9 BEST PRACTICES
FOR ARTIFACT MANAGEMENT

Artifacts are the valuable building blocks and products of software development. In today's fast-paced and rapidly evolving software development landscape, effectively managing artifacts has become a critical factor in ensuring business success. Understanding and implementing effective artifact management practices can significantly enhance your productivity, streamline processes, and ensure the successful delivery of software projects.

This ebook aims to provide you with an overview of nine best practices for artifact management. We'll explore each best practice and provide actionable tips along the way.
By following the recommendations outlined here, you'll be equipped with the knowledge and tools necessary to efficiently handle artifacts throughout their lifecycle.

JFrog

9 BEST PRACTICES

1. STORE PACKAGES AND ARTIFACTS IN A TOOL DESIGNED TO HOUSE THAT SPECIFIC FILE TYPE

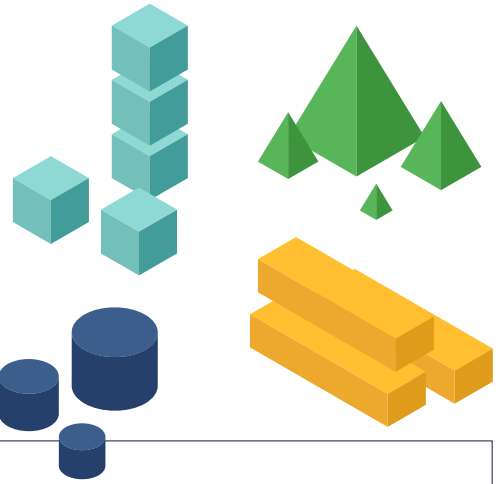
No two package types are the same — their structure, contents, and configurations will vary. While it's possible to store multiple file types in one generic repository, organizations that take this approach will end up limiting the automation of their software pipelines and reducing the data captured 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.

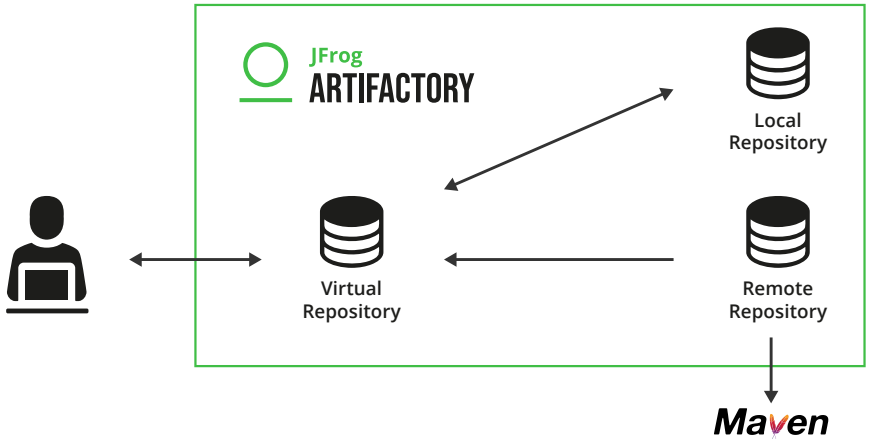
Another reason this is an important best practice is that it makes it easier to stay organized. A tool designed for a specific file type can be used to organize the packages and artifacts into folders, tags, and other categories, allowing users to easily find the packages and artifacts they need. It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



LEARN MORE

Package Managment Tool Consolidation

2. SEGREGATE ARTIFACTS AND DEPENDENCIES WITH A LOCAL, REMOTE, AND VIRTUAL REPOSITORY STRUCTURE



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.

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.

