ARTIFACT MANAGEMENT: **BEST PRACTICES FOR SCALABILITY & GROWTH**

Publish: December 2023

Table of contents

Introduction

Artifacts (aka binaries, packages, libraries, components, etc.) are the building blocks and by-products of software development. Every day, development teams create new artifacts by leveraging a variety of technologies and incorporating existing packages, both 1st and 3rd party, as part of their process.

Due to the increased pace of development, microservice approaches, the open-source movement, and globally distributed teams, there's an explosion of artifacts for organizations to manage.

Unfortunately, managing all of these artifacts creates complexity that can slow down or block development from moving forward. For example, distributed dev teams need to leverage consistent package versions to ensure their microservice apps work together. And open-source libraries introduce potential unknowns into software including vulnerabilities, version, and compliance concerns.

As organizations grow, these challenges only increase as organizations grow. It's therefore imperative that organizations have policies, processes, and tools in place that enable organization-wide visibility and control over the artifacts, binaries, and libraries being leveraged as part of the software supply chain.

This white paper introduces the concepts and reasons to adopt an artifact management strategy, including 9 best practices for artifact management. These practices were honed over 15+ years of helping thousands of organizations big and small control how artifacts are leveraged across software development pipelines while simultaneously speeding up software delivery. We'll also cover what to look for in tools used to implement and operationalize your artifact management approach.

What is artifact management?

Before delving into artifact management, it's helpful to first understand what an artifact is. An artifact is any software asset that's connected to or part of a software development project. Artifacts commonly take the form of binary packages and are used to characterize or describe the function, architecture, and design of the application. Common types of artifacts are images, executables, data models, libraries, metadata, use cases, and, of course, the compiled application binary.

Artifact management is the practice of storing the artifacts related to a software project in an organized and structured manner while also defining and controlling how and where those artifacts may be used by various stakeholders of the software project. Artifact management ensures that all artifacts are consistently managed, versioned, and deployed across development teams — and sometimes across multiple sites — to ensure quality, reliability, and auditability.

The predominant solution for implementing an artifact management strategy is, unsurprisingly, an artifact repository manager. You might also see them called binary repository managers. Some possible alternate solutions include a shared drive, generic storage solutions, or a source control management tool, but each of these have limitations to watch out for. For instance, a shared drive has limited version control and no artifact deployment capability, and a source control management tool is really only designed for managing source code text files, not complex artifacts like large binaries or Docker images. Keep reading or skip ahead to learn what to look for in solutions for implementing artifact management.

The benefit and business value of artifact management

A properly implemented artifact management strategy allows an organization to control the way artifacts enter, advance, and are leveraged throughout the software supply chain — from development to deployment. By employing an artifact management solution, organizations can understand the interrelationships of all their binaries, giving them robust visibility into binary usage and traceability, and the ability to create automated rules or policies based on binary metadata. Additionally, artifact management can provide development teams with these four tangible benefits:

Increased Velocity

Better management of dependencies and build artifacts reduces build time and ensures version consistency across teams.

Enhanced CI/CD Stability

Creating a central hub with governed rules to receive and serve the outputs of your development pipelines, including the metadata, helps improve reliability for predictable and repeatable results.

Consistent Security and Compliance

Knowing where and how all of your artifacts are housed and managed allows organizations to more effectively apply security policies and tools.

Scalability

Implementing artifact management allows organizations to easily scale for multi-site development through better versioning, controls, and (with the right tools) the ability to deploy artifacts consistently across multiple development sites and consumption points.

Glossary of terms

Here are some common terms used when discussing artifact management that may be unfamiliar to those who aren't seasoned DevOps professionals.

Binary

While all applications start as code written by humans, they're eventually assembled to a binary file that is interpreted and run by computers. Binaries are composed of a string of 0s and 1s, and ordered in a way that can be read and executed as part of a larger computer program. Binaries are one type of artifact.

Local Repositories

Local repositories are repositories into which you can deploy proprietary built artifacts. They provide a central location to store your internal binaries and, like all repos, should be specific to a given artifact type.

Remote Repositories

Remote repositories are a proxy for a repository located on a remote server. They allow for the management of artifacts brought in from an external source (ie. Maven Central). By caching required dependencies, remote repos provide consistent and reliable package access for developers.

Virtual Repositories

Virtual repositories aggregate local and remote repositories for a given package type under a single repository structure, simplifying development and configuration with a single URL for resolution and deployment.

