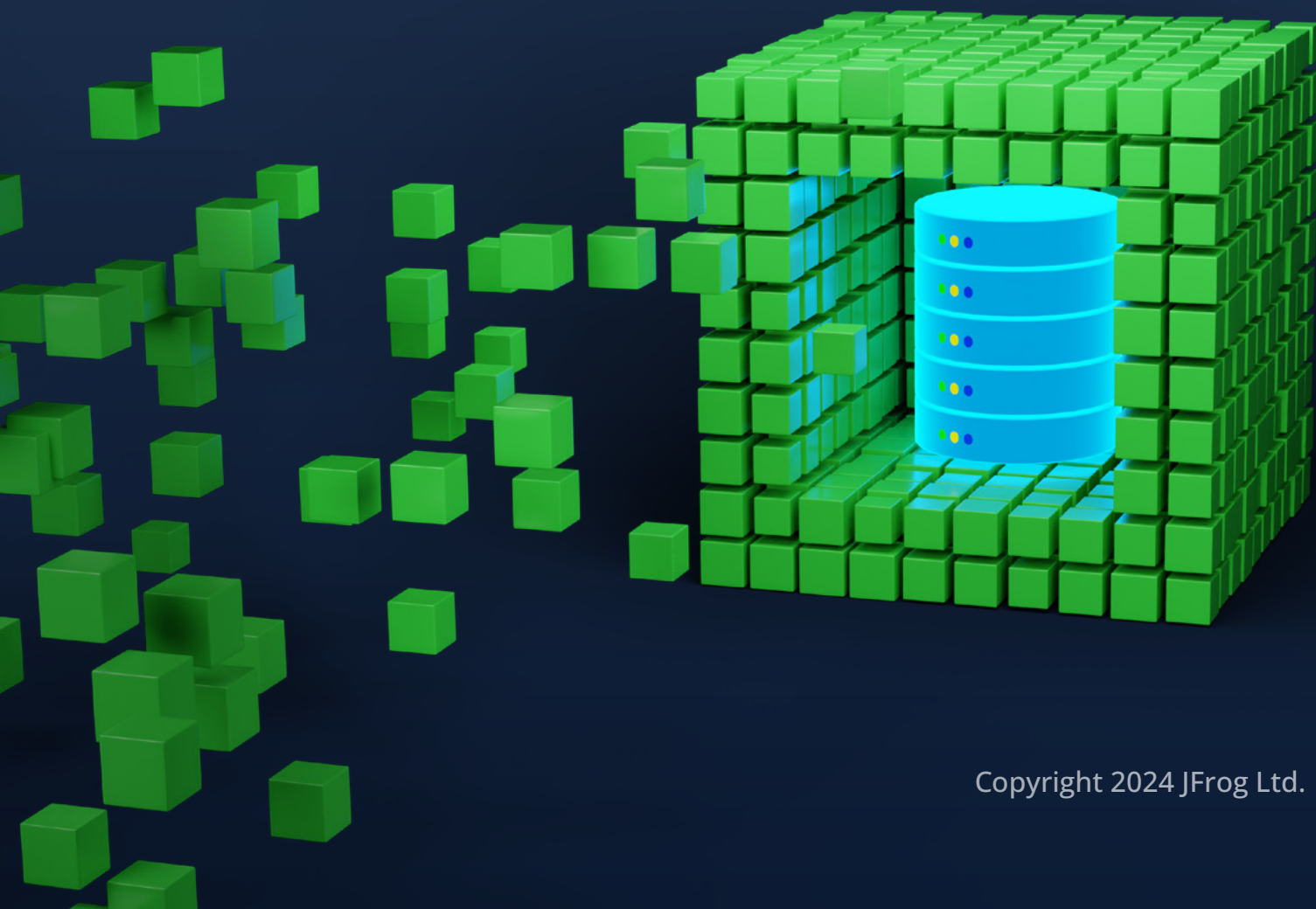




# Best Practices for Managing Your Artifactory Database

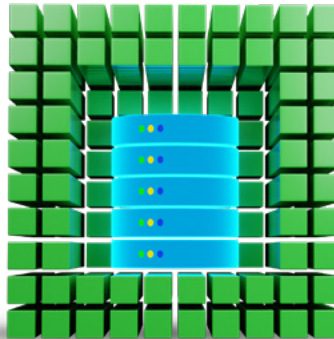
How to monitor your Artifactory Database and improve performance



