JFrog Artifactory

Complete Guide to Repository Types

Understanding Local, Remote, Virtual, and Federated Repositories

Table of Contents

Introduction to JFrog Artifactory

JFrog Artifactory is a universal package repository manager that allows organizations to manage all binary artifacts generated throughout the software development lifecycle. It serves as a central hub for storing, retrieving, and managing artifacts across different package types including Maven, Docker, NPM, Python, Ruby, Terraform, and many others.

Why Repository Types Matter

Artifactory provides four main repository types, each designed for specific use cases and organizational needs:

- **Local Repositories:** Store internally developed artifacts
- **Remote Repositories:** Cache external dependencies
- **Virtual Repositories:** Provide unified access points
- **Federated Repositories:** Enable multi-site collaboration

---

Local Repositories

What Are Local Repositories?

Local repositories are the core storage mechanism in Artifactory where organizations host artifacts that they develop and maintain internally. These are the primary repositories where your build pipeline deploys compiled artifacts, binaries, and packages.

Key Characteristics:

- Hosted locally within the Artifactory instance
- Artifacts uploaded directly by CI/CD pipelines or developers
- Full control over artifacts and metadata
- No external dependencies or proxying
- Source of truth for proprietary artifacts

Use Cases

Internal Artifact Storage: Store compiled JAR files, Docker images, WAR files, and other binaries produced by your build pipelines

📦 **Release Management:** Maintain versioned releases of your applications and libraries, separating snapshots from production releases

Security & Compliance: Host artifacts in a controlled, on-premises environment meeting regulatory requirements

Version Control: Track different versions of artifacts and maintain complete deployment history

Example

Repository Name: `my-app-releases-local`
Purpose: `Store production releases`
Contents: `my-app-1.0.0.jar, my-app-1.0.1.jar, my-app-2.0.0.jar`
Access Pattern: `https://artifactory.company.com/artifactory/my-app-releases-local/`

Naming Convention Example

```
tomcat-mvn-release-local
tomcat-mvn-snapshot-local
docker-images-local
npm-packages-local
```

Remote Repositories

What Are Remote Repositories?

Remote repositories act as a proxy and cache for external artifact repositories. Instead of allowing direct access to external sources like Maven Central, Docker Hub, or NPM registry, developers pull artifacts through Artifactory. This approach caches dependencies locally, reduces external network traffic, and allows for policy enforcement.

Key Characteristics:

- Proxy to external repositories (Maven Central, Docker Hub, PyPI, NPM, etc.)
- Caches artifacts locally for faster retrieval
- Reduces bandwidth and external network dependency
- Can enforce security and compliance policies
- Artifacts fetched on-demand (lazy loading)

Use Cases

Dependency Caching: Cache dependencies from Maven Central, eliminating repeated downloads and improving build speed for teams using the same libraries

Network Optimization: Reduce internet bandwidth by proxying Docker pulls through Artifactory instead of pulling directly from Docker Hub

🔒 **Access Control:** Enforce security policies by restricting which external dependencies can be used in your organization

⚡ **Offline Development:** Once cached, artifacts are available even if external sources become unavailable

📊 **Audit & Compliance:** Track and log all external dependency usage for compliance and security audits

Example

Repository Name: maven-central-remote
Source: https://repo1.maven.org/maven2
Purpose: Cache Maven Central dependencies
First Access: Fetches artifact from Maven Central and caches it
Subsequent Accesses: Serves from local cache

Real-World Example

Scenario: 50 developers pulling org.springframework:spring-core:5.3.0
Without Remote Repo: 50 downloads × 2MB = 100MB bandwidth
With Remote Repo: 1 download from external, 49 from cache = 2MB + cache hits
Savings: 98MB bandwidth and 10+ seconds faster builds

Virtual Repositories

What Are Virtual Repositories?

Virtual repositories provide a single endpoint that aggregates multiple local and/or remote repositories. They don't store artifacts themselves but instead act as a logical grouping, allowing clients to access both internal and external artifacts through one URL. This simplifies dependency resolution and provides flexibility in repository management.

Key Characteristics:

- Aggregates multiple local and remote repositories
- Single URL for accessing multiple repositories
- Doesn't store artifacts (logical grouping)
- Simplifies dependency management for clients
- Allows repository layout flexibility

Use Cases

Unified Access Point: Provide developers with a single repository URL that includes both your internal artifacts and cached external dependencies

Repository Migration: Switch between repositories without changing client configurations

Logical Organization: Group artifacts by function or team while maintaining simple client access

Security & Filtering: Apply consistent policies across multiple repositories through a single endpoint

Example

```
Virtual Repository Name: libs-all
Includes:
  ├─ my-app-releases-local
  ├─ my-app-snapshots-local
  └─ maven-central-remote
Client Configuration (pom.xml):
<repository>
  <url>https://artifactory.company.com/artifactory/libs-all</url>
</repository>
Benefit: Single URL handles internal + external dependencies
```

Advanced Example: Graduated Release Pipeline

```
Virtual Repository: maven-prod
Repository Order:
1. my-app-releases-local (Production releases)
```

2. third-party-libs-local (Vetted external libs)
3. maven-central-remote (Fallback to Maven Central)
Benefit: Ensures production code uses approved artifacts first

Federated Repositories

What Are Federated Repositories?