High Availability

High Availability (HA) means that a system that can operate continuously without interruptions, typically with an uptime of 99.9% or higher. You can ensure continuous and reliable access to your critical components with a package management system that delivers resiliency of your package repositories within a region or data center.

Replication

Replicate an artifact between one or more repositories located across multiple instances of a given artifact management solution. Replication supports important use cases such as distributed development (multi-site), High Availability, and disaster recovery.

Federation

Federation refers to a process that syncs data such as artifact metadata, identity, and authentication information across networked systems. For instance, when connecting two or more JFrog Platform instances via defined repositories in a bi-directional sync or via users and permissions.

Best practices for artifact management

1. Store packages and artifacts in a tool designed to house that specific file type

No two package types are the same — their structure, contents, and configurations will vary. While it's possible to store multiple file types in one generic repository, organizations that take this approach will end up limiting the automation of their software pipelines and reducing the data captured as part of their development process. That's why it's generally recommended to store packages and artifacts in repositories designed to house a given package type.

Why it's important

Each package manager has specific sets of commands, specifications, and data outputs. To allow for automation and seamless integration into your software pipelines, the repositories you store your artifacts in must be able to integrate and communicate with the various tools used in your build processes.

Storing packages and artifacts in a tool designed specifically for that file type also allows for easier metadata capture. When packages and artifacts are stored in a tool designed for that file type, the tool can capture and store information about the files, such as the type of file, when it was created, who created it, and any other related information.

Another important reason is that it makes it easier to stay organized. A tool designed for a specific file type can be used to organize the packages and artifacts into folders, tags, and other categories, allowing users to easily find the packages and artifacts they need. It also makes it easier to keep track of the different versions of packages and artifacts.

Example implementation

Let's take Maven, for example. Maven is a powerful tool for managing software projects and supports a wide range of project types, including Java EE projects, web applications, and mobile applications. Maven is also designed to make it easy to manage dependencies, build artifacts, and package applications.

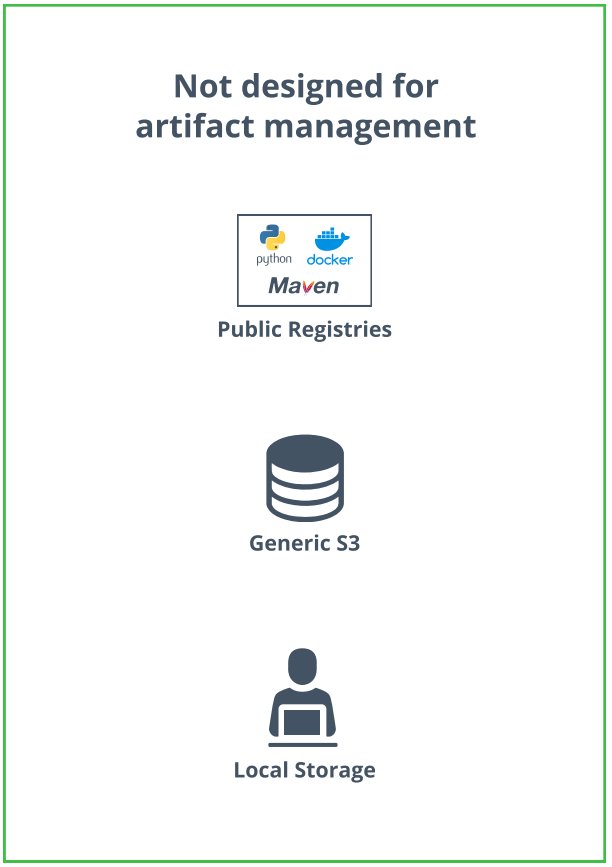
Maven Repository has to support several unique features in order to work natively with the Maven package manager. It must be able to automatically detect dependencies, including transitive dependencies, and ensure they're available and up-to-date. It also has to be able to support multi-module projects, allowing for the packaging of applications into multiple smaller parts that can be built

Best practices for artifact management

Contd

independently. Maven needs to be able to support multi-module projects, allowing for the packaging of applications into multiple smaller parts that can be built independently. Maven needs to be able to build artifacts such as JARS and WARs, and package them into distributable formats such as ZIP files. Finally, it must be able to support a wide range of build tools, such as Ant, Gradle, and of course, Maven itself.

