BEST PRACTICES FOR MANAGING YOUR ARTIFACTORY FILESTORE

July 2019 | www.jfrog.com

TABLE OF CONTENTS

INTRODUCTION

As your binary repository manager, JFrog Artifactory is a central piece in your CI/CD process. It is the gatekeeper of your binaries: 1. securing access and applying permissions such as read, write and delete to your repositories, 2. leveraging your binary lifecycles by applying metadata and using the promotion system, 3. ensuring the availability of data by implementing an Artifactory cluster.

In this highly available context, you will also rely on a highly available load balancer, proxies, database and storage solutions. Artifactory provides advanced capabilities that can be integrated with any kind of storage solution.

Learn more > about the benefits you can get from Artifactory.

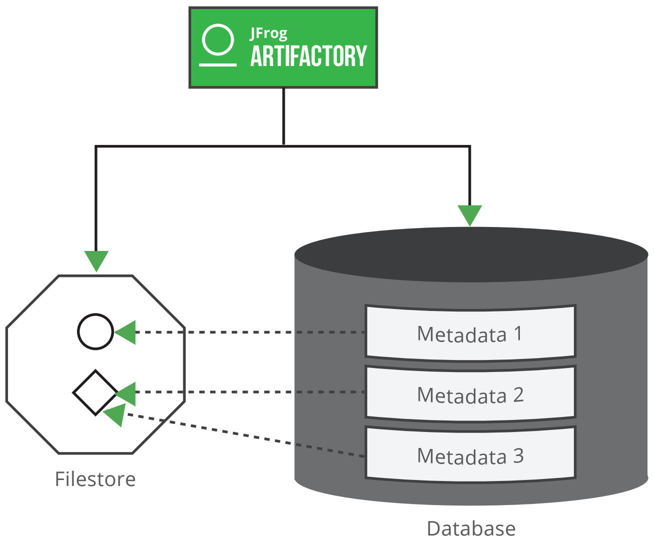
This white paper describes how Artifactory stores and manages your binaries, providing you with storage capabilities that include:

- Checksum based storage
- Internal caching mechanism
- Storage configuration options
- Backup and recovery strategies

Managing your data with Checksum-based storage

Artifactory's optimal design utilizes checksum-based storage, which uses two components: **filestore and database**. 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.

A checksum approach enables unique identification of artifacts, without duplication in the filestore. Remote teams can consume the same dependency, that is stored in the filestore, from two different repositories with different metadata referenced from the database. This prevents duplication and consumes less storage.



Separating the binaries from their metadata provides flexibility on how the data is managed.

- Moving or copying artifacts to a different path (for example, during promotion) will only affect the database and the SQL query, and not the filestore. This is because this will only change the artifact metadata, which is stored separately in the database.

The database server may be sized according to the following ratio: 1/100 of the filestore. This is not a strict formula to follow, it is meant to give you an initial sizing. Make sure to monitor your DB storage.

When sizing your DB storage, keep in mind that during Artifactory upgrades, the database schema may be modified and temporary tables will be created, which will increase the DB storage you'll need.

By default, Artifactory is installed with the Apache Derby database. This database can be migrated at any time to an alternative relational database. Database As A Service can also be used as long as they rely on the supported DBs and versions.

Learn more > about supported databases and versions

Caching binaries for better performance

Out-of-the box, Artifactory stores all the artifacts locally without any active caching. There are 2 types of caches that can be enabled in Artifactory: **Cache-fs** and **Eventual**.

1. **Cache-fs:** A read buffer, used to optimize the traffic between your Artifactory and a remote storage (for instance a NAS or cloud storage). This "Least Recently Used" (LRU) cache will host the most recent uploaded and downloaded artifacts. It is enabled to reduce the number of requests to your network storage, and consequently the response time. Since the maximum size can be set, artifacts in this cache will "rotate" based on their last downloaded date or creation/modified date.
**Available for Pro and Enterprise Licenses*

In case of concurrent connections requesting a binary not already cached, the cache-fs will only let the 1st connection download the binary from the remote server while the others will start streaming the bytes which would have already been downloaded

2. **Eventual:** A write buffer, used to optimize slowdowns during the upload process when using a slow and/or remote storage. This buffer is enabled by default for all Artifactory storage templates relying on object storage (such as AWS S3, Azure Blob Storage, ...). This allows to implement an asynchronous upload in order to not wait for the artifact to be uploaded to the remote storage and consume it right away. Once the artifact is uploaded to storage and is available in Artifactory, it will be deleted from the eventual cache. Unlike cache-fs, there is no cache size limit that can be tuned. The disk size hosting this cache will set the limit, and should be monitored.

IIt is recommended to have these caches installed on disks with the highest available IOPS and a low latency.

Tuning your cache

Cache-fs location and max size should be tuned according to your enterprise needs. Especially its max size, as it specifies when cache will rotate, and you may want to ensure that intensively used binaries are cached. For example, if one of your use cases is to:

- Consume Docker images, you might want all the layers belonging to your base images to be cached
- Fast provision your linux servers, you might want the latest OS packages (debian, rpm) to be cached
- Speed up builds, you might want to cache the most used dev dependencies

When enabled, this cache is set by default to 5 GB.

