Configuring the Filestore

Overview

From version 4.6, JFrog Artifactory offers flexible filestore management that is configurable to meet a variety of needs in terms of binary storage providers, storage size, and redundancy. Not only are you now able to use different storage providers, but you can also chain a series of providers together to build complex structures of binary providers and support seamless and unlimited growth in storage.

To support flexible filestore management, Artifactory introduces a new configuration file, *`binarystore.xml`*, located in the *`$ARTIFACTORY_HOME/etc`* folder. Through a simple modification of this file you can implement a variety of different binary storage configurations. The *`binarystore.xml`* file replaces the binary storage parameters in the *`storage.properties`* file. While *`storage.properties`* is still supported to manage your filestore, to utilize all the features of flexible filestore management, you need to add the new *`binarystore.xml`* file to your system.

Chains and Binary Providers

The *`binarystore.xml`* file specifies a chain with a set of binary providers. A binary provider represents a type of object storage feature such as "cached filesystem". Binary providers can be embedded into one another to form chains that represent a coherent filestore. Artifactory comes with a set of built-in set of chains that correspond to the binary.provider.type parameter that was used in previous versions of Artifactory. Therefore, the built-in set of chains available in Artifactory are:

- file-system
- cache-fs
- full-db
- full-db-direct
- s3
- s3Old
- google-storage
- double-shards
- redundant-shards

In addition, Artifactory allows you to set up your filestore in any way needed by defining a custom chain.

Page Contents

Configuring a Built-in Filestore

To configure Artifactory to use one of the built-in filestores, you only need some basic configuration elements.

Basic Configuration Elements

For basic filestore configuration, the *`binarystore.xml`* file is quite simple and contains the basic tags or elements that are described below along with the attributes that they may include:

config tag

The `<config>` tag specifies a filestore configuration. It includes a `version` attribute to allow versioning of configurations.

```
<config version="v1">
…
</config>
```

chain element

The config tag contains a `chain` element that that defines the structure of the filestore. To use one of the built-in filestores, the chain element needs to include the corresponding template attribute. For example, to use the built-in basic “file system” template, all you need is the following configuration:

```
<config version="v1">
        <chain template="file-system">
        </chain>
</config>
```

Built-in Templates

The following sections describe the basic chain templates come built-in with Artifactory and are ready for you to use out-of-the-box, as well as other binary providers that are included in the default chains .

file-system	The most basic filestore configuration for Artifactory used for a local or mounted filestore.
cache-fs	Works the same way as filesystem but also has a binary LRU (Least Recently Used) cache for upload/download requests. Improves performance of instances with high IOPS (I/O Operations) or slow NFS access.
full-db	All the metadata and the binaries are stored as BLOBs in the database with an additional layer of caching.
full-db-direct	All the metadata and the binaries are stored as BLOBs in the database without caching.
s3	This is the setting used for S3 Object Storage using the JetS3t library.
s3Old	This is the setting used for S3 Object Storage using JCloud as the underlying framework.
google-storage	This is the setting used for Google Cloud Storage as the remote filestore.
double-shards	A pure sharding configuration that uses 2 physical mounts with 1 copy (which means each artifact is saved only once).
redundant-shards	A pure sharding configuration that uses 2 physical mounts with 2 copies (which means each shard stores a copy of each artifact).

Modifying an Existing Filestore

To accommodate any specific requirements you may have for your filestore, you may modify one of the existing chain templates either by extending it with additional binary providers or by overriding one of its attributes. For example, the built-in filesystem chain template stores binaries under the $*ARTIFACTORY_HOME/data/filestore. directory. To modify the template so that it stores binaries under $FILESTORE/binaries you could extend it as follows:*

```
<config version="v1">
        <chain template="file-system">                                               <!-- Use the "file-system"
template -->
        </chain>

        <provider id="file-system" type="file-system">                                          <!-- Modify the
"file-system" binary provider -->
                        <fileStoreDir>$FILESTORE/binaries</fileStoreDir>                <!-- Override the
<fileStoreDir> attribute -->
        </provider>
</config>
```