By choosing to store packages and artifacts in a tool designed to house that specific file type (Maven in this example), developers can easily manage dependencies, build artifacts, and package applications, all without needing to manually configure and manage the process.



Best practices for artifact management

Contd

2. Segregate artifacts and dependencies with a local, remote, and virtual repository structure

Modern software applications consist of multiple packages and dependencies. This includes components built in-house and open source projects that we use to expedite our releases. For example, as part of a project, you may need logging functionality. Rather than build and test a whole logging framework from scratch, we rely on OSS logging frameworks like log4j, which are more advanced or mature.

Why it's important

For security, structural, and compliance reasons, organizations need an easy way to store and organize artifacts so that it's clear which are proprietary and which are pulled in from public places (i.e. dependencies). To better keep intellectual artifacts separate from non-intellectual artifacts, it's best to use separate repositories for these two different classifications of artifacts.

Example implementation

Best practice recommends using a repository structure that includes local, remote, and virtual repositories.

In **local** repositories, you can store all your intellectual artifacts, which you build as per your business needs.

Use **remote** repositories to cache packages from public OSS repositories like Maven Central, Docker Hub, npm, etc.

Virtual repositories offer a unique way to ease the administration of repository management. Since it's an aggregation of both local and remote repositories, you can use a virtual repository as a single endpoint to build new proprietary artifacts by resolving OSS package dependencies and publishing intellectual artifacts to Artifactory. When a client connects to a virtual endpoint, you should be able to add/remote/modify/prioritize the list of repositories that you'd like to resolve dependencies from without making any change on client configuration.

To summarize, by using local and remote repositories, organizations can separate their intellectual artifacts from non-intellectual artifacts. Using virtual repositories, organizations can ease the administration and governance efforts when change is required.

Best practices for artifact management

Contd

3. Proxy public registries for a locally cached set of artifacts

These days, open-source packages act as the basis of all software that's in use. Developers and build systems often rely on these open source and third-party libraries, which are typically hosted on remote systems on the internet, in order to build software applications. As ubiquitous as this practice is, there are certain challenges that organizations face when relying on publicly hosted artifacts.

Why it's important

Organizations must address reliability, availability, security, and traceability when dealing with artifacts from public registries. Problems can arise if artifacts/dependencies are no longer in the public repositories, or if network bottlenecks cause reliability issues. Additionally, traceability can be difficult if dependencies are downloaded directly from the internet. Caching artifacts in remote repos locally provides reliable, consistent package access without requiring constant downloads.

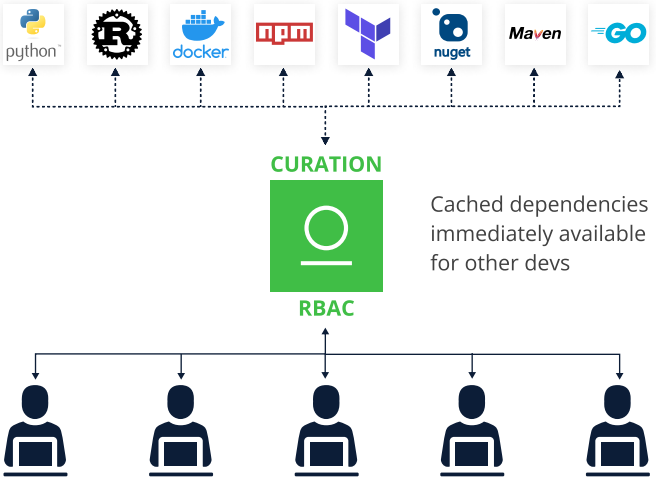
Example implementation

When pulling an Ubuntu Docker image from Docker Hub using docker pull ubuntu:20.04, the downloads come from Docker Hub directly. Everytime the same image is requested, the request has to reach Docker Hub and the download process starts all over again. Best practice is to create a remote repository, which acts as a proxy for all the requests reaching Docker Hub. Once the image has been cached in the remote proxy repository, all subsequent requests from any source are served the same cached image, which lowers ingress consumption.

SECURITY BEST PRACTICE

Allow developers to download packages directly from the internet

Organizations looking to adopt security best practices will leverage their artifact repository manager as an intermediary between developers and the internet by proxying public registries. Even if a malicious package doesn't make its way into the build, if a developer downloads a piece of malware onto their device, it could expose the entire network and system. Accessing all packages through a tool like Artifactory provides a first layer of defense and control.