Using Artifactory Query Language (AQL)
Get a list of the most recent downloaded and uploaded artifacts using AQL query, which will return a list of artifacts along with their size. Use scripting to add the artifact sizes and get an estimate.

Eventual cache: is enabled by default in the *binarystore.xml* for tem plates such as s3-storage-v3, azure-blob-storage, google-storage, cluster-s3-storage-v3, cluster-azure-blob-storage, and cluster-google-storage. Its location, $ARTIFACTORY_HOME/data/eventual, and its size cannot be tuned. As the eventual cache will queue all the binaries to be uploaded to the object storage, make sure the allocated storage for the eventual will be higher than your biggest bulk upload. This might be the size of your biggest artifacts or published artifacts in your Build Info. Increasing the number of threads dedicated to the upload can also be part of your tuning.

Each tuning has a direct impact on your Artifactory instances.
Learn more about how to monitor them. Also, learn more about BullFrog, an open source monitoring tool for Artifactory.

STORAGE CONFIGURATION OPTIONS

It’s a common best practice in software engineering to separate the data and configuration layers from the application layer. Storing the data and configuration in a different location, external to the service host, will be beneficial. This is especially true from an operational aspect, such as monitoring, upgrading, backup/restore and scalability.

There are several different filestore configurations available to choose from depending on your organization requirements. This document refers to IT teams that are responsible for storage and infrastructure. This section describes 4 storage configuration types in Artifactory:

1. Self-managed file and block storage (Local storage, NAS, SAN)
2. Cloud file storage (for instance, AWS EBS or EFS, Azure disks or files)
3. Object storage (for instance, AWS S3 or Azure Blob Storage)
4. Database storage

All Artifactory templates are fully configurable.
The filestore configuration is defined in Artifactory using a template specified in the *binarystore.xml.* Each template has specific parameters that can be tuned.

1. Self-managed file and block storage (Local Storage, NAS, SAN)

Note: NFS versions 3.* and 4.* are supported.

In an HA cluster configuration where all the members share the same filestore, concurrent accesses are managed at the filesystem level.

Use Case 1: No infrastructure restriction

Your IT teams will manage the availability of your binaries through a highly available NAS or SAN solution and without any size restrictions concerning your NFS mount points or SAN volumes.

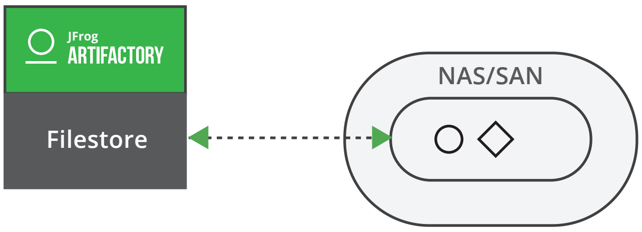
Recommended templates: filesystem and cache-fs

Use Case 1: No infrastructure restriction

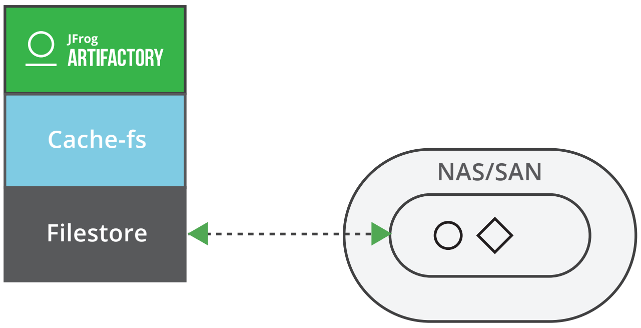
Your IT teams will manage the availability of your binaries through a highly available NAS or SAN solution and without any size restrictions concerning your NFS mount points or SAN volumes.

Recommended templates: filesystem and cache-fs
In an Artifactory standalone configuration, the filesystem template might suit your needs by referencing your mount point or local folder (pointing to a SAN volume) as the location of your binaries. Choosing the cache-fs template will enable a read cache (see cache-fs in the “Cache” section) which can bring great benefits when big binaries (from hundreds of MB to GB) are often downloaded. The binaries will be kept locally and the network time to reach the remote storage will be completely suppressed for the future requests.

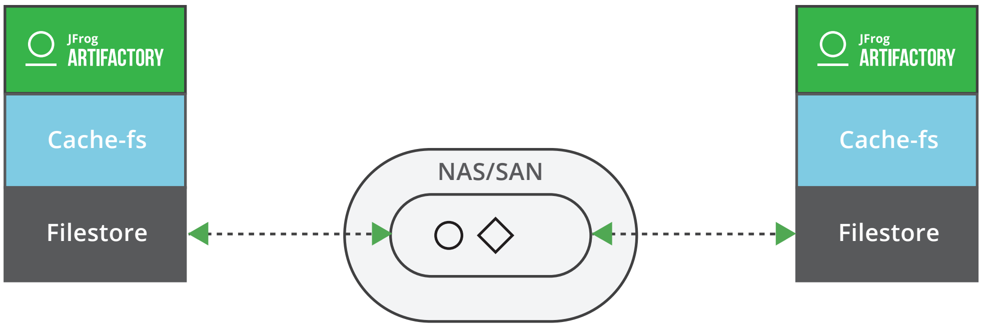
Filesystem template



Cache-fs template



In an Artifactory cluster configuration, all the nodes will use the same mount point.