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

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

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

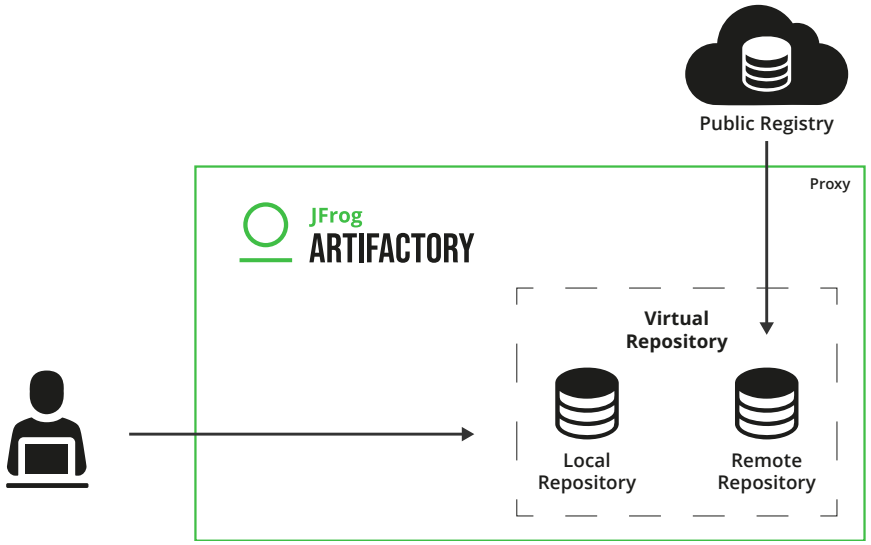
Use **remote** repositories to cache packages from public OSS repositories like Maven Central, Docker Hub, npm, etc.

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.

LEARN MORE

Best Practices for Structuring and Naming Artifactory Repositories

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 and consistent package access without requiring constant downloads.

SECURITY BEST PRACTICE

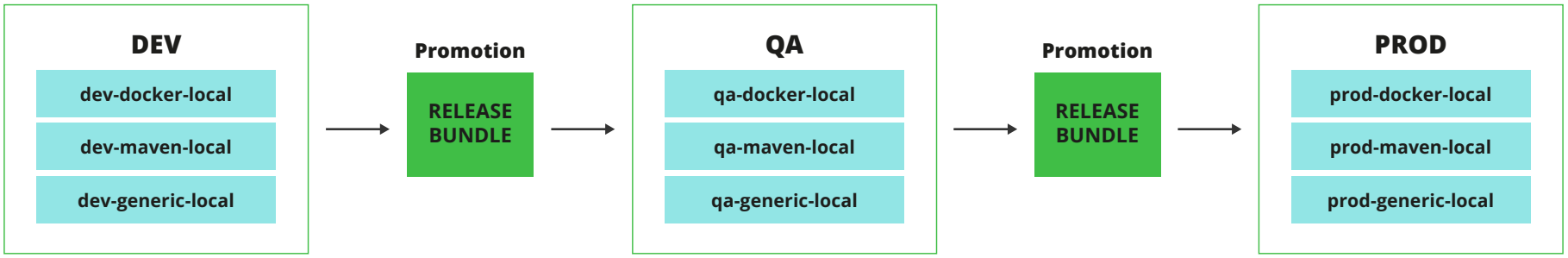
Don't allow developers to download packages directly from the internet

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.

LEARN MORE

Managing Open Source Security Risks and Vulnerabilities

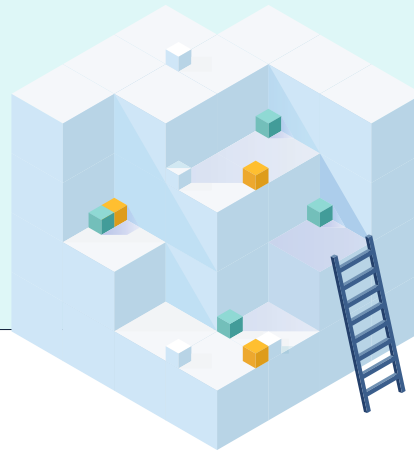
4. HAVE REPOSITORIES FOR EACH STAGE OF DEVELOPMENT



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.