# Table of Contents

- Introduction ..... 2
- What is the Artifactory Database? ..... 3
- Preliminary Considerations for Your Database ..... 4
- Database Connections ..... 4
- Configurable Parameters ..... 5
  - Defaults ..... 5
  - Parameters ..... 5
  - Timeout ..... 6
- Database Performance Factors ..... 6
  - Artifactory Query Language (AQL) ..... 7
  - Indexed Archived Entries ..... 7
  - Database Size ..... 8
  - Total Properties Per Artifact ..... 8
  - IOPS Value ..... 8
  - Database Resources ..... 9
  - Tomcat Configuration ..... 9
- Recommended Database Hardware ..... 10
- FAQ ..... 11

# Introduction



One of the key structural elements underlying JFrog Artifactory's capabilities is [checksum based storage](#), a unique artifact storage approach that optimizes many aspects of [repository management](#). This is done by storing the required artifact metadata in a dedicated database, and mapping it out to its physical storage (NAS/SAN/FileSystem/Blob Storage/S3 buckets/etc.). This relationship necessitates the connection and communication between the two and dictates the availability of resources for the database. For example, a stable and fast connection between Artifactory and the database is important to prevent issues such as slower response time for requests (GET, search, certain API calls, and even UI slowness), dropped connection, connection pool depletion, timeouts, and in some cases node synchronization issues (since Artifactory High Availability nodes synchronize via the database).

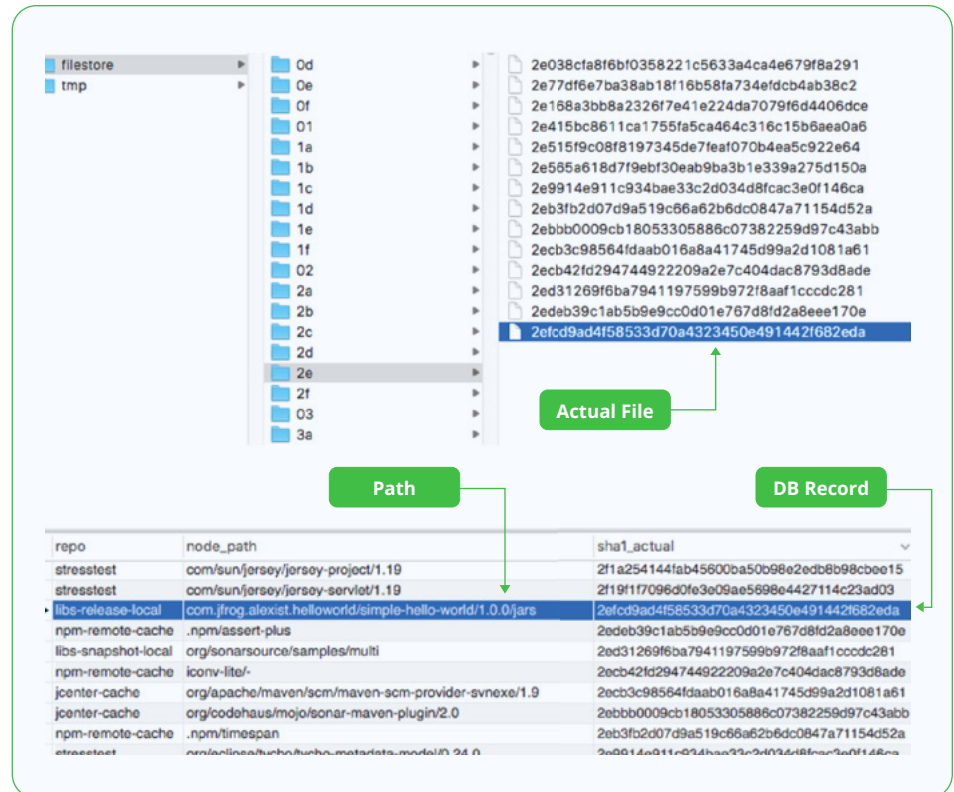
This white paper provides a foundation for understanding and managing the database used by Artifactory, including setup, optimized configuration, and tuning for working at scale.