The storage scaling process can be achieved vertically (scale up) by extending the existing volume and practically allocating more space to the NFS mount or volume.

Use Case 2: Restriction on storage size or storage availability across sites

Your IT teams will manage the availability of your binaries but have a restriction regarding the disk size, impacting the way you can scale up the storage dedicated to Artifactory.

Recommended templates: double-shards or redundant shards

Learn more > about the filestore sharding

The double sharding template implements the filestore sharding, where each shard represents a portion of the global filestore and where a shard can be a mount point (NAS or SAN). This allows you to scale up your storage by adding more shards.

The main difference between these 2 templates is the redundancy parameter, that is the number of copies per binary spread among the shards. If your IT teams guarantee a redundant storage (for instance a highly available NAS), you don't need Artifactory to manage the redundancy and so you can use the “double-shards” template with a redundancy = 1.

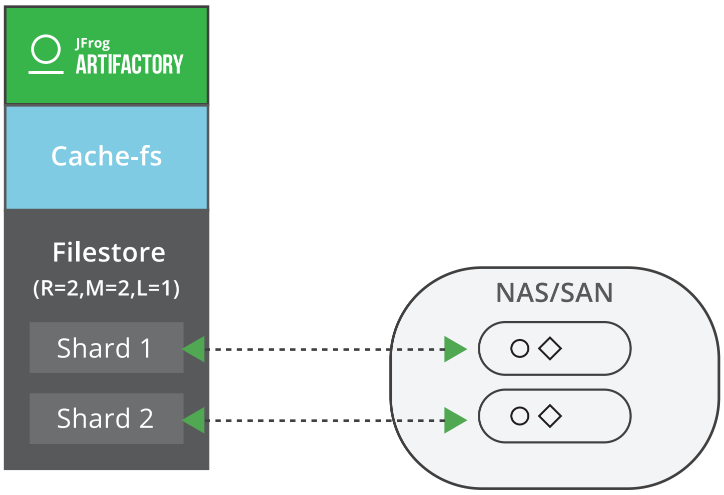
As mentioned, templates are tunable, with parameter changes such as:

- The number of shards (by default they're set with 2 shards)
- The redundancy - the number of copies per binary spread among the shards
- Write mechanism: free space per shard, percentage of free space per shard or round robin.
    - **roundRobin (default):** Binaries are written to each mount using a round robin strategy.
    - **freeSpace:** Binaries are written to the mount with the greatest absolute volume of free space available.
    - **percentageFreeSpace:** Binaries are written to the mount with the percentage of free space available.

These templates also enable a read cache (see cache-fs in the “Cache” section) per Artifactory instance. In a cluster configuration, these caches are not synchronized. This configuration can be applied to both standalone and cluster. All the mounts should be available for each Artifactory instance.

The scaling process can be done vertically by increasing the shard size or horizontally by adding more shards.

Standalone: Redundancy = 2, Mount (Shard) = 2, LenientLimit = 1



The redundancy must be less than or equal to the number of mounts in your system for Artifactory. If the number of writes specified by the redundancy is not reached (mount failure for example), the WRITE transaction will fail.

The “Lenient Limit” is the minimum number of successful writes to validate a “WRITE” transaction / upload process. Its default value is 1 and if set to 0, Artifactory will consider it as equal as the redundancy parameter.

The WRITE transaction will be committed once the number of copies of a binary has met the requirements defined by the “Lenient Limit” or the “Redundancy”. To optimize the transfer, the Write transaction relies on concurrent streams to copy the binaries onto multiple shards.

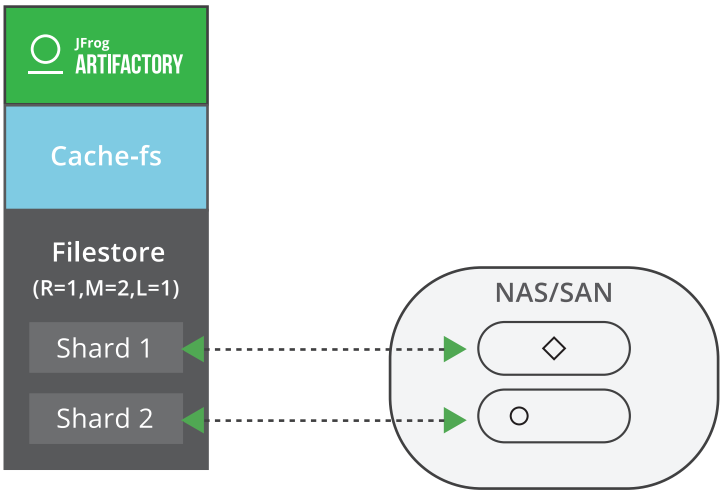
In this case, a recovering system will re-apply the redundancy within the filestore after each Garbage collector execution.

