Filestore Sharding

JFrog Subscription Levels

SELF-HOSTED

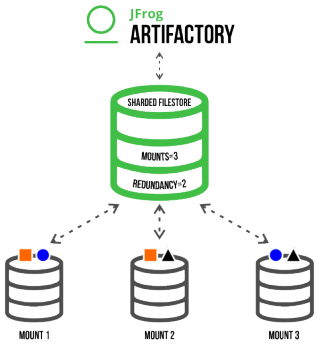
ENTERPRISE
ENTERPRISE+

Page Contents

Overview

From version 4.6, Artifactory offers a Sharding Binary Provider that lets you manage your binaries in a sharded filestore. A sharded filestore is one that is implemented on a number of physical mounts (M), which store binary objects with redundancy (R), where R <= M.

For example, the diagram below represents a sharded filestore where M=3 and R=2. In other words, the filestore consists of 3 physical mounts which store each binary in two copies.



Artifactory's sharding binary provider presents several benefits:

Unmatched stability and reliability

Thanks to redundant storage of binaries, the system can withstand any mount going down as long as M >= R.

Unlimited scalability

If the underlying storage available approaches depletion, you only need to add another mount; a process that requires no downtime of the filestore. Once the mount is up and running, the system regenerates the filestore redundancy according to configuration parameters you control.

> **For a standalone Artifactory setup**
> A restart is required to apply the changes to your *binarystore.xml/ settings*.
>
> **For a High Availablity setup**
> Restarting each cluster separately to apply the changes to your *binarystore.xml/* settings will result in no downtime.

Filestore performance optimization

Sharding Binary Provider offers several configuration parameters that allow you to optimize how binaries are read from or written to the filestore according to your specific system's requirements.

> **Enterprise license required**
>
> Sharded filestore is available for Artifactory installations activated with an enterprise license.

Configuring a Sharding Binary Provider

A sharding binary provider is a binary provider as described in Configuring the Filestore. Basic sharding configuration is used to configure a sharding binary provider for Artifactory instance.

Basic Sharding Configuration

The following parameters are available for a basic sharding configuration:

readBehavior	This parameter dictates the strategy for reading binaries from the mounts that make up the sharded filestore. Possible values are: **roundRobin (default):** Binaries are read from each mount using a round robin strategy.
writeBehavior	This parameter dictates the strategy for writing binaries to the mounts that make up the sharded filestore. Possible values are: **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.
redundancy	Default: r=1 The number of copies that should be stored for each binary in the filestore. Note that redundancy must be less than or equal to the number of mounts in your system for Artifactory to work with this configuration.
lenientLimit	Default: 1 (From version 5.4. Note that for filestores configured with a custom chain and not using the built-in templates the default value of the `lenientLimit` parameter is 0 to maintain consistency with previous versions.) The minimum number of successful writes that must be maintained for an upload to be successful. The next balance cycle (triggered with the GC mechanism) will eventually transfer the binary to enough nodes such that the redundancy commitment is preserved. In other words, Leniency governs the minimal allowed redundancy in cases where the redundancy commitment was not kept temporarily. For example, if `lenientLimit` is set to 3, my setup includes 4 filestores, and 1 of them goes down, writing will continue. If a 2nd filestore goes down, writing will stop. The amount of currently active nodes must always be greater or equal than the configured lenientLimit. If set to 0, the redundancy value has to be kept.
concurrentStreamWaitTimeout	Default: 30,000 ms To support the specified redundancy, accumulates the write stream in a buffer, and uses “r” threads (according to the specified redundancy) to write to each of the redundant copies of the binary being written. A binary can only be considered written once all redundant threads have completed their write operation. Since all threads are competing for the write stream buffer, each one will complete the write operation at a different time. This parameter specifies the amount of time (ms) that any thread will wait for all the others to complete their write operation. If a write operation fails, you can try increasing the value of this parameter.
concurrentStreamBufferKb	Default: 32 Kb The size of the write buffer used to accumulate the write stream before being replicated for writing to the “r” redundant copies of the binary. If a write operation fails, you can try increasing the value of this parameter.
maxBalancingRunTime	Default: 3,600,000 ms (1 hour) Once a failed mount has been restored, this parameter specifies how long each balancing session may run before it lapses until the next Garbage Collection has completed. For more details about balancing, please refer to Using Balancing to Recover from Mount Failure. To restore your system to full redundancy more quickly after a mount failure, you may increase the value of this parameter. If you find this causes an unacceptable degradation of overall system performance, you can consider decreasing the value of this parameter, but this means that the overall time taken for Artifactory to restore full redundancy will be longer.
freeSpaceSampleInterval	Default: 3,600,000 ms (1 hour) To implement its write behavior, Artifactory needs to periodically query the mounts in the sharded filestore to check for free space. Since this check may be a resource intensive operation, you may use this parameter to control the time interval between free space checks. If you anticipate a period of intensive upload of large volumes of binaries, you can consider decreasing the value of this parameter in order to reduce the transient imbalance between mounts in your system.

minSpareUploaderExecutor	Default: 2 Artifactory maintains a pool of threads to execute writes to each redundant unit of storage. Depending on the intensity of write activity, eventually, some of the threads may become idle and are then candidates for being killed. However, Artifactory does need to maintain some threads alive for when write activities begin again. This parameter specifies the minimum number of threads that should be kept alive to supply redundant storage units.
uploaderCleanupIdleTime	Default: 120,000 ms (2 min) The maximum period of time threads may remain idle before becoming candidates for being killed.