Each binary is located in the filestore (local disk by default) and identified by its checksum (sha1), with all the metadata (such as package metadata, artifact names, sizes, creation dates, repo locations, sha256 values, and signatures) saved in the database.

[Learn more about best practices for managing your Artifactory filestore >](#)

# What is the Artifactory Database?

Let's first take a look at the relationship between Artifactory and its database. As mentioned, Artifactory utilizes a checksum based storage. This means that each unique binary is stored and renamed on the filesystem by its checksum once and only once. In order to support this, Artifactory uses the database to map the required references between the logical representation of a file identified by its checksum to its location in a virtual file system.



In addition to supporting checksum based storage, the Artifactory database contains additional information, including:

- **Properties** – *key:value* entries, part of the available artifact metadata
- **Security entities** – users, groups, permission targets, etc
- **File checksum** – md5, SHA1, SHA2
- **Archive indices** – internal structure mapping
- **File stats** – created/modified dates, size, download count, deployer, etc.

Storing this information in the database means that all Artifactory requests are translated into database queries. For example, a single REST API call to Artifactory can translate into six database queries. Artifactory uses the database for standard OLTP usage, leading to short lived and very efficient transactions. This is true from upload/download requests to the most complex [AQL queries](#). Artifactory also keeps application-related content in the database. This includes security, general configurations, and information such as [storage info](#) details.

[Learn More about the JFrog Platform resources >](#)

# Preliminary Considerations for Your Database

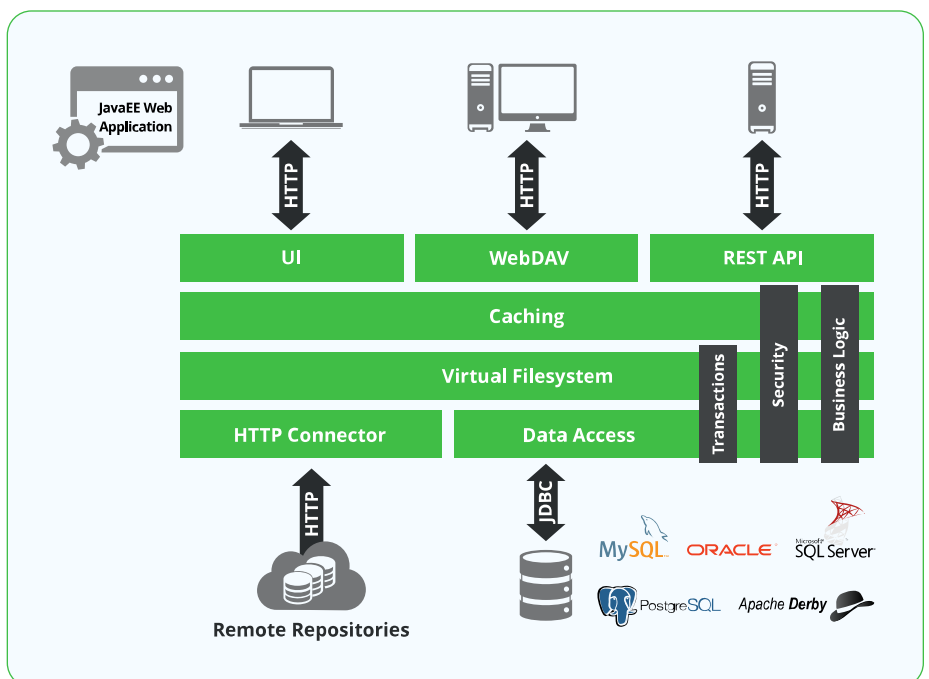
The following list includes important considerations when working with and monitoring your Artifactory database.