The maxBalancingRunTime parameter defines the duration of the recovery system after each Garbage collector run.

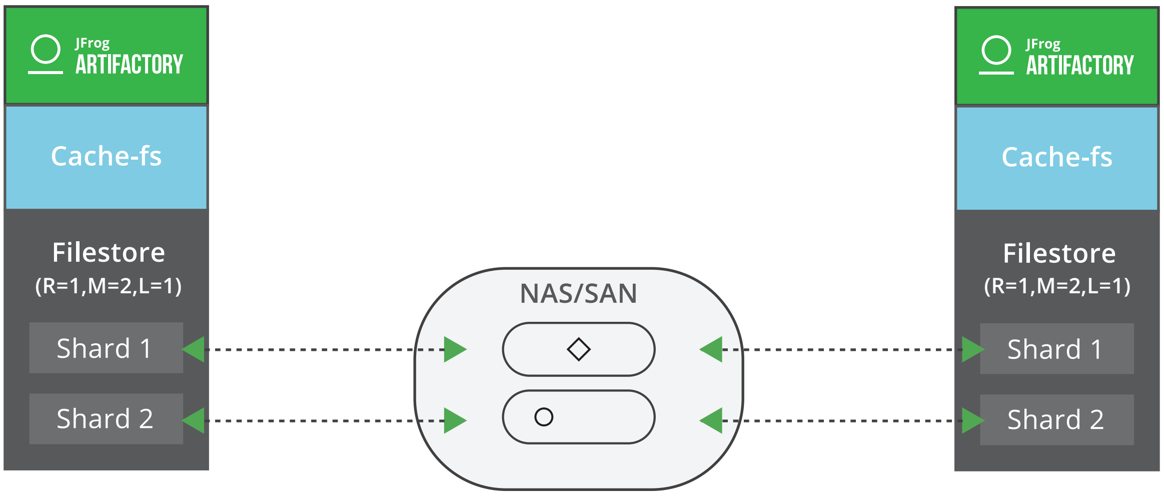
The Optimize Storage Rest API allows you to manually apply a new redundancy or re-apply the current one after shard failures. In both cases, redundancy has to be higher than 1. The process will be executed after the next Garbage collector run.

Artifactory does not know which shards hold your binaries, which allows you to freely move them from a shard to another

Standalone: Redundancy = 1, Mount (Shard) = 2, LenientLimit = 1



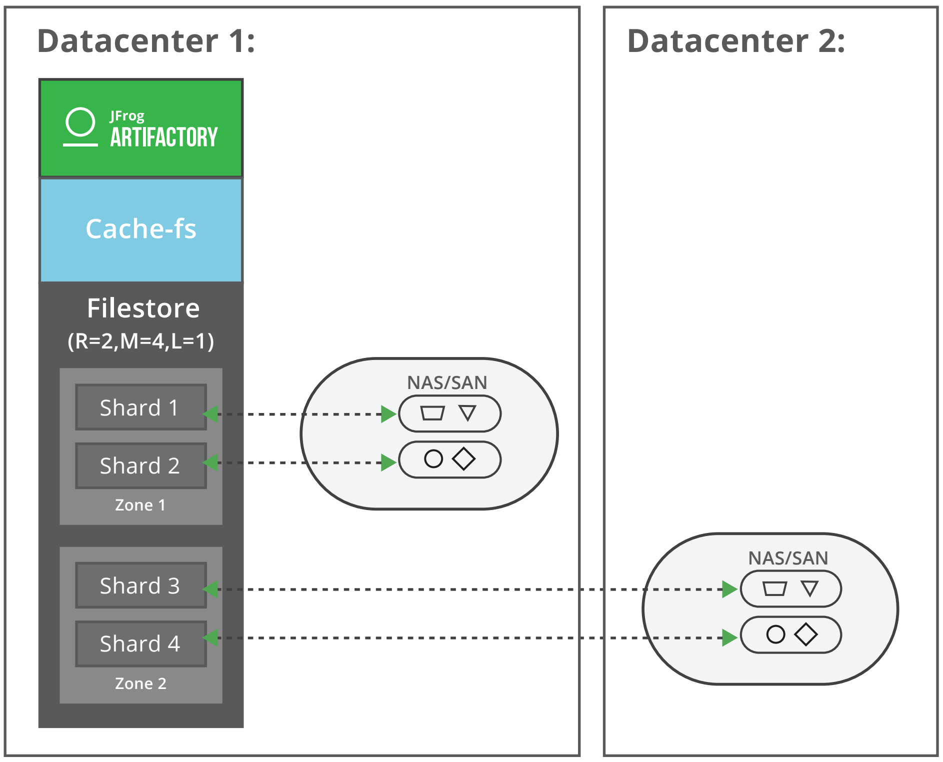
Cluster: Redundancy = 1, Mount (Shard) = 2 , LenientLimit = 1