Best practices for artifact management

Contd

4. Have repositories for each stage of development

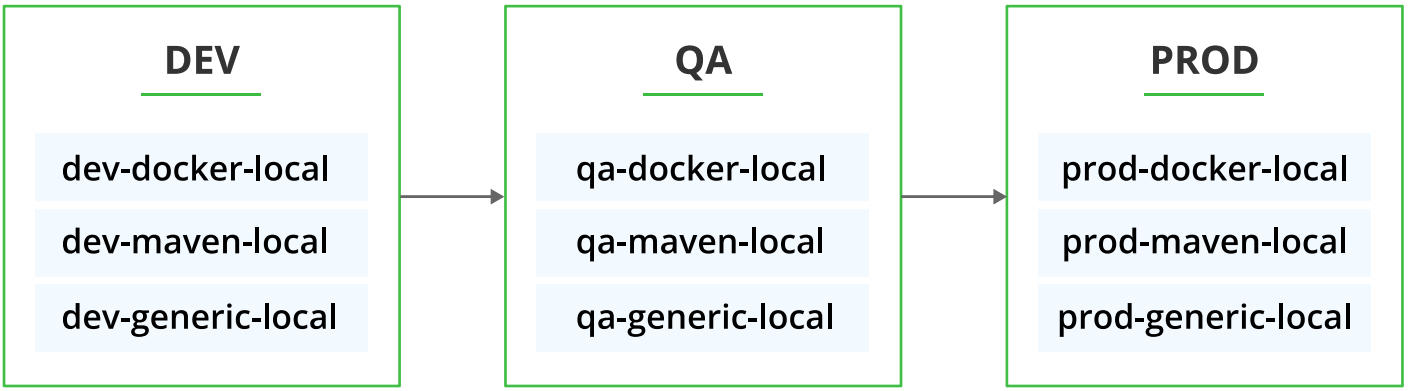
Organizations should maintain repositories aligned with every stage of their SDLC. Software releases consist of multiple builds/packages/artifacts, and organizations mature software for release by moving it through stages of the SDLC and performing various tests at each stage.

Why it's important

By establishing repositories for each SDLC stage, organizations can move all the components that are part of a potential release to repositories aligned to each stage, control when the software is promoted (typically once they've passed certain validations), control who can access the components depending on which stage of the SDLC the components are in, and never rerun a build for a piece of the release at any time. This approach helps organizations improve integrity and security, and enables transparent tracking of the status of software releases.

Example implementation

The diagram below shows a hypothetical organization's SDLC and the components that make up a release. Imagine how having a dedicated repository for each stage will help this organization move securely and seamlessly from one software development lifecycle stage to the next.



Best practices for artifact management

Contd

5. Assign each team a dedicated set of local, remote, and virtual repos

Each team or "project" should be given its own dedicated set of repositories for use in software development. Further, an artifact management solution should have some sort of management entity for grouping resources such as repositories, builds, release bundles, and pipelines. This will allow for easier delegation of resources to a specific team.

Why it's important

This is considered best practice for many reasons. First, while the instinct for organizations is to store all of their artifacts in a single system, it might make more sense for certain teams to have access to a predefined subset of artifacts for simplicity and security purposes. Additionally, assigning each team their own set of repos makes it easier for them to find and access the components they need to work on, share components across different teams, and replicate or federate specific repos when necessary. When you're working with very large repositories, this can get complicated. Assigning teams their own set of repos also removes some of the burden and maintenance overhead from the central admin. For example, a central admin may not know what they can remove or update without impacting a given team. Finally, it's a smart financial move as this allows for more efficient and traceable allocation of costs to specific teams.

Example implementation

Team A would have local, remote, and virtual repos for their required package types (Maven, Docker, Npm, and Generic, for example). They would have their own project-level admin, permissions, and policies as well.

Team B would have local, remote, and virtual repos for their required package types (Rust, Conan, Docker, Helm, and Generic for example). Like Team A, Team B would have their own project-level admin, permissions, and policies.

Best practices for artifact management

Contd

6. Promote – never rebuild – artifacts across SDLC environments

The software development lifecycle (SDLC) includes different steps, ranging from initial planning and design to coding, testing, deployment, and maintenance. One essential element of the SDLC is the transfer of artifacts from one environment to another. To make the process as efficient and effective as possible, it's recommended to promote the artifacts rather than rebuilding them for each stage.

