ebook

10 Best Practices for Artifact Management

Table of Contents

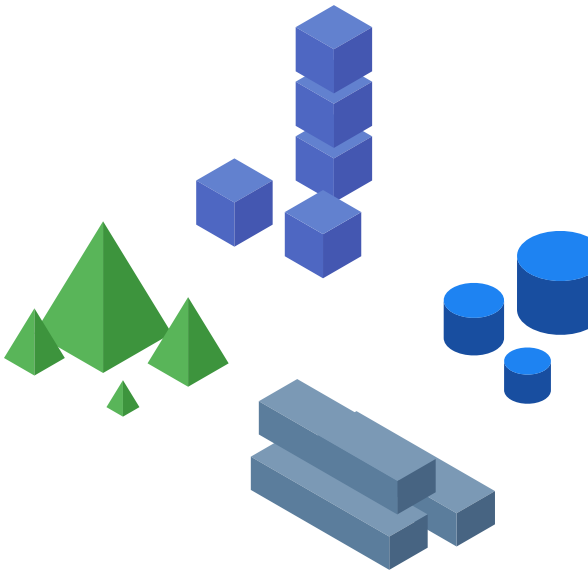
00 Introduction

Artifacts are no longer just the secondary products of development; they are the primary assets of the modern enterprise. In the era of AI, the sheer volume of binaries, libraries, and models being produced has created a deluge of data that has become the ultimate scalability concern.

For the modern digital business, the ability to manage this massive influx at scale is now the definitive hurdle to success. It is no longer enough to simply stay organized; you must deliver unconditional trust across a unified software supply chain that spans both traditional code and massive AI/ML models.

Mastering this complexity requires a shift from reactive management to a proactive strategy. The following ten best practices provide the tactical foundation needed to secure, automate, and govern your artifacts throughout their entire lifecycle, ensuring your development pipeline remains a driver of velocity rather than a source of risk.

01 Match Repositories to Package Types



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 true scalability requires storing packages and artifacts in repositories designed specifically to house a given package type. In the modern era of development, this must include support for AI assets and Machine Learning models alongside traditional binaries.

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. This now extends beyond traditional code to include native integration with AI/ML tools and model hubs like Hugging Face.

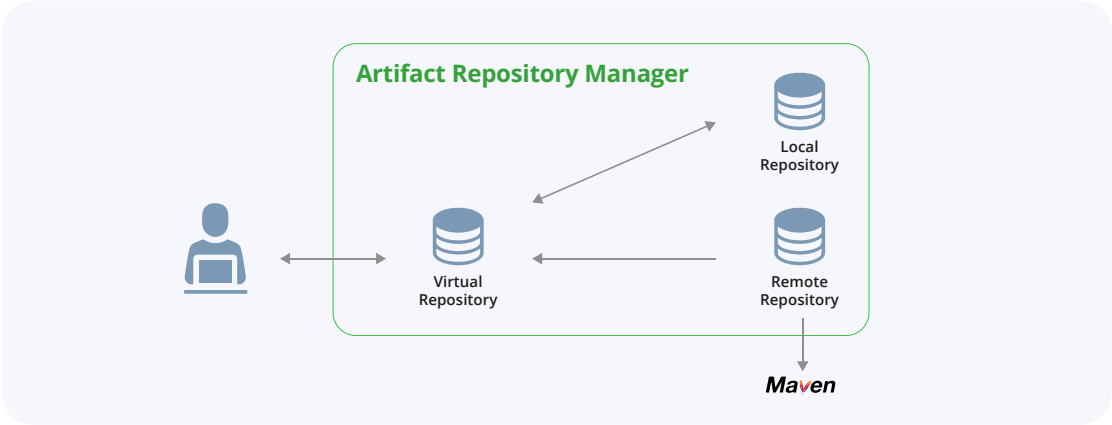
Storing packages and artifacts in a tool designed specifically for that file type also allows for easier metadata capture. When these assets are maintained in a purpose-built repository, the system can automatically record essential details relating to the provenance of the asset. For AI models, this includes vital data such as base model versions and licensing.