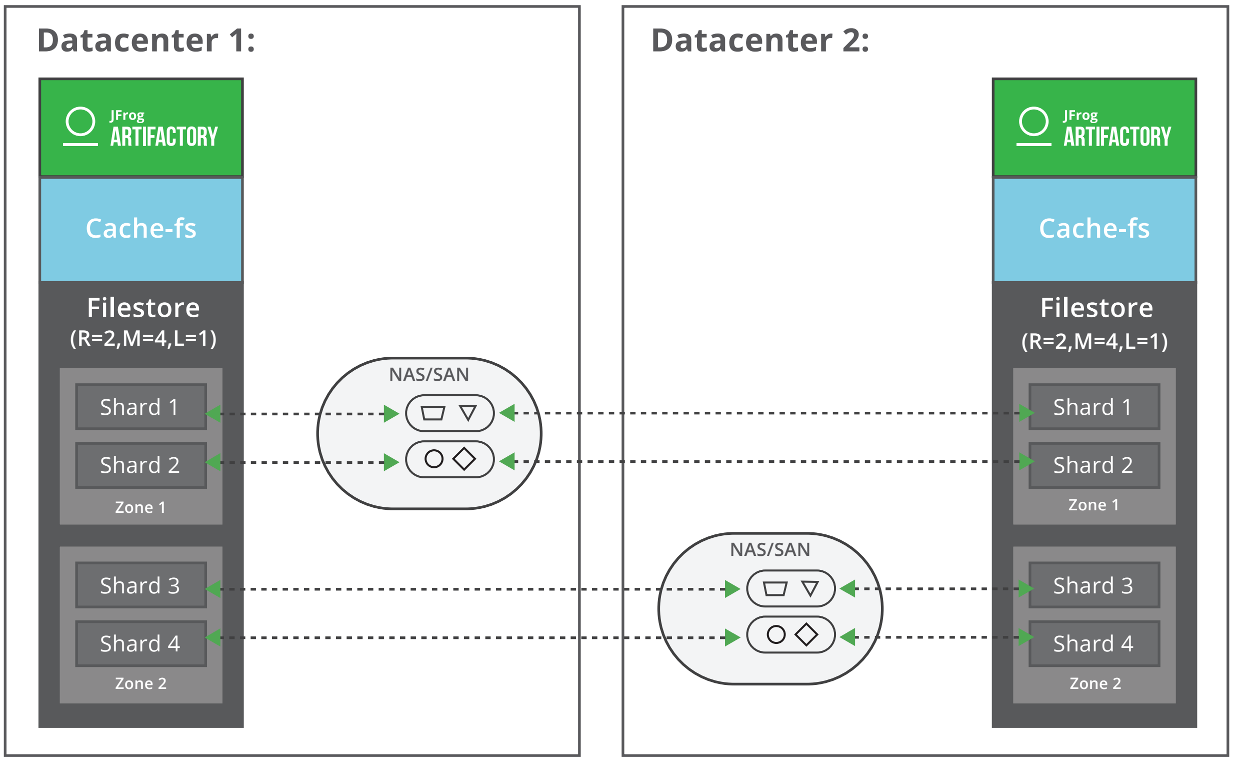
The higher the redundancy, the lower the average retrieval ("READ") time will be.
Yet, higher redundancy will increase the duration of the artifact upload ("WRITE") transaction on the storage which depends as well on the number of shards. Obviously, The redundancy level will also affect the total storage required for Artifactory.

By default, the filestore sharding is configured to serve shards in a single zone, but you can extend it to multiple ones by assigning shards to a specific zone. With this implementation you can also cover the use case where your IT teams do not manage the redundancy across data center/site/zone.

Standalone: Redundancy = 2, Mount (Shard) = 4, LenientLimit = 1



Cluster: Redundancy = 2, Mount (Shard) = 4, LenientLimit = 1



This configuration can apply to both standalone and cluster. All the mounts should be available for all Artifactory nodes.

Warning: Beware of the latency between your zones as writing the copies of the binary will be a single transaction.

Use Case 3: Restriction on the storage management

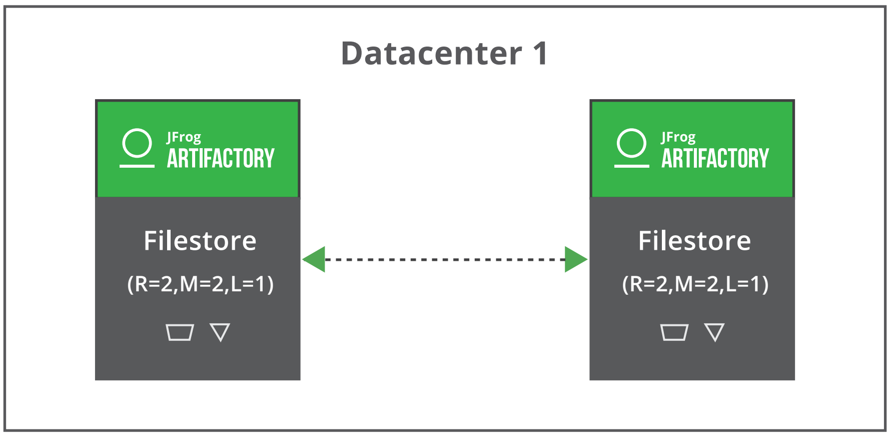
Your IT teams have restrictions regarding the number of mount points per server or do not provide the right storage SLA for your Artifactory, and you prefer handling the availability of your data.