Example 1

The code snippet below is a sample configuration for the following setup:

- A cached sharding binary provider with three mounts and redundancy of 2**.**
- Each mount "X" writes to a directory called /**filestoreX.**
- The read strategy for the provider is **roundRobin**.
- The write strategy for the provider is **percentageFreeSpace**.

```
<config version="4">
   <chain>
       <provider id="cache-fs" type="cache-fs">                                        <!-- This is a
cached filestore -->
           <provider id="sharding" type="sharding">                                       <!-- This is a
sharding provider -->
               <sub-provider id="shard1" type="state-aware"/>               <!-- There are three mounts -->
               <sub-provider id="shard2" type="state-aware"/>
               <sub-provider id="shard3" type="state-aware"/>
           </provider>
       </provider>
   </chain>

// Specify the read and write strategy and redundancy for the sharding binary provider
  <provider id="sharding" type="sharding">
       <readBehavior>roundRobin</readBehavior>
       <writeBehavior>percentageFreeSpace</writeBehavior>
       <redundancy>2</redundancy>
 </provider>


//For each sub-provider (mount), specify the filestore location
   <provider id="shard1" type="state-aware">
       <fileStoreDir>filestore1</fileStoreDir>
   </provider>

   <provider id="shard2" type="state-aware">
       <fileStoreDir>filestore2</fileStoreDir>
   </provider>

   <provider id="shard3" type="state-aware">
       <fileStoreDir>filestore3</fileStoreDir>
   </provider>
</config>
```

Example 2

The following code snippet shows the "double-shards" template which can be used as is for your binary store configuration.

```
<config version="4">
        <chain template="double-shards" />

        <provider id="shard-fs-1" type="state-aware">
                <fileStoreDir>shard-fs-1</fileStoreDir>
        </provider>

        <provider id="shard-fs-2" type="state-aware">
                <fileStoreDir>shard-fs-2</fileStoreDir>
        </provider>
</config>
```

The double-shards template uses a cached provider with two mounts and a redundancy of 1, i.e. only one copy of each artifact is stored.

```
<chain>
        <provider id="cache-fs" type="cache-fs">
                <provider id="sharding" type="sharding">
                        <redundancy>1</redundancy>
                        <sub-provider id="shard-fs-1" type="state-aware"/>
                        <sub-provider id="shard-fs-2" type="state-aware"/>
                </provider>
        </provider>
</chain>
```

To modify the parameters of the template, you can change the values of the elements in the template definition. For example, to increase redundancy of the configuration to 2, you only need to modify the `<redundancy>` tag as shown below.

```
<chain>
        <provider id="cache-fs" type="cache-fs">
                <provider id="sharding" type="sharding">
                        <redundancy>2</redundancy>
                        <sub-provider id="shard-fs-1" type="state-aware"/>
                        <sub-provider id="shard-fs-2" type="state-aware"/>
                </provider>
        </provider>
</chain>
```

Cross-Zone Sharding Configuration

Sharding across multiple zones in an HA Artifactory cluster allows you to create zones or regions of sharded data to provide additional redundancy in case one of your zones becomes unavailable. You can determine the order in which the data is written between the zones and you can set the method for establishing the free space when writing to the mounts in the neighboring zones.

The following parameters are available for a cross-zone sharding configuration in the **binarystore.xml** file:

readBehavior	This parameter dictates the strategy for reading binaries from the mounts that make up the cross-zone sharded filestore. Possible value is: **zone:** BInaries are read from each mount according to zone settings.
writeBehavior	This parameter dictates the strategy for writing binaries to cross-zone sharding mounts: Possible values are: **zonePercentageFreeSpace:** Binaries are written to the mount in the relevant zone with the highest percentage of free space available. **zoneFreeSpace:** Binaries are written to the mount in the zone with the greatest absolute volume of free space available.

Add to the Artifactory System YAML file

The following parameters are available for a cross-zone sharding configuration in the Artifactory System YAML file:

