selection resulting in higher quality applications and less unplanned work.

With this background, you will understand why using a repository manager is considered a best practice in modern software development and operations scenarios.

Repository management is a foundational step in a broader trend towards managing binary components across your software supply chain and throughout the software development life cycle.

CONCEPT BASICS

So what are components?

A component is a resource like a library or a framework that is used as part of your software application at runtime, integration or unit test execution time or required as part of your build or deployment process. It can also be an entire application or a static resource, like an image, without any dynamic behavior. Even an entire operating system can be viewed as a component when used with container-based systems such as Docker.

Typically, components are archives of a large variety of files such as Java byte code in class files, C object files, binary files such as images, PDF files, sound and music files and many more.

The archives use a variety of formats such as Java JAR, WAR, EAR formats; plain ZIP or .tar.gz files; other package formats such as NuGet packages, RubyGems, npm packages, Docker images, and others.

Components can be composed of multiple, nested components themselves. For example, a Java web application packaged as a WAR component contains a number of JAR components and a number of JavaScript libraries. All of these are standalone components in other contexts and happen to be included as part of the WAR component.

There are libraries and frameworks written in various languages on different platforms that are used for application development every day. It has become a default pattern to build applications by combining the features of multiple components with your own custom components containing your application code. Components provide all the building blocks and features that allow a development team to create powerful applications by assembling them and adding their own custom, business-related components to create a full-fledged application.

In various toolchains components are called 'artifacts', 'packages', 'bundles', 'archives', 'images' and other terms. The concept is the same and we use 'component' as the independent, generic term.

There are a wide variety of components created by the open source community and proprietary vendors. This ecosystem is quite large and growing quickly. For example, the Central Repository of Maven/Java components contain over 120,000 unique components and over 1 million total component versions.

Components in Public Repositories

To provide easy access to components, the open source community aggregates collections of components into 'public repositories'. These repositories are typically accessible via the Internet for free. On different platforms, you may hear terms like 'registry' used to describe the same concept. A few of the better known repositories are The Central Repository, NuGet Gallery, RubyGems.org, npmjs.org and Docker Hub. Components in these repositories are accessed by numerous tools such as package managers, build tools, IDEs, provisioning tools and custom integrations using scripting languages.

The public repositories are more efficient than a simple directory structure or download website. Users no longer have to manually find the components and their transitive dependencies and then store them in their own infrastructure. Instead they can rely on tools to perform all those tasks after a simple declaration of the components needed.

Repository Formats

Public and private repositories use varying technologies to store and expose components to client tools. This defines a 'repository format' and as such is closely related to the tools interacting with the repository.

For example, the Maven repository format relies on a specific directory structure and file naming convention defined by the identifiers of the components and a number of XML-formatted files for metadata. Component interaction is performed via plain HTTP(S) commands and some additional custom interaction with the XML files. Tools like Apache Maven, Apache Ivy, Gradle, Eclipse Aether and many others are able to easily access a Maven repository.

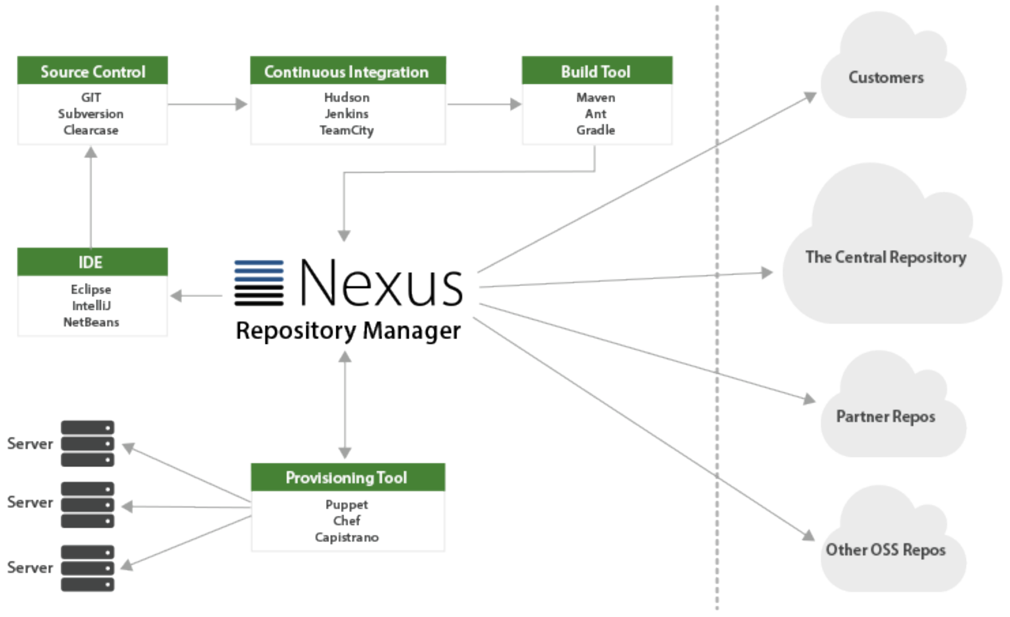
Other repository formats use databases for storage and REST API interactions, or different directory structures with format-specific files for the metadata.