Configuring a Custom Filestore From Scratch

In addition to the built-in filestore chain templates, you may construct your own chain template to accommodate any filestore structure you need. To construct a custom filestore from scratch, you need to be familiar with the different binary provider types you can work with as defined in the `provider` tag which includes the following attributes:

id	A logical name for the provider
type	Defines a fundamental feature of the filestore as follows: • **file-system:** files are stored in the filesystem • **cache-fs:** files are stored in the filesystem with a caching • **blob:** files are stored in a database as blobs • **eventual:** files are initially stored in a cache, and eventually are moved to persistent storage • **retry:** if an attempt to upload files to persistent storage fails, try again. • **s3:** files are uploaded to an S3 compliant object store using JetS3t as the framework • **S3Old:** files are uploaded to an S3 compliant object store using JClouds as the framework • **google-storage:** files are uploaded to Google Cloud Storage • **sharding:** files are stored in a sharded filestore

Binary providers in a chain must be compatible

Binary providers are not always compatible with each other. You need to make sure that the combination of binary providers you choose creates a coherent and viable filestore. For example you cannot combine an **S3** provider (which wants to store files on an S3 object store) with a **filestore** provider which contradicts and wants to store files on the file system.

Setting Up the Custom Filestore

To set up a custom filestore, you need to be familiar with **sub-providers** and understand how they differ from **providers**

As described before, your overall filestore is defined by a chain of providers that specify the hierarchy of actions that are taken when you need to read or write a file. The important notion with providers is that they are invoked as a hierarchy, one after the other, through the chain. For example, for a `cache-fs` provider chained to a `fulldb` provider, when reading a file, you would first try to read it from the cache. You would only move on to extract the file from the database if it is not found in the cache.

A sub-provider will normally be defined with a set of sibling sub-providers. All the sub-providers are in the same hierarchy in the chain and are accessed in parallel. The classical example is sharding in which several of the sub-providers may be accessed in parallel for each write to implement redundancy.

To summarize, providers are accessed sequentially according to their location in the chain hierarchy; sub-providers are accessed in parallel and are all in the same level of the hierarchy.

Example:

The following snippet is an example of a customized binary provider based on the S3 default chain.

```
<!-- S3 chain template structure  -->
<chain>
 <provider id="cache-fs" type="cache-fs">                  <!--It first tries to read from the cache -->
       <provider id="eventual" type="eventual">          <!--It is eventually persistent so writes are also
written directly to persistent storage -->
           <provider id="retry" type="retry">            <!-- If a read or write fails, retry -->
               <provider id="s3" type="s3"/>             <!-- Actual storage is S3 -->
           </provider>
       </provider>
   </provider>
</chain>

<provider id="cache-fs" type="cache-fs">
        <maxCacheSize>10000000000</maxCacheSize>          <!-- The maximum size of the cache in bytes --
>
    <cacheProviderDir>cache</cacheProviderDir>  <!-- The cache -->
</provider>

<provider id="eventual" type="eventual">
        <numberOfThreads>20</numberOfThreads>                 <!-- The maximum number of threads for parallel
upload of files -->
</provider>

<provider id="retry" type="retry">
    <maxTrys>10</maxTrys>                                                 <!-- Try any read or write a maximum
of 10 times -->
</provider>

<provider id="s3" type="s3">
    <identity>test</identity>                                      <!-- Credentials and endpoint for your
Amazon S3 storage -->
    <credential>test</credential>
    <endpoint>s3.amazonaws.com</endpoint>
        <bucketName>bucket-name</bucketName>
</provider>
</config>
```

Built-in Chain Templates

Artifactory comes with a set of chain templates built-in allowing you to set up a variety of different filestores out-of-the-box. However, to override the built-in filestores, you need to be familiar with the attributes available for each binary provider that is used in them. These are described in the following sections which also show the basic configuration and a usage example

Filesystem Binary Provider

This is the basic filestore configuration for Artifactory and is used for a local or mounted filestore.