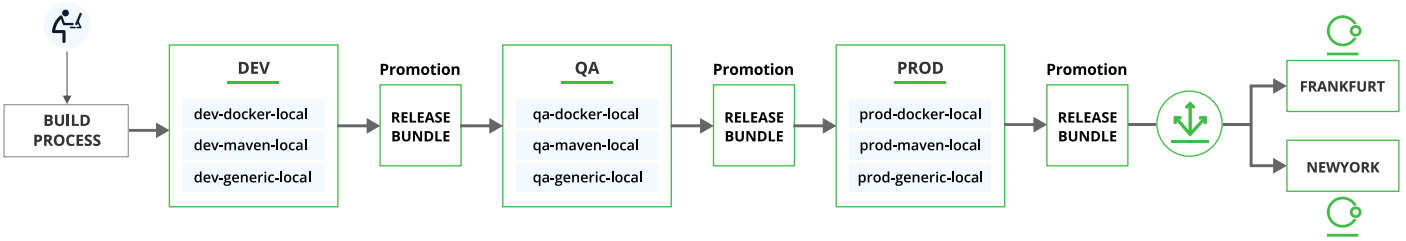
Why it's important

Promoting artifacts across the SDLC helps to ensure consistency, reliability, and traceability while saving time and resources. By promoting rather than rebuilding artifacts, developers can trust that the code they're testing is the same version that'll eventually be deployed, while also maintaining a clear trail of the software's maturation. Performing security scans as part of the promotion process can enable scan results to serve as a gate to block or approve promotion, ensuring builds are secure as they advance to the next stage of the SDLC. This approach helps to reduce errors, improve quality assurance, increase productivity, and reduce time to market, making it a valuable approach for any development team.

Example implementation

An organization with three stages of the SDLC – DEV, QA, and PROD – would mirror those stages in their artifact manager. Here's what that looks like:

- **DEV:** Build outputs and dependencies used in the coding phase would be housed in the DEV repos.
- **QA:** The builds and packages comprising a release would be bundled together and then promoted from DEV to the QA repos for further testing.
  - ***Note:*** Rules can be set up so that only artifacts promoted from DEV can populate the QA. It can further be established that only defined users have read-only access to the repo to perform testing, or can only publish back metadata against the artifact.
- **PROD:** Once proper QA validations have taken place, the software release bundle can be promoted to PROD, which can be designated as the only place production systems can pull updates from.



Best practices for artifact management

Contd

7. Store and manage the metadata associated with each artifact alongside it

As a refresher, metadata refers to the additional attributes related to an artifact. These attributes provide a means for administrators to organize artifacts in an effective way. Let's look at the different types of metadata.

General metadata

- Artifact name
- Artifact version
- Artifact type
- Artifact size
- Artifact checksum

Build-related metadata

- Build name
- Build number
- Build timestamp

Dependency metadata

- Information about the dependencies of the artifact

Custom metadata

- Additional metadata specific to the project can be defined and associated with artifacts

Why it's important

Metadata plays an important role in software development. It allows organizations to understand the history of an artifact or build and track what's happened to it, such as validations as software matures towards release. Metadata also plays an important role in understanding if something has changed about the package, which can indicate detailed metadata is essential for traceability of software and can even be used for triggering workflows, webhooks, or plugins.

Example implementation

We can store and manage metadata for artifacts using various methods. Here are a few ways to accomplish this:

1. Metadata Properties

Your artifact manager should be able to associate key-value pairs with artifacts. These properties can be used as metadata and should be added or updated via the UI or REST APIs, either individually or in bulk.

- **Manual Management via UI-** Using the user interface, you can manually add, edit, or remove metadata for individual artifacts. This allows you to enter key-value pairs or define specific metadata fields directly within the UI.
- **REST API-** Comprehensive REST APIs allow you to programmatically manage artifact metadata. You can use the API

Best practices for artifact management

Contd

to retrieve, update, or delete metadata for specific artifacts or a group of artifacts.

- **File Upload Requests**- When uploading artifacts to an artifact management solution, you should be able to include metadata information in the file upload requests. This can be achieved by specifying custom headers or parameters in the upload request, such as artifact name, version, or any other desired metadata fields.

2. Integration with Build Tools

By integrating with various build tools such as Maven, Gradle, or Jenkins, artifact management solutions can automatically capture build-related metadata during the build process. This includes details like build name, build number, timestamps, and more. The build information is then associated with the uploaded artifacts.