To manage the massive influx of data at scale, teams must prioritize structured categorization. Using a tool designed for a specific file type allows for folders, tags, and other categories that ensure users can always find what they need. It also makes it easier to keep track of the different versions of packages and artifacts across the entire software supply chain, providing a single system of record for both DevOps and MLSecOps teams.

Learn More

Package Management Tool Consolidation

02 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, AI/ML models, 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. To secure the enterprise against the modern data deluge, organizations should adopt 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. These are often configured as federated repositories to ensure your code and binaries are automatically synchronized across global sites.

Use remote repositories to cache packages and AI models from public OSS repositories like Maven Central, Docker Hub, npm, Hugging Face, etc. These repositories now serve as an enforcement point to proactively block malicious or non-compliant components before they enter your environment.

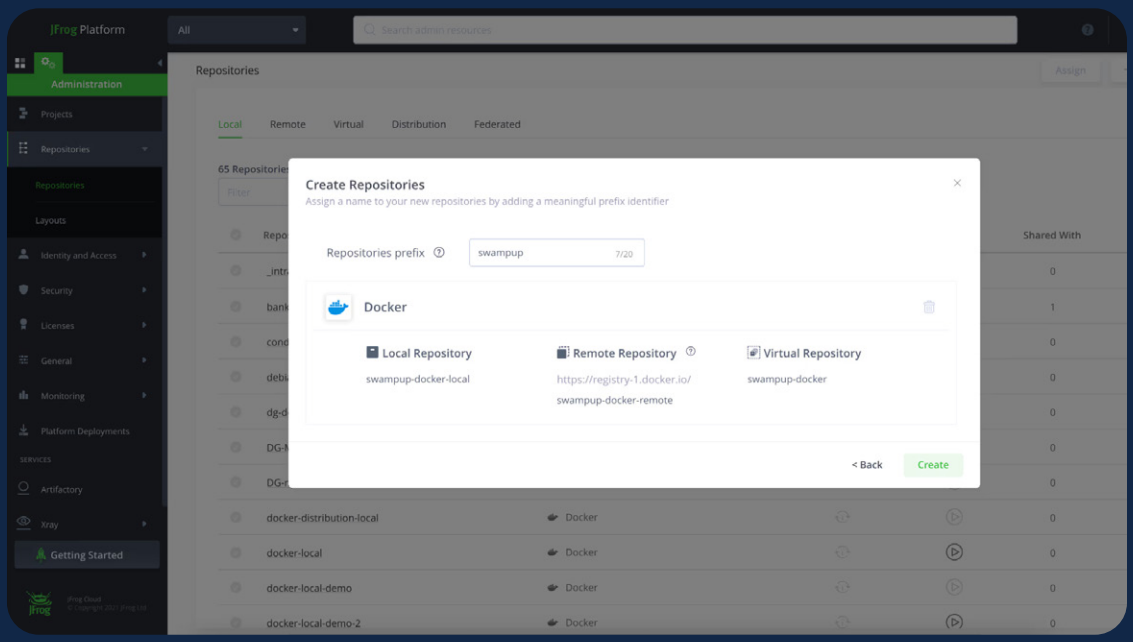
Virtual repositories offer a unique way to ease the administration of repository management. Since it's an aggregation of both local and remote repositories, you can use a virtual repository as a single endpoint to build new proprietary artifacts by resolving OSS package dependencies and publishing intellectual artifacts to your repository manager.

Why it's important

To deliver unconditional trust across the supply chain, 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 nonintellectual 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. The addition of Federation and Curation within this structure ensures that this separation is maintained globally and securely across the entire software supply chain.