id	file-system
baseDataDir	Default: *$ARTIFACTORY_HOME/data* The root directory where Artifactory should store data files.
fileStoreDir	Default: *filestore* The root folder of binaries for the filestore. If the value specified starts with a forward slash (“/”) the value is considered the fully qualified path to the filestore folder. Otherwise, it is considered relative to the **baseDataDir**.
tempDir	Default: *temp* A temporary folder under **baseDataDir** into which files are written for internal use by Artifactory. This must be on the same disk as the **fileStoreDir**.

```
<!-- Example -->
<!-- In this example, the filestore and temp folder are located under the root directory of the machine -->
<config version="v1">
<chain template=”file-system”>
</chain>
<provider id="file-system" type="file-system">
        <fileStoreDir>/filestore</fileStoreDir>
        <tempDir>/temp</tempDir>
</provider>
</config>
```

The above "file-system" chain template is an implicit configuration of the following chain definition:

```
<!-- Template chain configuration for 'file-system' -->
<chain>
<provider id="file-system" type="file-system"/>
</chain>
```

Cached Filesystem Binary Provider

The `cache-fs` serves has a binary LRU (Least Recently Used) cache for all upload/download requests. This can improve Artifactory's performance since frequent requests will be served from the `cache-fs` (as in case of the S3 binary provider).

The `cache-fs` binary provider will be the closest filestore layer of Artifactory. This means that if the filestore is mounted, we would like the `cache-fs` to be local on the artifactory server itself (if the filestore is local, then cache-fs is meaningless). In the case of an HA configuration, the `cache-fs` will be mounted and the recommendation is for each node to have its own `cache-fs` layer.

id	cache-fs
maxCacheSize	Default: 5000000000 (5GB) The maximum storage allocated for the cache in **bytes**.
cacheProviderDir	Default: cache The root folder of binaries for the filestore cache. If the value specified starts with a forward slash (“/”) the value is considered the fully qualified path to the filestore folder. Otherwise, it is considered relative to the **baseDataDir**.

```
<!-- cahce-fs template configuration -->
<chain>
        <provider id="cache-fs" type="cache-fs">
        <provider id="file-system" type="file-system"/>
        </provider>
 </chain>

<!-- Example -->
<!-- This example sets the cache-fs size to be 10 Gb and its location (absolute path since it starts with a "
/") to be /cache/filestore -->
<config version="v1">
<chain template="cache-fs">

</chain>
<provider id="cache-fs" type="cache-fs">
        <cacheProviderDir>/cache/filestore</cacheProviderDir>
        <maxCacheSize>10000000000</maxCacheSize>
</provider>
</config>
```

Full-DB Binary Provider

This binary provider saves the binary content as blobs in the database.

There are two basic default chains: with Caching that uses cache-fs as a checksum based layer, and without caching

id	blob

```
<!-- Basic template configuration with caching 'full-db'  -->
<chain>
        <provider id="cache-fs" type="cache-fs">
                <provider id="blob" type="blob"/>
        </provider>
</chain>

<!-- Basic configuration without caching 'full-db-direct' -->
<chain>
        <provider id="blob" type="blob"/>
</chain>
```

Eventual Binary Provider

This binary provider is not independent and will always be used as part of a template chain for a remote filestore that may exhibit upload latency (e.g. S3 or GCS). To overcome potential latency, files are first written to a folder called "eventual" under the **baseDataDir** in local storage, and then later uploaded to persistent storage with the cloud provider. The default location of the `eventual` folder is under the `$ARTIFACTORY_HOME/data` folder (or `$CLUSTER_HOME/ha-data` in the case of an HA configuration) and is not configurable. You need to make sure that Artifactory has full read/write permissions to this location.