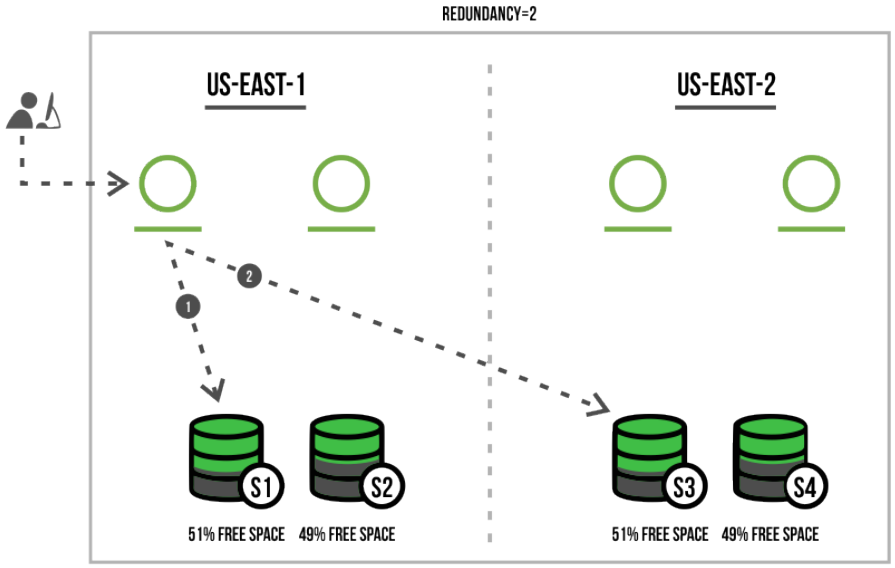
node.id	Unique descriptive name of this server. **Uniqueness** Make sure that each node has an id that is unique on your whole network.
node.crossZoneOrder	This parameter sets the zone order in which the data is written to the mounts. In the following example, *crossZoneOrder: "us-east1*,us-east2", the sharding will write to the US-EAST-1 zone and then to the US-EAST-2 zone.

You can dynamically add nodes to an existing sharding cluster using the Artifactory System YAML file. To do so, you will need your cluster to already be configured with sharding, and by adding the 'crossZoneOrder: us-east-1,us-east-2' property, the new node will be able to write to the existing cluster nodes without changing the binarystore.xml file.

Example:

This example displays a cross-zone sharding scenario in which the Artifactory cluster is configured with a redundancy of 2 and includes the following steps:

1. The developer first deploys the package to the closest Artifactory node.
2. The package is then automatically deployed to the 'US-EAST-1" zone to the shard with the highest percentage of free space in the "S1" shard (with 51% free space).
3. The package is deployed using the same method to the "S3" shard, that also has the highest percentage of free space in the 'US-EAST-2' zone.



The code snippet below is a sample configuration of our cross-zone setup:

- 1 Artifactory cluster across 2 zones: "us-east-1" and "us-east-2" in this order.
- 4 HA nodes, 2 nodes in each zone.
- 4 mounts (shards), 2 mounts in each zone.
- The write strategy for the provider is **zonePercentageFreeSpace**.

Example: Cross-zone sharding configuration in Artifactory System YAML

```
node:
  id: "west-node-1"
  crossZoneOrder: "us-east-1,us-east-2"
```

Example: Cross-zone sharding configuration in the binarystore.xml

```
<config version="4">
        <chain>
                <provider id="sharding" type="sharding">
                          <sub-provider id="shard1" type="state-aware"/>
                       <sub-provider id="shard2" type="state-aware"/>
                       <sub-provider id="shard3" type="state-aware"/>
                       <sub-provider id="shard4" type="state-aware"/>
                </provider>
        </chain>

        <provider id="sharding" type="sharding">
                <redundancy>2</redundancy>
                <readBehavior>zone</readBehavior>
                <writeBehavior>zonePercentageFreeSpace</writeBehavior>
        </provider>

        <provider id="shard1" type="state-aware">
                <fileStoreDir>mount1</fileStoreDir>
                <zone>us-east-1</zone>
        </provider>

        <provider id="shard2" type="state-aware">
                <fileStoreDir>mount2</fileStoreDir>
                <zone>us-east-1</zone>
        </provider>

        <provider id="shard3" type="state-aware">
                <fileStoreDir>mount3</fileStoreDir>
                <zone>us-east-2</zone>
        </provider>

        <provider id="shard4" type="state-aware">
                <fileStoreDir>mount4</fileStoreDir>
                <zone>us-east-2</zone>
        </provider>
</config>
```

Using Balancing to Recover from Mount Failure

In case of a mount failure, the actual redundancy in your system will be reduced accordingly. In the meantime, binaries continue to be written to the remaining active mounts. Once the malfunctioning mount has been restored, the system needs to rebalance the binaries written to the remaining active mounts to fully restore (i.e. balance) the redundancy configured in the system. Depending on how long the failed mount was inactive, this may involve a significant volume of binaries that now need to be written to the restored mount, which may take significant amount of time. Since restoring the full redundancy is a resource intensive operation, the balancing operation is run in a series of distinct sessions until complete. These are automatically invoked after a Garbage Collection process has been run in the system.

Restoring Balance in Unbalanced Redundant Storage Units

In the case of voluntary actions that cause an imbalance the system redundancy, such as when doing a filestore migration, you may manually invoke rebalancing of redundancy using the Optimize System Storage REST API endpoint. Applying this endpoint raises a flag for Artifactory to run rebalancing following the next Garbage Collection. Note that, to expedite rebalancing, you can invoke garbage collection manually from the Artifactory UI.

Optimizing System Storage

Artifactory REST API provides an endpoint that allows you to raise a flag to indicate that Artifactory should invoke balancing between redundant storage units of a sharded filestore after the next garbage collection. For details, please refer to Optimize System Storage.