By leveraging these methods, you can store, update, and manage metadata for your artifacts, making it easier to organize and retrieve information about your artifacts throughout their lifecycle.

8. Create policies to define who can access a given repository and what actions they can take

Organizations should have clear rules and policies in place to control which individuals and systems can access software artifacts, and when. It's important for your artifact management solution to be able to implement these policies via RBAC and/or integrations with other identity management solutions.

Why it's important

Software artifacts are the valuable life-blood of software development. Because there is proprietary code and information contained within the packages, a big part of artifact management is ensuring that no unauthorized parties can access assets they're not supposed to.

Example implementation

Users, Groups, Roles, and Permissions are all connected in the sense that Permissions are assigned to Roles, which can then be assigned to Users and Groups. It's possible to grant Permissions at the repository level or the project level. This means that different users and groups can have different access rights depending on the scope of the project or the repository.

Best practices for artifact management

Contd

9. Define cleanup policies for repo hygiene and performance

The number of binaries being produced today is mind-boggling. As all of these binaries accumulate, they require a significant amount of storage over time. Cleaning up can mean multiple things. It can mean actually deleting the binaries from the repository manager, or in many cases it can mean archiving binaries to a solution with less expensive storage options (this is particularly relevant for highly regulated industries. In this scenario, the archived binaries are removed from the repository manager into the archival solution).

Why it's important

Just like in the physical world, if our "stuff" continues to accumulate and isn't properly organized or managed, then it becomes a serious problem. Cleanup policies for artifact repositories are important to ensure that artifacts being leveraged are secure and up to date. Having these policies in place supports the integrity of the repository by preventing old, vulnerable, or malicious artifacts from remaining in it. By removing artifacts that are no longer in use, cleanup policies can also help to reduce the overall size of the repository. This not only helps to keep the repository organized and efficient, but can also help to reduce costs associated with maintenance.

Example implementation

The industry best practice for cleanup is to use the metadata associated with the binary for cleanup. Cleanup based on metadata is extremely powerful and flexible because it provides critical information on what the artifact is, how and where it's used, its age, and more. To achieve the best results on cleanup based on metadata, the repository manager being used needs to be able to support metadata for the various types of artifacts stored.

You can use metadata such as date/time, folder name, regular expressions, number of downloads, last time downloaded, etc.

Here's an example of how to delete unused artifacts from Artifactory.

How to pick the right artifact management solution

Doing artifact management – the right way – is challenging without the right tool. That's where an artifact repository manager comes in.

An artifact repository manager is specifically designed to house, manage, version, and deploy different types of artifacts for software builds from a central location, and provides the best way to manage an infinitely expanding number of artifacts. You can try to implement artifact management best practices without an artifact repository manager, but your success will be limited.

When selecting an artifact repository manager, be sure to look for one with these capabilities:

1. Universality

Ask yourself: Will this solution support the required artifact types and build tools I already use today and those I may use in the future?

There are two elements to consider when examining the universality of a repository manager:

- First, the artifact repository should natively support whatever package and file types you work with. Native support entails supporting the same commands as the tool (package manager) used to build the package and multiple repository types (local, remote, virtual) to house and organize the artifacts. Universal repository managers typically support the widest number of package types.
- Second, evaluate the solution's ability to integrate with any of the tools you're already using as part of the development process. Explore the out-of-the-box integrations as well as APIs and CLI offered by any repository solution.

Upfront, these might seem like "nice-to-haves", but both are necessary for pipeline automation.

2. Automation

Ask yourself: How does this solution support automation of my development pipelines?

One of the biggest reasons to transition artifact management from ad-hoc, general-purpose storage solutions and multiple private registries to a purpose-built artifact repository is to allow for the full automation of software pipelines.

Look for solutions that enable a variety of triggers and workflows via webhooks, plugins, integrations, and CLI. Also, make sure the solution you invest in can support the same commands used with a given build tool to allow for full automation.

How to pick the right artifact management solution

Contd

3. Scalability

Ask yourself: Can this solution scale to handle distributed development and increased load?