Repository Management

The proliferation and usage of many varied public repositories has triggered the need to improve the process of managing and accessing components at a local level. There is a growing need to locally host internal components for teams to efficiently exchange components during all phases of the software development life cycle. Furthermore, since research shows that as many as 1 in 16 components downloaded from public repositories have a known security or license risk, component intelligence and visibility is needed early in the development process to improve overall software quality and avoid technical debt.

The task of managing access to all the public repositories and components used by your development teams can be simplified and accelerated with a dedicated server application known as a 'repository manager'. A repository manager provides the ability to proxy remote repositories and cache and host components locally. Additionally the repository is the deployment target for internal software components. These development outputs can be treated as static finished goods that are managed in the software supply chain just like external components. Other processes can pick up these goods from the repository manager for production delivery, etc.

Repository managers are an essential part of any enterprise or open source software development effort. They enable greater collaboration between developers and wider distribution of software by facilitating the exchange and usage of binary com-



Just as Source Code Management (SCM) tools like CVS, Subversion, Git and others are designed to manage source code, repository managers have been specifically designed to manage components.

ponents. When you install a repository manager, you are bringing the power of a public repository, like the Central Repository, into your organization.

Additionally, some repository managers, such as the Nexus Repository Manager, help reduce unplanned work and improve application quality by enabling development to see known security vulnerabilities, license obligations, component versions and other key factors to aid in smart component selection.