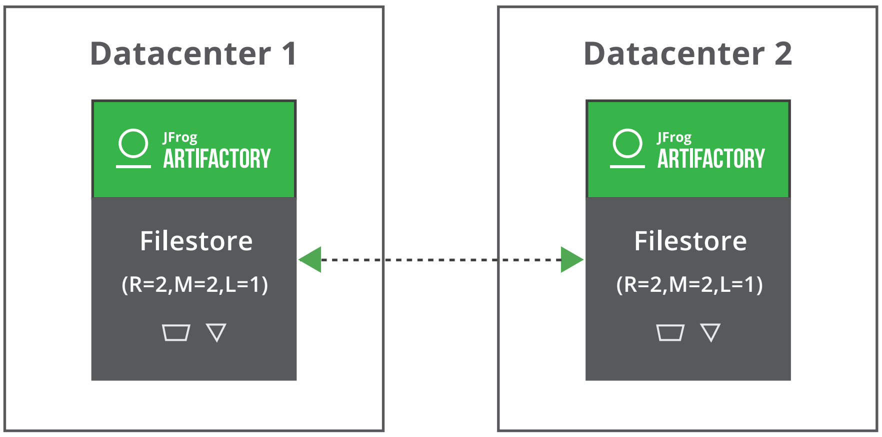
Recommended templates: filesystem, cache-fs and cluster-fs

In a standalone configuration, you would rely on the filesystem or cache-fs.
For more details, you can refer to use case 1.

In an Artifactory cluster configuration, the cluster-fs template can be used and each Artifactory instance will have its own local filestore (also called shard) which will be pointing to an NFS mount, mounted SAN volume or even local storage. The Artifactory cluster will manage this "super filestore" made of several shards. The cache-fs will be activated on each Artifactory instance. By default, this template is using a redundancy of 2, meaning that in a 2 node HA cluster, each shard will contain the full filestore. This guarantees no impact in the possibility a site goes down.
The scaling process can be done vertically by increasing the disk size or horizontally by adding more Artifactory nodes to the cluster.

Cluster: Redundancy = 2, Mount (Shard) = 2 (filestores hosted on attached disks), LenientLimit = 1





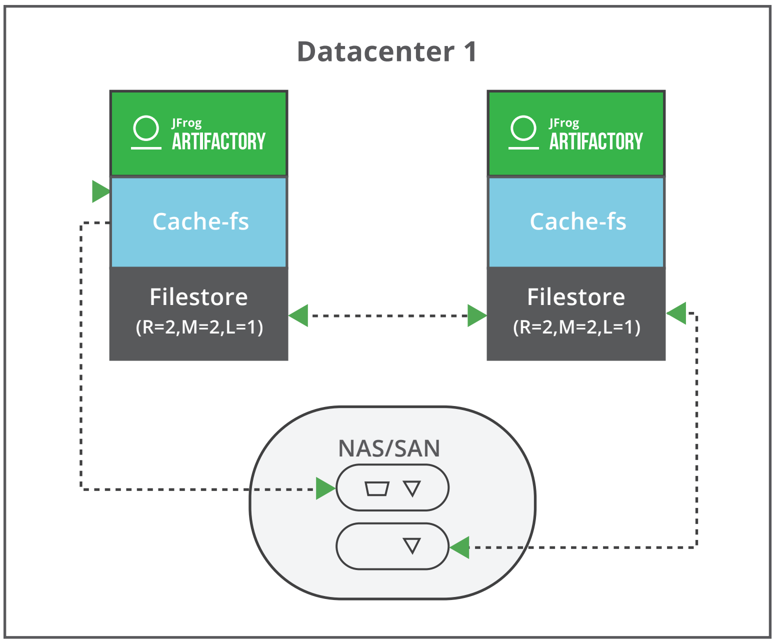
Warning: Make sure your network topology is compliant to our recommendations when setting up your Artifactory HA cluster

The Sharding cluster provider inherits optimization from the Sharding provider, used by the filestore sharding, such as the lenient limit and concurrent streaming to copy a binary into multiple shards

The Optimize Storage Rest API allows you to manually apply a new redundancy or re-apply the current one after shard failures. In both cases, redundancy has to be higher than 1. The process will be executed after the next Garbage collector run.

The maxBalancingRunTime parameter defines the duration of the recovery system after each Garbage collector run.