There are a few attributes to consider when thinking about scaling, including:

- How many artifacts will you manage; Consider how many artifacts you have today and how many you expect to have in the future. According to JFrog data, 20% of organizations manage at least 5TB worth of artifacts, with the largest organizations handling PetaBytes of data.
- How many sites must you support; If you leverage multi-site development, your artifact repository must also offer a variety of options for keeping teams in sync, such as pull and push replication, full bi-directional sync, and a variety of definable triggers for when synchronization occurs.
- How many clients will be accessing the system at any given time; Consider the mechanisms available to handle increased load on systems that can occur from thousands of clients simultaneously pulling artifacts. Look for solutions that have caching or other distribution mechanisms available to mitigate these issues.

4. Hybrid Deployment

Ask yourself: Where can I deploy and run this solution?

A variety of factors will come into play when determining where you need to host and run your artifact management solution.

Organizations with a particular focus on security or subject to industry regulations may require certain artifacts to be stored in a self-hosted, on-premises environment. At the same time, they may have applications running in the cloud, requiring that certain artifact registries remain close to their runtime environment.

For the greatest degree of flexibility, look for artifact management solutions that allow for instances to be deployed wherever needed (in the cloud – hybrid or multicloud – and on-prem) and automatically stay in sync.

5. Security

Ask yourself: Does this solution ensure that the components are protected from outside access and are free from vulnerabilities?

The security aspect of an artifact management tool is critical and must be evaluated through two lenses: Platform Security and Application Security.

How to pick the right artifact management solution

Contd

- Platform Security involves controlling the access to the artifacts you house and manage within your artifact management solution. Role-based access controls (RBAC) with granular permission sets and user profiles are a base requirement. Also be sure that your artifact management solution integrates with your required identity management solutions such as LDAP, OAuth, SAML, SCIM, Crowd, Vault, etc.
- Application Security involves ensuring that the software you release — and thus the components within it — are free from vulnerabilities that may be exploited by bad actors. Ask: does the application management solution have integrations with the various Application Security tools (SCA, SAST, DAST, etc.) that are widely used today? Better yet, does it offer any built in capabilities native to the solution you're adopting? One of the benefits of consolidating artifact management onto a single system is that it provides a single place to apply application security efforts.

JFrog for artifact management

JFrog provides an end-to-end, hybrid, universal Software Supply Chain Platform that powers and manages the binary lifecycle. The JFrog Platform, with Artifactory at its core, manages and secures the entire flow of software components (including those flowing into your organization from external sources), the creation of net-new software, all pipeline tasks and promotions, and finally, delivery out of your software factory. With advanced, DevOps-centric security built-in, JFrog defends the software supply chain at every stage and allows for intelligent remediation of any vulnerability found.

Every binary in the JFrog Platform is fully traceable, providing insight into what software was built, how it was built, who it was built by, dependencies used, as well as which version is running and where. Each binary artifact's upload, download, promotion, inclusion, runtime deployment, retirement, archival, and deletion are comprehensively recorded and can be queried through an easy-to-use query language, REST API, and user interface. Through robust metadata and evidence capture, immutable release bundles, and continual security validations, JFrog creates trust in every release.

The JFrog Platform for Artifact Management

Definitive artifact management for flexible development and trusted delivery at any scale.

Support DevOps Flexibility

- Proxy and/or store all your artifacts with 30+ integrated package types
- Self-Hosted, SaaS, hybrid, multi-cloud deployments with the same great experience
- Integrate Artifactory across all your DevOps processes and tools

Your Single Source of Truth

- A single system for your artifacts with advanced search/tagging
- Manage the full lifecycle of binaries from build to release to archival
- Rich metadata enables wide observability and control over your assets

Create a Trusted Foundation for DevSecOps

- Set policies for access, use, distribution, and archival across teams
- Integrated DevOps-centric security available courtesy of JFrog Xray
- Fortify critical processes with enterprise-grade access to your artifacts anywhere

JFrog delivers a comprehensive approach to managing the secure flow of artifacts across your SSC, from entry to consumption.

Conclusion

Effective artifact management is a crucial component of successful software development. As the digital landscape continues to evolve, the significance of adept artifact management remains undeniable.

The best practices we've discussed are applicable for software teams of any size and serve to enhance velocity, efficiency, traceability, and overall project outcomes.

By adhering to these principles, organizations can mitigate risks, streamline workflows, and ensure the longevity and accessibility of their valuable artifacts. To take the next leap in your artifact management journey, **start a trial** of JFrog to see how easy it is to put these tips into practice.

Take artifact management to the next level:

8 Reasons for DevOps to Use an Artifact Repository Manager >