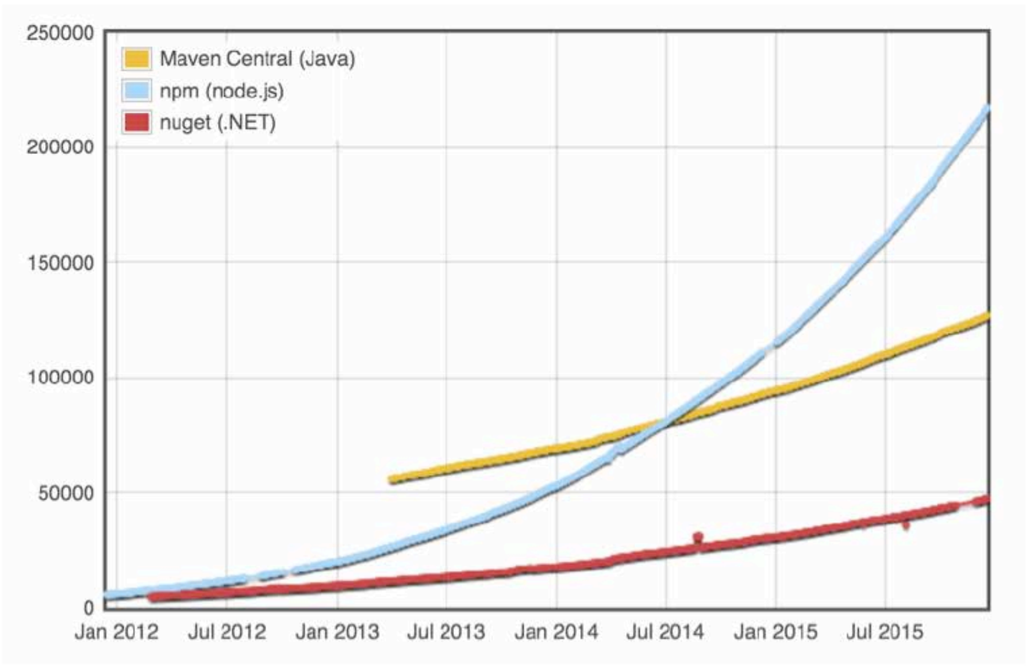
COMPONENT POPULARITY AND CHALLENGES

The days of writing your own logging framework, database abstraction layer and many other tools are long gone. All modern software development stacks rely heavily on the power of shared components (which are most often open source) to deliver this sort of essential functionality and more. This lower level functionality is often considered 'plumbing' and is an essential part of your development efforts. By taking advantage of these components to build powerful features, you can more quickly deliver applications that deliver business value and competitive differentiation.

The quantity of components has exploded as can be seen from the volume of components in the Central Repository, npmjs.org and other public repositories.

Since components form the foundation of your application, the characteristics of these components greatly influence the quality of your application. Since components are freely available and usage is accelerating, important component quality information is hard to find or easy to overlook, or both.

Furthermore, complexity has increased since components are used in all development stacks and most applications are a mixture of stacks. For example, a server-side application may be implementing a REST API using Java technologies and accessing components via Maven. However, the web application using these APIs to create a user interface uses a pure JavaScript-based approach, and sources respective components via npm. Furthermore a mobile appli-



cation is using both of these component sources, but adds platform specific components for iOS and Android into the mix.

Without a local repository manager, common scenarios like these create inefficiencies that drag down developer productivity. For example:

- **Direct download from public repositories:** Most often, each developer downloads components directly from public repositories. It's not uncommon for teams to then consume multiple versions of the same components, creating downstream maintenance issues.
- **Repeated component downloads:** Builds running on developer machines or continuous integration server clusters repeatedly download the same components and metadata.
- **Manual component distribution:** Proprietary or 3rd party components are passed around from developer to developer. Developers likely pass components around as an email attachment with some ad-hoc instructions, by overloading the usage of your version control or software configuration management (SCM) system or invent some other manual process.
- **Usage of inefficient source control system storage:** The source control system is used to store components used for your development as well as component produces by your build process. However, version control systems are typically not designed to store binary components. This results in performance degradation for all users of the SCM system, potentially rendering the system unusable. No component specific features such as improved browsing or search or rich component information is available.
- **Heavy dependence on public repositories:** The continuous integration servers and your developers heavily depend on public repositories. When you change your build or add a new dependency, your builds download dependencies from the public repositories. They rely on the availability and performance of these public resources to run. If the public resources are down, your internal development efforts slow down.
- **Inefficient build and deployment processes:** Production deployments potentially have to run the entire build, from start to finish, to generate components for deployment. When a build is tested and then ultimately pushed to production, the build and deployment scripts check out source code, run the build, and deploy the resulting components to production systems. Alternatively production deployment relies on components to be moved to the production systems using custom processes including manual file copy processes and other workarounds.
- **Custom processes for component publishing:** Since there is no established mechanism for publishing components, sharing source code with external partners means granting them access to your SCM or designing your own, potentially laborious process.
- **Difficult to understand component usage:** Creating an inventory of used components is a nearly impossible, tedious and manual task.
- **Higher storage costs:** Storage and backup costs are a lot higher due to duplicated copies of identical components in different storage locations.

The general theme in all of these behaviors is that either your systems depend on public repositories, or they all depend on the SCM system or some other storage as a central collaboration point. In many cases a central collaboration point is entirely absent, producing further inefficiencies. In addition you have to develop, manage and maintain numerous custom integration systems. By contrast, a repository manager provides an optimal solution for managing components.

CAPABILITIES AND BENEFITS OF A REPOSITORY MANAGER

In short, the repository manager acts as the authoritative storage facility for all components.