LEARN MORE

Rethinking the SDLC

5. ASSIGN EACH TEAM A DEDICATED SET OF LOCAL, REMOTE, AND VIRTUAL REPOSITORIES

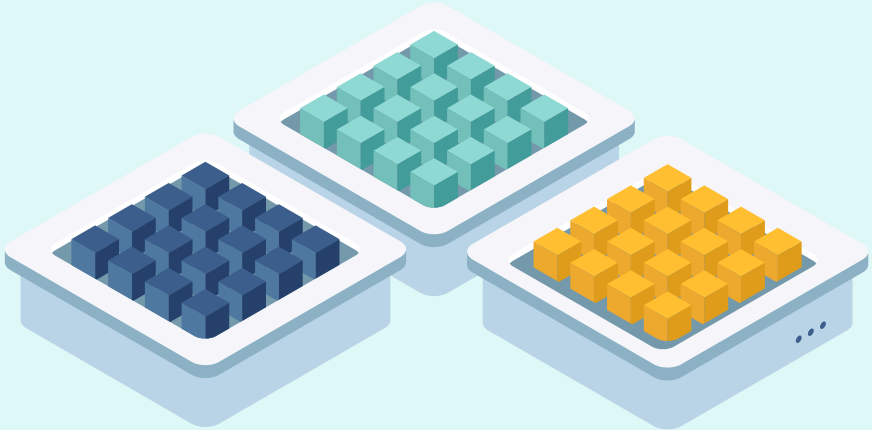
Each team or “project” should be given its own dedicated set of repositories for use in software development. Further, the artifact management solution you use should have some sort of management entity for grouping resources such as repositories, builds, release bundles, and pipelines. This will allow 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.

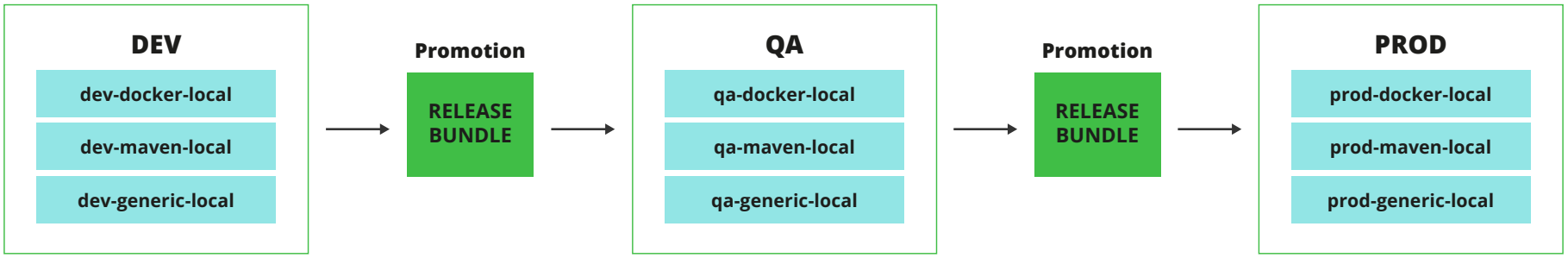
When you’re working with very large repositories, this can get complicated. Assigning teams their own set of repos also removes some of the burden and maintenance overhead from the central admin. For example, a central admin may not know what they can remove or update without impacting a given team. Finally, it’s a smart financial move as this allows for more efficient and traceable allocation of costs to specific teams.



LEARN MORE

JFrog Federated Repositories

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 software development lifecycle 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 rebuild them for each stage.

WHY IT'S IMPORTANT

Promoting artifacts across the software development lifecycle 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.

LEARN MORE

How Does Build Promotion Work?

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.