- **Ensure sufficient resources** for your database, including computing, storage, and networking
- **Establish backup and restore procedures** in advance to [protect your database](#) [1]
- **Plan regular [monitoring and tuning](#)** in advance for your system
- **Establish troubleshooting and support availability**
- **Create a disaster recovery (DR) plan**

[1] When considering the infrastructure, one of the most common questions is whether it is recommended to install the Artifactory database on a dedicated server or share it with other applications. While this depends on your organization's needs, it is important to take into consideration that there is continuous communication between Artifactory and the database which, in most cases, will result in high database utilization.

## Database Connections

Currently, Artifactory uses JDBC. JDBC is a Java API that allows you to connect and execute a query with a database. The JDBC API uses JDBC drivers to connect with the database as in the following diagram.



# Configurable Parameters

## Defaults

---

The default number of configured concurrent database connections is 100 for every service. This translates to 300 open database connections from Artifactory, 100 for the Artifactory application, 100 for the [Access service](#), and 100 for the Metadata. The maximum number of idle database connections that Artifactory can hold is 10.

## Parameters

---

The number of idle and database connections can be modified using the [system.yaml](#) file and the following parameters:

```
Unset
artifactory:
  database:
    maxOpenConnections: 100
    maxIdleConnections: 10
...
access:
  database:
    maxOpenConnections: 100
    maxIdleConnections: 10
...
metadata:
  database:
    maxOpenConnections: 100
```

The above values are the current default values.

When increasing the number of connections on the Artifactory application side you may also need to increase the number of connections on the database side to accommodate for the rise in connections number. Perform the following calculation to get the right value:

```
Unset
Total number of connections = (number of artifactory nodes) * 3 *
(pool.max.active) + 50
```



## Timeout

The following exception is returned and will be displayed in the [artifactory-service.log](https://artifactory-service.log) if Artifactory fails to reserve a connection.

Unset

```
org.springframework.transaction.CannotCreateTransactionException:  
Could not open JDBC Connection for transaction; nested exception is  
org.apache.tomcat.jdbc.pool.PoolExhaustedException: [art-exec-672866]  
Timeout: Pool empty. Unable to fetch a connection in 120 seconds, none  
available[size:100; busy:100; idle:0; lastwait:120000].
```

**Important Note:** Make sure to monitor your wait time for available connections. A several second wait can result in higher response time and should be avoided.

# Database Performance Factors

Estimating the number of concurrent database connections and size requires taking into consideration the daily usage of Artifactory.

This sections describes some of the common factors that will affect your database performance you'll need to take into consideration, including:



AQL queries



Indexed archive entries



Database size



Total properties per artifact



IOPS value



Database resources



Tomcat configuration



## Artifactory Query Language (AQL)

---

If you are running an enhanced [AQL query](#) for every build, this query is running for almost 30 seconds. AQL uses streaming, so Artifactory will hold the database connection for 30 seconds for every build.



## Indexed Archived Entries

---

In order to allow stored archive artifacts (zip, tar, tar.gz) to be searchable through class search and [browsable from the UI](#), Artifactory maintains the `indexed_archives_entries` table. The `indexed_archives_entries` table represents an index of files that are contained within artifact files.

When using the archive indexing functionality, the `indexed_archives_entries` will become larger. Moreover, when a new archive file is deployed to Artifactory, its content is indexed and the table is updated. These archive files can be jar, war, zip, or other archive types, as defined in the `${ARTIFACTORY_HOME}/etc/mimetypes.xml` file.

Entries are deleted from this table whenever an indexed artifact is deleted by the garbage collector, so this table should not hold any indexes of non-existing archives.

More often than not, the archive indexing related table can occupy about 40% of DB storage as well as increase DB CPU usage during related operations such as deleting an artifact. Therefore, we recommend [disabling this feature](#) when it is not in use.