Components flow into your repository manager from external repositories as well as from internal builds and other sources. Subsequently they are accessed by development, QA and operations processes to create the final finished goods and bring them into your production environments. The repository manager is the central access and management point for any component usage in your software development life cycle. This central role makes it easy for everyone to understand where components are stored.

In addition, a repository manager can support the following use cases:

- Search and browse components in repositories and component archives.
- Display detailed component data, including component dependencies, security, license info.
- Control access to components and repositories, including audit tracking.
- Integrate with external security systems, such as LDAP or Atlassian Crowd.
- Control component releases with rules and automated notifications.
- Scale repository usage for multiple data centers, distributed teams and organizations.
- Central storage to be referenced for backup, archival and audit purposes.

As a result, a repository manager provides the following benefits:

- Time-savings and increased performance by significantly reducing remote repository downloads.
- Improved build stability by reducing reliance on external repositories.
- Reduced build times by proxying public repositories and enabling local access to components.
- Improved collaboration by providing a central location to store and manage components.
- Improved control with visibility into component information and component usage.
- Better quality software by avoiding outdated components with known security or license issues.
- Easier access to components for developers and others across continuous delivery.
- Less complexity with one, unified method to provide components to internal consumers.
- Simplified development environment and the flexibility to use a variety of build tools.

The repository manager is the central access and management point for any component usage in your software development life cycle.

CONCLUSION

You are now equipped with the understanding of the scale of component usage and their importance in your software development efforts. Components allow you to bring more powerful applications to market with less effort. A repository manager allows you to reduce and manage complexity while also building better and safer software. It is therefore no surprise that using a repository manager is considered a best practice in all organizations, especially those requiring faster and faster releases in continuous deployment and DevOps scenarios. As the local 'parts warehouse' for all build components, it is an essential foundation for a well optimized and secure software supply chain.

ABOUT SONATYPE SOLUTIONS

Nexus Repository

Nexus Repository serves as the universal local warehouse to efficiently manage and distribute component parts, assemblies & finished goods across your software supply chain. Nexus Repository supports popular component formats, including Java/Maven, npm, NuGet, RubyGems, Docker, P2, OBR, RPM and others. Furthermore, the Nexus Repository Manager has built-in software supply chain intelligence to help you avoid security vulnerabilities and restrictive licenses that lead to service interruptions, break-fixes, unplanned work and unnecessary risk. With over 80% market share, the Nexus Repository Manager is the "go-to" solution for organizations seeking to accelerate software development for Agile, DevOps, and continuous delivery or for competitive differentiation. A 14-day free trial is available from www.sonatype.com.

Nexus Firewall

Nexus Firewall provides an innovative solution to block undesirable components from getting into your repository manager. Now you can automate otherwise manual, human reviews and 'golden repository' strategies in order to keep pace with the speed of today's development practices. With Nexus Firewall you can shield your application development from waste and risk by automatically and continuously blocking these unacceptable software components inbound and preventing release of applications containing such components outbound. Nexus Firewall goes beyond blocking, providing organizations with the visibility and data needed to make ideal decisions for open source component selection early, significantly reducing risk, unplanned work and technical and security debt.

For more information about all Nexus software supply chain solutions, including Nexus Lifecycle and Nexus Auditor, please visit www.sonatype.com.

Sonatype helps organizations build better software, even faster. Like a traditional supply chain, software applications are built by assembling open source and third party components streaming in from a wide variety of public and internal sources. While re-use is far faster than custom code, the flow of components into and through an organization remains complex and inefficient. Sonatype's Nexus solutions apply proven supply chain principles to increase speed, efficiency and quality by optimizing the component supply chain. Sonatype has been on the forefront of creating tools to improve developer efficiency and quality since the inception of the Central Repository and Apache Maven in 2001, and the company continues to serve as the steward of the Central Repository serving 17.2 Billion component download requests in 2014 alone. Sonatype is privately held with investments from New Enterprise Associates (NEA), Accel Partners, Bay Partners, Hummer Winblad Venture Partners and Morgenthaler Ventures. Visit: www.sonatype.com