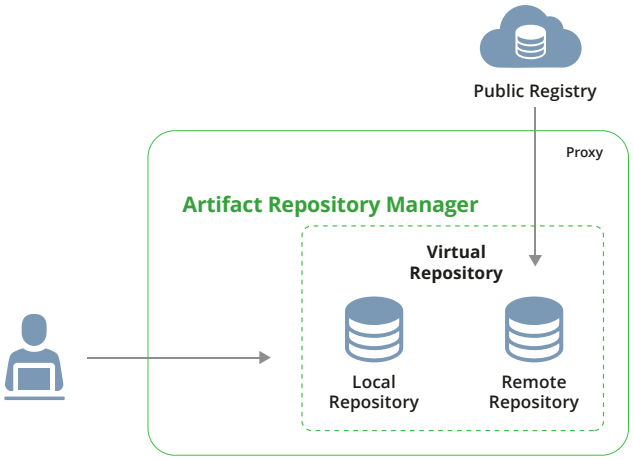
Learn More

[Best Practices for Structuring and Naming Artifactory Repositories]



03 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, agents, and build systems rely on third-party libraries and AI/ML models hosted on public registries 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.

Learn More

[Managing Open Source Security Risks and Vulnerabilities]

Security Best Practice

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

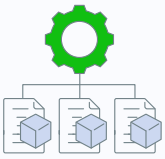
Organizations looking to adopt security best practices will leverage their artifact repository manager as an intermediary between developers and the internet by proxying public registries. Even if a malicious package doesn’t make its way into the build, if a developer downloads a piece of malware onto their device, it could expose the entire network and system.

Accessing all packages through a centralized repository manager provides a first layer of defense and control. This allows admins to set and enforce policies to block malicious or non-compliant packages and AI models “at the gate” before they can reach the developer's environment.

04 Align Repository Structures with Project-Based Lifecycle Stages

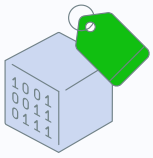
In the modern software supply chain, organizations are moving away from managing isolated repositories toward a holistic, project-centric management model. To achieve "unconditional trust," your artifact management must mirror your Software Development Lifecycle (SDLC) through defined Projects and Stages.

While traditional best practices focused on moving artifacts between physical repositories (Dev, QA, Staging, Prod), the new standard leverages unified trust and governance models to manage these as logical Stages within a single Project. This allows organizations to:



Consolidate Management

Group multiple repositories, builds, and release bundles under a single project entity for better governance.



Utilize Property Tagging

Instead of moving large binaries between different physical locations, many teams now use tags/metadata to delineate an artifact's current stage (e.g., "vetted" or "deployed").

Standardize Environments

Map specific "Stages" to your internal environments, ensuring that the terminology used in your pipeline matches your organizational structure.

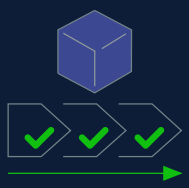
Why it's important

Automated Policy Gates

By defining stages, you can establish control gates—automated security and quality measures—that artifacts must pass before executing or entering the next stage of the lifecycle.

Project-Level Governance

Managing at the project level, rather than just the repository level, allows for better scaling of features like replication and security policy enforcement.



Eliminate Rebuilds

This structure ensures that you never rerun a build for a piece of the release. The exact version (or the validated lineage of an AI model) that passed testing in the "Dev" stage is the same one promoted to "Production".

Granular Access Control

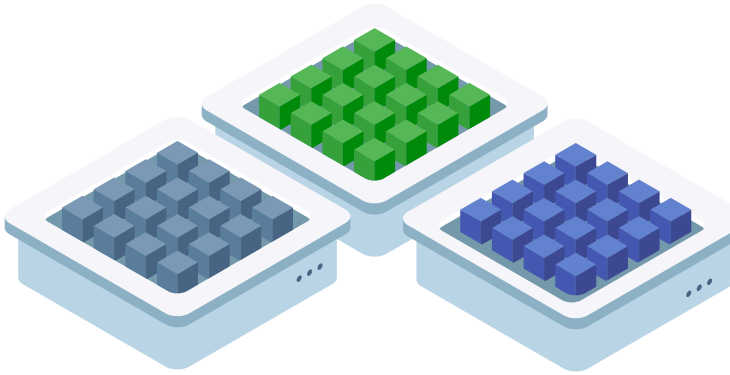
Project-level administrators can manage their own security policies and access independently, ensuring only authorized team members can interact with artifacts at sensitive stages of the SDLC.