There are three additional folders under the `eventual` folder:

- _pre: part of the persistence mechanism that ensures all files are valid before being uploaded to the remote filestore
- _add: handles upload of files to the remote filestore
- _delete: handles deletion of files from the remote filestore

id	eventual
timeout	The maximum amount of time a file may be locked while it is being written to or deleted from the filesystem.
dispatchInterval	Default: 5000 ms The interval between which the provider scans the "eventual" folder to check for files that should be uploaded to persistent storage.
numberOfThreads	Default: 5 The number of parallel threads that should be allocated for uploading files to persistent storage.

```
<!-- Example configuration -->
<!-- Configures 10 parallel threads for uploading and a lock timeout of 180 seconds-->
<provider id="eventual" type="eventual">
        <numberOfThreads>10</numberOfThreads>
        <timeout>180000</timeout>
</provider>
```

Retry Binary Provider

This binary provider is not independent and will always be used as part of a more complex template chain of providers. In case of a failure in a read or write operation, this binary provider notifies its underlying provider in the hierarchy to retry the operation.

id	retry
interval	Default: 5000 ms The time interval to wait before retries.
maxTrys	Default: 5 The maximum number of attempts to read or write before responding with failure.

```
<!-- Example configuration -->
<!-- Configures a maximum of 10 retry attempts in case of a failure-->
<provider id="retry" type="retry">
        <maxTrys>10</maxTrys>
</provider>
```

Chaining Eventual and Retry Providers

The eventual and retry providers can be chained to support a remote filestore (since the combination is not needed for a local filestore).

The following example shows a chain for a mounted filesystem.

```
<chain>
        <provider id="cache-fs" type="cache-fs">
                <provider id="eventual" type="eventual">
                        <provider id="retry" type="retry">
                                <provider id="file-system" type="file-system"/>
                                </provider>
                        </provider>
                </provider>
</chain>
```

State-aware Binary Provider

This provider is aware if its underlying disk is functioning or not.

id	state-aware
checkPeriod	Default: 15,000 ms During read and write operations, this binary provider checks that the underlying disk functioning. This parameter specifies the minimum interval between checks.

External File Binary Provider

This binary provider reads binaries from an external directory rather than from the main filestoreDir. This can be useful when migrating your binaries from one filestore to another, or when setting up a new filestore if the current one is full.

This binary provider is always wrapped with an External WrapperExternalWrapperBinaryProvider binary provider which determines what to do on read operations.

id	external-file
externalDir	The external directory from which files are read.

External Wrapper Binary Provider

This provider wraps the External File binary provider to implement different read modes on an External File binary provider. Files are read from the **externalDir** specified in the External File binary provider, and handled according to the **connectMode** specified.

id	external-wrapper

connect Mode	Default: **passThrough** Specifies what to do with the binary file once downloaded from the external directory specified in the External File binary provider. • **passThrough:** When a file is read from the **externalDir**, Artifactory passes it directly to the caller. • **copyOnRead:** When a file is read from the **externalDir**, Artifactory stores a copy in its local filestore. From then on, when the same file is read, it will be supplied from the local filestore. • **move:** When a file is read from the externalDir, Artifactory stores a copy in its local filestore, and deletes it from the **externalDir**. From then on, when the same file is read, it will be supplied from the local filestore.

Google Storage, S3 and S3Old Binary Providers

These three binary providers for cloud storage solutions have a very similar selection of parameters. The main difference between S3 and S3Old is in the underlying framework, where S3 uses JetS3t and S3Old uses JClouds. These providers will typically be wrapped with other binary providers to ensure that the binary resources are always available from Artifactory (for example, to enable Artifactory to serve files when requested even if they have not yet reached the cloud storage due to upload latency).
Requires an enterprise license.

id	google-storage, s3, or s3old respectively
testConnection	Default: true When true, the binary provider uploads and downloads a file when Artifactory starts up to verify that the connection to the cloud storage provider is fully functional.
useSignature	Default: false. Only available for **S3**. When true, requests to AWS S3 are signed. Available from AWS S3 version 4. For details, please refer to Signing AWS API requests in the AWS S3 documentation.
multiPartLimit	Default: 100,000,000 bytes File size threshold over which file uploads are chunked and multi-threaded.
identity	Your cloud storage provider identity.
credential	Your cloud storage provider authentication credential.
region	Only available for **S3** or **S3Old**. The region offered by your cloud storage provider with which you want to work.
bucketName	Your globally unique bucket name.
path	The path to your file within the bucket.
proxyIdentity	Corresponding parameters if you are accessing the cloud storage provider through a proxy server.
proxyCredential	
proxyPort	
proxyHost	
port	The cloud storage provider's port.
endPoint	The cloud storage provider's URL.
roleName	Only available on **S3.** The IAM role configured on your Amazon server for authentication.
refreshCredentials	Default: false. Only available on **S3.** When true, the owner's credentials are automatically renewed if they expire. When **roleName** is used, this parameter **must** be set to true.
httpsOnly	Default: false. Only available on **google-storage** and **S3**. Set to true if you only want to access your cloud storage provider through a secure https connection.
httpsPort	Must be set if **httpsOnly** is **true**. The https port for the secure connection.