Cluster: Redundancy = 2, Mount (Shard) = 2 (filestore hosted on remote storage), LenientLimit = 1



The higher the redundancy, the lower the average retrieval (“READ”) time will be. Yet, higher redundancy will increase the duration of the artifact upload ("WRITE") transaction on the storage which depends as well on the number of shards. Obviously, The redundancy level will also affect the total storage required for Artifactory.

The cluster-fs template does not support multiple zones.

The cluster-fs template does not support multiple mount points per Artifactory instance

2. Cloud File Storage

Artifactory filestore can also be integrated in a cloud environment in the same way you would do it in your own data centers. As the purpose is to delegate hardware maintenance and availability of your binaries, you might want to switch from a traditional file storage to an object storage which is also supported (see the next section). This section will cover the case where you want to stick to a file storage in a cloud environment.

Recommended templates: cache-fs and cluster-fs

The general best practices can be applied for both on-prem and cloud environment.

Enable the cache-fs and install it on a disk with an appropriate IOPS based on your needs and budget. If the read cache is vital for your use case, store it in a persistent disks. Your SLA will judge if you can trade off performance over resilience.

Major cloud providers also provide auto extendable shared disks (EFS for AWS, Azure files, Google cloud filestore) where you could store your binaries. Although they offer good performance and scalability, they’re generally much more expensive than managing a persistent disks (EBS for AWS, Azure managed disk, Google persistent disks) or using an object storage.

Using Amazon Elastic Filesystem (EFS) with Artifactory HA
Before using AWS EFS, read more about how bursting mode can impact your Artifactory performance and recommended best practices.

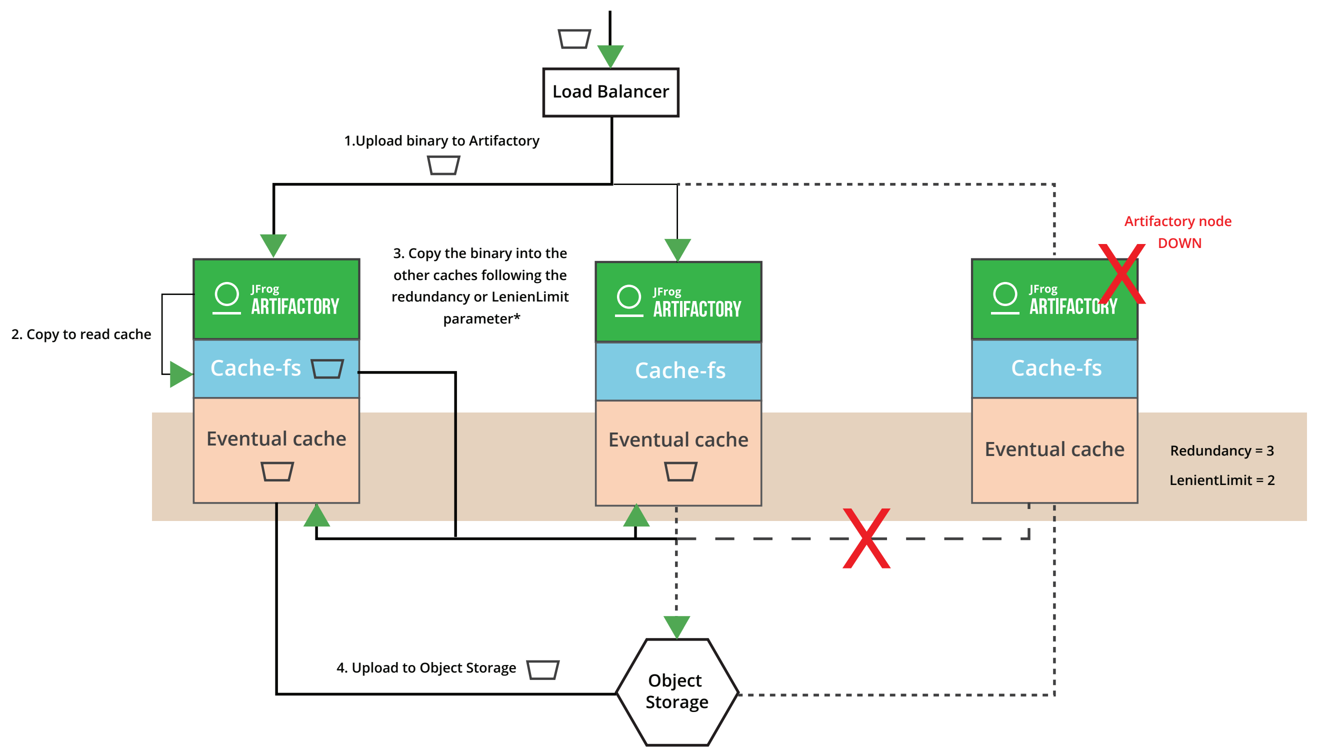
3. Object Storage

Recommended templates: cluster-s3-storage-v3 cluster-azure-blob-storage and cluster-google-storage

Object storage has been popularized with the emergence of cloud providers and can be anot her solution for your IT teams to easily scale out the storage capacity onto different data centers. Artifactory can also rely on this type of storage and implement specific features to fully support it like a retrial mechanisms for reads/writes and a read cache and a redundant “Write cache” (see Caches section) to reduce the latency from remote storage and enables asynchronous uploads. Several filestore templates are available for specific cloud providers and for any object storage supporting the S3 protocol.