Learn More

AppTrust Quickstart & Lifecycle Management

05 Assign Dedicated Projects to Every Team

Each team or project should be given its own dedicated Project environment for use in software development. The artifact management solution you use should have some sort of management entity for grouping resources such as repositories, builds, release bundles, and pipelines under these formal Projects.



Why it's important

This is considered best practice for many reasons. First, while organizations should store all of their artifacts in a single system, you will want to ensure only the right teams or Projects have access to certain artifacts within that system to enhance security.

Additionally, assigning each team their own formal Project 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 repositories when necessary. This is critical for scaling AI/ML initiatives where data science teams require their own workspace within the broader organization.

When you're working with very large environments, things can get complicated. Assigning teams their own Projects also removes some of the burden and maintenance overhead from the central admin while still being able to apply certain policies broadly across all Projects. For example, a central admin may not know what they can remove or update without impacting a given team. Project-level administrators can now manage their own storage quotas and project policies independently. 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

06 Promote – Never Rebuild – Artifacts Across SDLC Stages

The SDLC includes different steps, ranging from initial planning and design to coding, testing, deployment, and maintenance. One essential element of the SDLC is the logical promotion of artifacts from one stage to another. To deliver the unconditional trust required in the modern enterprise, artifacts must be promoted rather than rebuilt at each stage of the lifecycle.

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 will eventually be deployed, while also maintaining a clear trail of the software's maturation. It also ensures that the exact version of the software (or the validated lineage of an AI model) that passed testing is what actually reaches production.

Entrance and exit to a given stage is the ideal time to apply and enforce governance policies. Those policies can dictate that security scans, tests, or reviews have occurred before promotion to the next stage. With automated, evidence-based policies in place, the guardrails to trusted software delivery apply to both agent-powered and traditional software delivery. This approach reduces errors, improves quality assurance, and increases productivity.

Learn More

How Does Build Promotion Work?

07 Prioritize Artifact Metadata Management

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

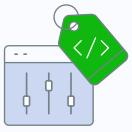
- **Artifact name**: The name of the artifact
- **Artifact version**: The version number of the artifact
- **Artifact type:** The type or format of the artifact (e.g., JAR, WAR, Docker image, or Model file type)
- **Artifact size**: The size of the artifact file in bytes
- **Artifact checksum**: The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

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

Dependency Metadata

- **Dependencies**: Information about the dependencies of the artifact, including their names,versions, and their security status.



Custom Metadata

Additional metadata specific to the project can be defined and associated with artifacts, such as AI model performance metrics or compliance certifications.

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. This metadata is foundational for governing the software delivery process and generating an accurate Software Bill of Materials (SBOM) and an AI Bill of Materials (AI-BOM) to ensure total supply chain transparency.

Learn More

[Collect and Manage Your Binary Metadata Using Build-Info]


```
{
 "agent": {},
 "buildAgent": {
   "name": "GENERIC"
 },
 "modules": [
   {
     "type": "go",
     "id": "github.com/jfrog/build-info-go",
     "dependencies": [
       {
         "id": "github.com/stretchr/testify:v1.7.0",
         "type": "zip",
         "requestedBy": [
           [
             "github.com/jfrog/gofrog:v1.1.1",
             "github.com/jfrog/build-info-go"
           ],
           [
             "github.com/jfrog/build-info-go"
           ]
         ],
         "sha1": "53b5c82ff76628b33b04017e8c81fbc1875f5737",
         "md5": "3cb74476ca750cb267db738a4db2f534",
```

08 Govern Access and Permissions Centrally

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. Security is extended through access federation, which allows organizations to synchronize these security policies and user permissions across multiple global sites automatically. This ensures that whether a developer is accessing a repository in London or Singapore, the same strict security and compliance standards are enforced.