providerID	Set to S3. Only available for **S3Old**.
s3AwsVersion	Default: 'AWS4-HMAC-SHA256' (AWS signature version 4). Only available on **S3.** Can be set to 'AWS2' if AWS signature version 2 is needed. Please refer the AWS documentation for more information.
bucketExists	Default: false. Only available on **google-storage**. When true, it indicates to the binary provider that a bucket already exists in Google Cloud Storage and therefore does not need to be created.

S3 Binary Provider

The snippets below show some examples that use the S3 binary provider:

```
<!-- The chain Template for s3-->
<chain>
        <provider id="cache-fs" type="cache-fs">
                <provider id="eventual" type="eventual">
                        <provider id="retry" type="retry">
                                <provider id="s3" type="s3"/>
                        </provider>
                </provider>
        </provider>
</chain>

<!-- Example 1 -->
<!-- A configuration for OpenStack Object Store Swift-->
<config version="v1">
<chain template="s3">
</chain>
<provider id="s3" type="s3">
    <identity>XXXXXXXXX</identity>
    <credential>XXXXXXXX</credential>
    <endpoint><My OpenStack Server></endpoint>
    <bucketName><My OpenStack Container></bucketName>
    <httpsOnly>false</httpsOnly>
    <property name="s3service.disable-dns-buckets" value="true"></property>
</provider>
</config>

<!-- Example 2-->
<!-- A configuration for CEPH -->
<config version="v1">
<chain template="s3">
</chain>
<provider id="s3" type="s3">
        <identity>XXXXXXXXXX</identity>
    <credential>XXXXXXXXXXXXXXXXX</credential>
    <endpoint><My Ceph server></endpoint>
    <bucketName>[My Ceph Bucket Name]</bucketName>
    <httpsOnly>false</httpsOnly>
</provider>
</config>
<!-- Example 3-->
<!-- A configuration for CleverSafe -->
<config version="v1">
<chain template="s3">
</chain>
<provider id="s3" type="s3">
    <identity>XXXXXXXXX</identity>
    <credential>XXXXXXXX</credential>
    <endpoint>[My CleverSafe Server]</endpoint>
    <bucketName>[My CleverSafe Bucket]</bucketName>
    <httpsOnly>false</httpsOnly>
        <property name="s3service.disable-dns-buckets" value="true"></property>
</provider>
</config>

<!-- Example 4 -->
<!-- A configuration for S3 with a proxy between Artifactory and the S3 bucket -->
```

```
<config version="v1">
<chain template="s3">
</chain>
<provider id="s3" type="s3">
    <identity>XXXXXXXXXX</identity>
        <credential>XXXXXXXXXXXXXXXXX</credential>
    <endpoint>[My S3 server]</endpoint>
    <bucketName>[My S3 Bucket Name]</bucketName>
    <proxyHost>[http proxy host name]</proxyHost>
    <proxyPort>[http proxy port number]</proxyPort>
    <proxyIdentity>XXXXX</proxyIdentity>
    <proxyCredential>XXXX</proxyCredential>
</provider>
</config>

<!-- Example 5 -->
<!-- A configuration for AWS using an IAM role instead of an IAM user -->
<config version="v1">
<chain template="s3">
</chain>
<provider id="s3" type="s3">
        <roleName>XXXXXX</roleName>
        <endpoint>s3.amazonaws.com</endpoint>
        <bucketName>[mybucketname]</bucketName>
        <refreshCredentials>true</refreshCredentials>
</provider>
</config>


<!-- Example 6 -->
<!-- A configuration for AWS when using server side encryption-->
<config version="v1">
        <chain template="s3">
                </chain>
        <provider id="s3" type="s3">
            <identity>XXXXXXXXX</identity>
            <credential>XXXXXXXX</credential>
            <endpoint>s3.amazonaws.com</endpoint>
            <bucketName>[mybucketname]</bucketName>
            <property name="s3service.server-side-encryption" value="AES256"></property>
        </provider>
</config>
```