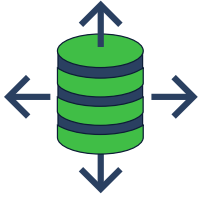
To verify if this feature is being used in your organization, you can examine the Artifactory request log and search for requests to the following paths:

[`/api/search/archive`](#)

[`/ui/api/v1/ui/artifactsearch/class`](#)

These will indicate this feature is being used, and what is the origin of the request.

[Cleaning the unused indexed archives tables](#) is also recommended if the feature is disabled. It will free up some storage that these tables previously occupied.



## Database Size

---

The database size is expected to grow as your Artifactory usage increases. Therefore, sizing the database appropriately should be taken into account when setting up the database. We need to either allow the database to autoscale its storage or ensure we can increase its storage size as we go.

Additionally, it is highly recommended to ensure some cleanup process is enabled and tuned for better performance and maintenance of the database size and reclaiming storage space (such as the PostgreSQL AUTOVACUUM for example).



## Total Properties Per Artifact

---

The database size can also be affected when Artifactory has many artifacts with a large number of properties. Today Artifactory keeps a 4,000 varchar (approximately 4k) for every property value. This means that if there are ten properties per artifact, Artifactory keeps 0.04 MB for every artifact in the database.

JFrog encourages leveraging the properties' capabilities as much as possible, since metadata is at the heart of DevOps. However, extensive use of metadata might impose performance and usage impact on the database. We recommend the `node_props` table and its related indexes table should be regularly monitored.



## IOPS Value

---

The number of IOPS (Input/Output Operations Per Second) should also be taken into consideration when configuring the Artifactory database - for example, when using the [AWS RDS](#). IOPS are a popular performance metric used to distinguish between one storage type and another. As [IOPS values increase](#), performance needs and costs rise.

Based on our expertise, when a database reaches its IOPS limits, it may cause up to a 40% decrease in performance.



## Database Resources

---

In addition to modifying database connections, there are other settings that need to be taken into consideration when it comes to database performance. For fine tuning examples review our [MySQL documentation](#), which includes suggestions for setting up your my.cnf file to use the [InnoDB engine](#).

For PostgreSQL, you can find out more about tuning in the [PostgreSQL wiki](#).



## Tomcat Configuration

---

Artifactory runs on [Tomcat](#) which uses the Apache HTTP client to handle HTTP requests. Tomcat controls a pool of threads and allocates a single thread for each incoming request. As the pool of threads is reduced, Tomcat will create more connections to handle additional requests that come in. If the number of requests exceeds the `maxThreads` value, requests will be queued. If the `acceptCount` limit is also reached, Tomcat will throw connection refused errors for any additional requests until the required resources are eventually released.

We can override the default thread pool limit by modifying the Artifactory System YAML.

Default values:

```
Unset
artifactory:
  tomcat:
    connector:
      maxThreads: 200
...
access:
  tomcat:
    connector:
      maxThreads: 50
```

**Important Note:** When modifying the Access `maxThreads`, it is required to update the `$JFROG_HOME/artifactory/var/etc/artifactory/artifactory.system.properties` file with:

```
Unset
artifactory.access.client.max.connections = <VALUE>
```

This is to modify the internal HTTP connection pool Artifactory uses to internally interact with Access.

# Recommended Database Hardware

| Concurrent Connections | Max Connections                                                       | Tomcat maxThreads | Tomcat acceptCount | Storage                                 | Processor | RAM (xmx) |
|------------------------|-----------------------------------------------------------------------|-------------------|--------------------|-----------------------------------------|-----------|-----------|
| 1 – 20                 | 100 (default settings)                                                | 200               | 100                | At least 20% of the Artifactory storage | 4 cores   | 8 GB      |
| 1 – 20                 | 100 (default settings)                                                | 200               | 100                | At least 20% of the Artifactory storage | 4 cores   | 16 GB     |
| 50 – 100               | 200                                                                   | 200               | 100                | At least 20% of the Artifactory storage | 8 cores   | 32 GB     |
| 100 – 200              | 400                                                                   | 4900              | 200                | At least 20% of the Artifactory storage | 16 cores  | 64 GB     |
| 200 – 500              | 800                                                                   | 800               | 400                | At least 20% of the Artifactory storage | 32 cores  | 128 GB    |
| 500 +                  | Please contact <a href="#">JFrog Support</a> for a recommended setup. |                   |                    |                                         |           |           |

# FAQ

## 01 How many queries does Artifactory send to the database per user request?

Approximately 1 to 8 queries run against the database per request to the Artifactory server.