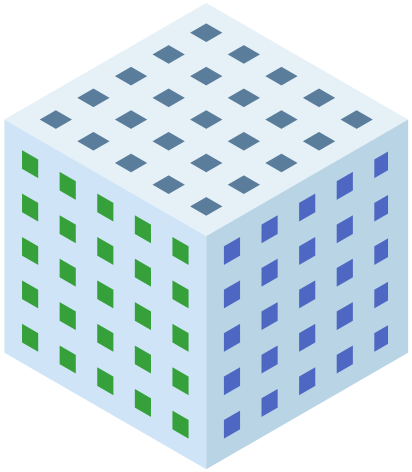
Learn More

[Manage Project Roles and Members in JFrog]

09 Implement Retention Policies for Repository Hygiene

The volume of binaries being produced today is staggering, creating a deluge of data that can quickly overwhelm your infrastructure. As these assets accumulate, they require a significant amount of storage and can degrade system performance over time. Maintaining a healthy environment requires a shift from simple deletion to comprehensive automated cleanup and archival retention policies.

Cleaning up is no longer just about deleting old files; it is about strategic lifecycle management. In many cases, especially for highly regulated industries, it means archiving binaries to a long-term optimized storage solution. By implementing automated archival policies, you can move non-operational, but governed binaries to long-term storage while maintaining their full context and metadata, keeping them fully searchable and retrievable for compliance or historical needs.



Why it's important

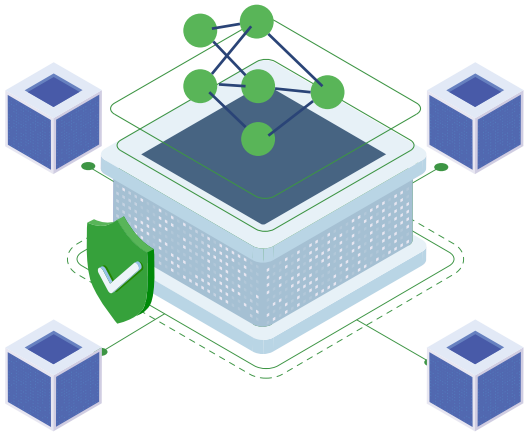
Just like in the physical world, if our “stuff” continues to accumulate without proper management, it becomes a serious burden. Strategic retention policies ensure that your active repositories remain lean, secure, and up to date. Having these policies in place supports the integrity of the repository by preventing old, vulnerable, or malicious artifacts from remaining in active circulation.

By removing or archiving artifacts that are no longer in use, you can contain the overall size of the active repository. This not only keeps the environment organized and efficient but also ensures high-performance storage. Furthermore, as organizations scale their AI initiatives, these retention policies are essential for managing massive AI/ML models and training data to prevent storage bloat and ensure precise version control.

Learn More

JFrog Retention Policies

10 Secure the AI Supply Chain: From Models to MCP Servers



As AI moves from experimental ideas into production environments, organizations must treat AI assets as first-class artifacts. This requires managing them with the same proactive strategy, governance, and security used for traditional code and binaries. This expanded plane of assets includes not just Machine Learning (ML) models, but also MCP (Model Context Protocol) servers, agent skills, and even Integrated Development Environment (IDE) extensions that power the modern developer's workflow.

Why it's important

To master the complexity of the AI era, teams need a unified way to manage the diverse range of assets that drive agent-powered delivery. By managing these assets within a unified platform, teams can leverage a centralized catalog to discover and vet models and extensions based on licensing and security scores.

Treating these as first-class artifacts ensures that every version of an agent skill or MCP server is traceable, tamper-proof, and secure. This centralized approach allows organizations to apply automated guardrails across the entire supply chain, ensuring that the AI components reaching production are vetted and trusted.

11 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 across both traditional software and AI.

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 and a unified software supply chain, 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 Scalability and Growth

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, multicloud 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.