The following diagram explains how these templates take advantage of the caches to prevent data loss and optimize the upload process. The write/eventual cache relies on sharding-cluster provider, used by the cluster-fs template (see P12) to ensure the availability of a binary during the upload process to the object storage.

Cluster: Redundancy = 3 (in eventual cache), LenientLimit = 2 (in eventual cache)


Load Balancer
1.Upload binary to Artifactory
2. Copy to read cache
3. Copy the binary into the other caches following the redundancy or LenienLimit parameter*
Artifactory node DOWN
JFrog ARTIFACTORY
Cache-fs
Eventual cache
Redundancy = 3
LenientLimit = 2
4. Upload to Object Storage
Object Storage

Write/Eventual caches are redundant, read caches/cache-fs are not.

The write/eventual cache relies on the same mechanism as the cluster-fs (sharding-cluster provider) to ensure the availability of a binary during the upload process to the object storage. This is achieved by spreading copies of a binary to several eventual caches (redundancy parameter). The lenient limit is another parameter which specify the minimum of successful writes to perform.

In case the upload to the Object storage fails, a retrial mechanism will be fired and also backed by a continuous scanning on the eventual cache. While

the object storage isn't available, Artifactory will use the cache-fs as the "source" of binaries. Keep it mind the cache-fs has a maximum size and keeps only the most recent downloaded/uploaded binaries.

Each “cluster” template has a “no-cluster” implementation where eventual caches are not redundant / synchronized.

“Cluster” template name	“No cluster” template name
cluster-s3-storage-v3	s3-storage-v3
cluster-azure-storage	azure-storage
cluster-google-storage	google-storage

These “no-cluster” templates initially cover a standalone configuration. In an HA cluster configuration, the eventual cache has to be redundant to prevent any binary loss. Such a loss can happen at the intermediate stage when an Artifactory node has completed storing a binary in its cache, and started uploading it to the object storage, and during this process it goes down. The simplest solution to avoid this situation, is to rely on a shared storage. With the “cluster” templates, this case is fully managed by Artifactory by setting a redundancy parameter to spread copies of a same binary to multiple eventual caches.

4. Database

By default, Artifactory separates the binaries from their metadata. Binaries are stored in the filestore and their metadata is stored in a database. While it’s possible to store the binaries in the database as a BLOB, it is not recommended because databases have limitations when storing large files and this can lead to performance degradation and negatively impact operational costs.

DATA RECOVERY STRATEGIES

The trash can should be the first option for data recovery as, by default, any deleted artifact will end up there. You can modify the retention period in the trash can which is set to 14 days. Note, this should be taken care of as part of the overall storage allocation when you do your storage sizing.

The data recovery you choose will depend on the size of your filestore, your Recovery Point Objective, and Recovery Time Objective requirements for Artifactory.

- **Filestore under 1TB not using Object Storage:**
  You can use the Artifactory built-in backup system which will aggregate the binaries, their metadata and the general configuration on the Artifactory host server. In a cluster setup, the backup will be generated by the node performing the backup task (by default the primary node).

- **Filestore under 1TB using Object Storage or Filestore above 1TB using Object Storage or not:**
  The built-in backup might not respect your SLA regarding RTO and it is advised to manage the backup strategy based on database backups/snapshot synchronized with filestore backups which could depend on services (S3 versioning for instance) or 3rd party IT tool/services. Remember to backup the configuration folder

By default, the backup folder will be saved locally in $ARTIFACTORY_HOME/backup. Make sure the storage is at least 3 times bigger than your current filestore as the backup will not benefit from the checksum optimization and will contain all the metadata stored in the database. You can also store it into a specific volume (dedicated disk) to ensure that in case of insufficient space, this will not affect Artifactory.

If you decide to perform backups on a shared mount, verify that disk has the highest possible IOPS and the low latency (remote storage in the same datacenter).

To reduce the RPO, you would also rely on a Disaster Recovery/Mirror site which could be passive (Cold DR) to reload your backups or active where you would take advantage of database and storage replication.

Learn more > about best practices for managing your backups and disaster recovery

CONCLUSION

Artifactory is an optimized binary manager that focuses on managing, storing and retrieving your binaries in the most efficient way thanks to its checksum based storage and read and write caches. In this white paper, we covered the different storage configurations based on infrastructure and storage requirements. As the source of your binaries, a backup and recovery strategy should be implemented based on your own SLA.

This white paper is intended to help you better understand your Artifactory storage setup and configuration according to your needs. Hopefully it will allow you to use Artifactory as your main source of binaries and leverage it to its full potential.