1. **General metadata**
- Artifact name: The name of the artifact
- Artifact version: The version number of the artifact
- Artifact type: The type or format of the artifact (e.g., JAR, WAR, Docker image)
- Artifact size: The size of the artifact file in bytes
- Artifact checksum: The checksum value (e.g., MD5, SHA1, SHA256) of the artifact file

2. **Build-related metadata**
- Build name: The name of the build associated with the artifact
- Build number: The number or identifier of the build
- Build timestamp: The timestamp indicating when the build was created

3. **Dependency metadata**
Dependencies: Information about the dependencies of the artifact, including their names and versions

4. **Custom metadata**
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 potential security concerns. Capturing detailed metadata is essential for traceability of software and can even be used for triggering workflows, webhooks, or plugins.

LEARN MORE

Collect and Manage Your Binary Metadata Using Build-Info

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's 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.

LEARN MORE

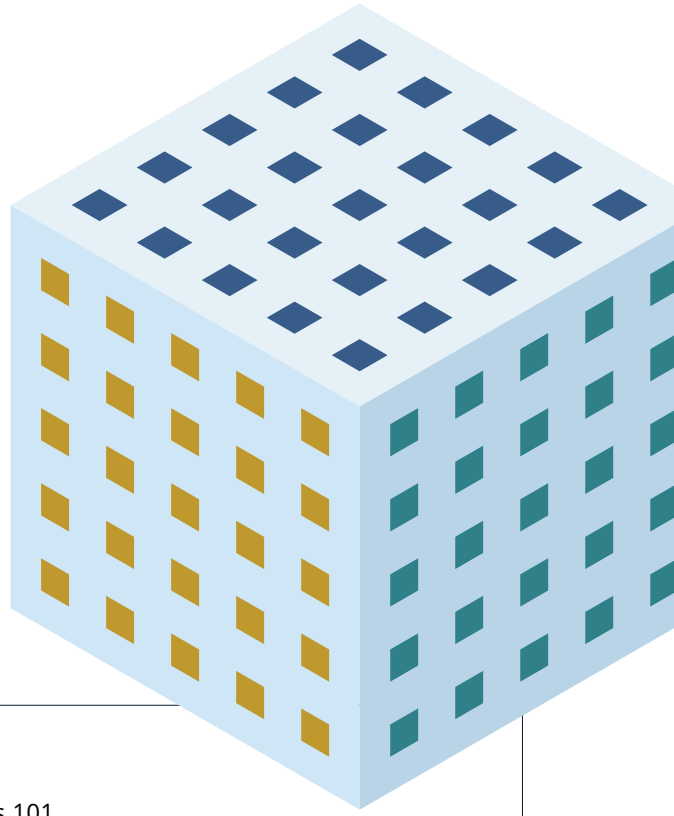
Manage Project Roles and Members in JFrog

9. DEFINE CLEANUP POLICIES FOR REPOSITORY 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.



LEARN MORE

Custom Cleanup Strategies 101

CONCLUSION

By incorporating these best practices into your artifact management processes, you're setting yourself up for success. You'll experience streamlined workflows, improved collaboration, and enhanced project outcomes. Remember, artifact management isn't just about staying organized; it is about optimizing and securing your entire development lifecycle.

We hope that this ebook has provided you with the knowledge and tools necessary to revolutionize your artifact management practices. Embrace these best practices, adapt them to your specific needs, and continuously strive for improvement. With a solid foundation in artifact management, you're well-equipped to navigate the challenges of the modern business landscape and achieve remarkable results.

For a deeper dive into artifact management, see our white paper: Artifact Management: Best Practices for Enterprise Success

ABOUT JFROG

JFrog is on a mission to create a world of software delivered without friction from developer to device. Driven by a "Liquid Software" vision, the JFrog Software Supply Chain Platform is a single system of record that powers organizations to build, manage, and distribute software quickly and securely, ensuring it's available, traceable, and tamper-proof. The integrated security features also help identify, protect, and remediate against threats and vulnerabilities. JFrog's hybrid, universal, multi-cloud platform is available as both self-hosted and SaaS services across major cloud service providers. Millions of users and 7K+ customers worldwide, including a majority of the Fortune 100, depend on JFrog solutions to securely embrace digital transformation.