Federated repositories enable multi-site DevOps by allowing multiple Artifactory instances across different geographic locations to act as a unified repository. Changes to artifacts and configurations are automatically synchronized across all federated members using bi-directional mirroring and replication.

Key Characteristics:

- Connects up to 10 local repositories across different JFrog Platform deployments
- Bi-directional synchronization and mirroring
- Automatic metadata synchronization
- Uses binary provider tokens for secure federation (no certificate setup needed)
- All artifact types supported (Maven, Docker, NPM, etc.)

Use Cases

🌍 **Global Distribution:** Enable development teams across multiple geographic locations (US, EU, APAC) to share artifacts with local access speeds

Headquarters + Remote Offices: Synchronize artifacts between headquarters and satellite offices while maintaining local access

Disaster Recovery & Redundancy: Replicate artifacts across multiple sites for business continuity and failover scenarios

Edge Computing: Distribute artifacts to edge locations with automatic synchronization

Example

Federated Repository Structure:
Federation Name: docker-images-global

Members:
  ├─ HQ-US: Artifactory in New York
  ├─ EU: Artifactory in Frankfurt
  ├─ APAC: Artifactory in Singapore
  └─ Canada: Artifactory in Toronto

Synchronization: When engineer in Singapore deploys app-v1.0.0.tar to Singapore repo, it automatically syncs to all other sites within minutes

Multi-Type Federation Example

> Organization can create multiple federations:
>
> 1. docker-images-federation
> ├─ HQ Docker repo (New York)
> ├─ EU Docker repo (Frankfurt)
> └─ APAC Docker repo (Singapore)
>
> 2. npm-packages-federation
> ├─ HQ NPM repo (New York)
> ├─ EU NPM repo (Frankfurt)
> └─ APAC NPM repo (Singapore)
>
> **Benefit:** Teams use local repositories while staying synchronized globally

---

Comparison Matrix

Feature	Local	Remote	Virtual	Federated
Stores Artifacts	✅ Yes	✅ Yes (cache)	❌ No (logical)	✅ Yes
Internal/External	Internal	External proxy	Both	Internal multi-site
Single URL Endpoint	Yes	Yes	✅ Yes (aggregates)	Yes (per site)
Caching Capability	N/A	✅ Yes (lazy load)	N/A	✅ Yes (local)
Synchronization	Manual uploads	On-demand	N/A	✅ Auto (bi-directional)
Geographic Distribution	Single site	Single site	Single site	✅ Multi-site (up to 10)
Use Case	Store own artifacts	Cache external deps	Unified access	Global distribution
Maintenance Complexity	Low	Low-Medium	Low	Medium-High
Scalability	Single instance	Single instance	Single instance	✅ Enterprise scale

Decision Matrix: Which Repository Type to Use?

Scenario	Recommended Type	Reason
Store your application JAR files and Docker images	**Local**	You control and deploy these artifacts
Cache Maven Central dependencies locally	**Remote**	Proxy external source, reduce internet bandwidth
Provide one URL for Maven builds to fetch both internal and external JARs	**Virtual**	Aggregates local + remote into single endpoint
Synchronize Docker images across US and EU data centers	**Federated**	Automatic multi-site replication with local access
Restrict Docker Hub access to approved images only	**Remote**	Acts as controlled proxy with policy enforcement
Support 1000+ developers with artifact access worldwide	**Federated**	Geographic distribution with automatic sync

Best Practices & Recommendations

Naming Conventions

Recommended Format: [name]-[type]-[qualifier]-[repository-type]

- ✅ `my-app-mvn-release-local` - Clear, structured name
- ✅ `docker-images-prod-local` - Indicates Docker images for production
- ✅ `npm-public-remote` - NPM proxy for public packages
- ✅ `maven-all-virtual` - Virtual repo combining Maven repos
- ❌ `repo1` - Too vague

Repository Organization Strategy

Recommended approach for medium to large organizations:

- **1. Local Repositories:** Separate for releases and snapshots (e.g., app-releases-local, app-snapshots-local)
- **2. Remote Repositories:** One per external source (maven-central-remote, dockerhub-remote, pypi-remote)
- **3. Virtual Repositories:** Production-ready (prod-all-virtual), Development (dev-all-virtual)
- **4. Federated (if applicable):** Mirror critical artifacts across geographies

Security Considerations

Access Control: Use virtual repositories to enforce permission policies across multiple repositories

Artifact Scanning: Integrate with security scanning tools to detect vulnerabilities in cached dependencies

Federated Security: Use binary provider tokens (no certificates required) for inter-site communication

Performance Optimization

- **Remote Caching:** Configure pull replication to pre-populate caches during off-hours
- **Virtual Ordering:** Order repositories in virtual repos by frequency of use (most used first)
- **Retention Policies:** Clean up old snapshots to reduce storage and improve performance
- **Federated Sync:** Schedule federation sync during low-traffic periods for large synchronizations

Conclusion

JFrog Artifactory's four repository types—Local, Remote, Virtual, and Federated—provide a flexible and scalable solution for managing artifacts across the entire software development lifecycle. The choice of which repository type to use depends on your specific needs:

- **Local** for your own artifacts
- **Remote** for external dependencies
- **Virtual** for unified access
- **Federated** for global distribution

By understanding the strengths and use cases of each type, you can design an artifact management strategy that supports your organization's growth, improves build performance, enhances security, and enables seamless collaboration across teams and geographies.