Google Storage Binary Provider

The snippets below show some examples that use the Google Cloud Storage binary provider:

```
<!-- Basic chain template for 'google-storage'  -->
<chain>
        <provider id="cache-fs" type="cache-fs">
                <provider id="eventual" type="eventual">
                        <provider id="retry" type="retry">
                                <provider id="google-storage" type="google-storage"/>
                        </provider>
                </provider>
        </provider>
</chain>

<!-- Example 1 -->
<config version="v1">
<chain template="google-storage">
</chain>
<provider id="google-storage" type="google-storage">
        <endpoint>commondatastorage.googleapis.com</endpoint>
        <bucketName><BUCKET NAME></bucketName>
        <identity>XXXXXX</identity>
        <credential>XXXXXXX</credential>
</provider>
</config>

<!-- Example 2 -->
<!-- A configuration with a dynamic property from the jetS3t library. -->
<!-- In this example, the httpclient.max-connections parameter sets the maximum number of simultaneous
connections to allow globally (default is 100) -->

<config version="v1">
<chain template="google-storage">
</chain>
<provider id="google-storage" type="google-storage">
        <endpoint>commondatastorage.googleapis.com</endpoint>
        <bucketName><BUCKET NAME></bucketName>
        <identity>XXXXXX</identity>
        <credential>XXXXXXX</credential>
        <property name=”httpclient.max-connections” value=150></property>
</provider>
</config>
```

S3Old Binary Provider

The snippets below show some examples that use the S3 binary provider where JClouds is the underlying framework:

```
<!-- Basic chain template for 's3Old' -->
<chain>
        <provider id="cache-fs" type="cache-fs">
                <provider id="eventual" type="eventual">
                        <provider id="retry" type="retry">
                                <provider id="s3Old" type="s3Old"/>
                        </provider>
                </provider>
        </provider>
</chain>

<!-- Example 1 -->
<!-- A configuration AWS -->
<config version="v1">
<chain template="s3Old">
</chain>
<provider id="s3Old" type="s3Old">
        <identity>XXXXXXXXX</identity>
        <credential>XXXXXXXX</credential>
        <endpoint>s3.amazonaws.com</endpoint>
        <bucketName>[mybucketname]</bucketName>
</provider>
</config>
```

Sharding Binary Provider

For examples that use a sharding binary provider, please refer to Configuring a Sharding Binary Provider.

Configuring the Filestore for Older Versions

For versions of Artifactory below 4.6, the filestore used is configured in the *`$ARTIFACTORYe_HOME/etc/storage.properties`* file as follows

binary.provider.type	**filesystem (default)** This means that metadata is stored in the database, but binaries are stored in the file system. The default location is under *`$ARTIFACTORY_HOME/data/filestore`* however this can be modified. **fullDb** All the metadata and the binaries are stored as BLOBs in the database. **cachedFS** Works the same way as *filesystem* but also has a binary LRU (Least Recently Used) cache for upload/download requests. Improves performance of instances with high IOPS (I/O Operations) or slow NFS access. **S3** This is the setting used for S3 Object Storage.
binary.provider.cache.maxSize	This value specifies the maximum cache size (in bytes) to allocate on the system for caching BLOBs.
binary.provider.filesystem.dir	If *`binary.provider.type`* is set to *filesystem* this value specifies the location of the binaries (default: *`$ARTIFACTORY_HOME/data/filestore`*).
binary.provider.cache.dir	The location of the cache. This should be set to your $ARTIFACTORY_HOME directory directly (not on the NFS).