## 02 Does Artifactory support High Availability databases?

Artifactory can be configured to use [multiple PostgreSQL databases \(HA\)](#) in an active/passive manner by specifying the two endpoints in a single URL block in the Artifactory system.yaml file.

This is also the case for JFrog cloud solutions that work with various cloud solutions such as Amazon RDS or others (PostgreSQL, MSSQL, MySQL, etc.), using active/passive topology behind the scenes.

Working with an active-active database set-up is highly dependent on the internal database implementation and therefore currently not fully supported for all vendors and all solutions. However, alternative solutions such as Oracle RAC, MSSQL active-active clusters, and additional solutions are also successful.

It is recommended to first validate the requirement for the additional database nodes:

For high availability, use a supported active-passive setup. For load balancing, set up your database in a way that will work for your needs and use cases.

In summary, an active-passive solution is recommended.

## 03 Do we have any caching mechanism for the database?

Artifactory doesn't cache results but every database has its mechanism for caching, such as for [PostgreSQL](#).

## 04 What is the recommended latency between Artifactory and the Database?

The network latency must be a few milliseconds at the most.

In an HA cluster, the nodes require LAN quality network connectivity, a stable network with low latency, and high throughput. This means that different availability zones within the same region are supported, but cross zone topology for an HA cluster is not.

The reason for this requirement is that JFrog Artifactory is chatty with the database. In a best case scenario, slow connection to the database is directly correlated with slower response time for requests (such as GET, search, certain API calls, and even UI slowness). In other cases, this can lead to dropped connections, connection pool depletion, timeouts, and in some scenarios even lead to node synchronization issues (nodes synchronize via the database).

Following this, it is recommended having a cluster in every data center with replication between clusters instead of having an extended cluster in two data centers.

## 05 Is it recommended to use the bundled database or an externalized one?

Artifactory comes with an embedded Derby Database out-of-the-box which it is pre-configured to use, however, for better performance and to reuse existing infrastructures you may have, you can configure Artifactory to work with alternative supported databases.

In general, the bundled DerbyDB is a very good, fast, and reliable database for supporting up to ~700K artifacts. After this point, we noticed performance degradation.

Pros and Cons for utilizing a bundled database:

### Pros:



**Speed:** Derby is a very lean and fast database that is improved by sharing the same JVM as Artifactory (no “connection over HTTP”).



**Management:** Using the bundled database reduces maintaining additional products/components (aka external database).

### Cons:



**Resources:** Since the database and Artifactory live in the same JVM, there is a risk of competition over resources. This is especially true as the number of artifacts in the system increases. Not as scalable as an external database.



**Management:** Although there is less management in a bundled database, maintenance and other operations are limited. For example, connection to the database is possible only when the Artifactory application is turned off.

## 06 Which database is recommended? Oracle, PostgreSQL, MySQL, or other?

While Artifactory supports a wide variety of database options, we do recommend [PostgreSQL](#). Using PostgreSQL allows for better alignment with all JFrog products and a simpler and more resilient setup for the JFrog Platform. PostgreSQL also supports HA and limitless scalability as well as unique PostgreSQL features like Native Locks.

A full list of all supported databases can be found in our [Artifactory Database Requirements page](#).

## 07 Can I modify the configuration during an upgrade of an HA cluster?

In an Artifactory HA setup, it is possible to continue the regular, day-to-day, operations during a system upgrade. However, it is not possible to modify configuration settings.

This is because many changes could be happening in the background during a system upgrade. For example, new tables/columns, and configuration files are created or modified

To prevent conflicts and data corruption, until all Artifactory nodes are on the same version, no configuration changes can be submitted.

Notice that an Artifactory upgrade is a relatively short process, and should take minutes.