S3 Object Storage

S3 Object Storage Overview

S3 object storage requires JFrog Container Registry, Pro, Enterprise X, or an Enterprise+ license.

Artifactory fully supports S3 object storage for distributed file systems so your Artifactory filestore can reside on the cloud. This presents several benefits:

1. **Unlimited scalability**
   Since your files are now stored on the cloud, this means that your Artifactory filestore is scalable and effectively unlimited (to the extent offered by your storage provider). You may freely continue to upload files without having to install or maintain any file storage devices. You can even upload files larger than 5 GB using multi-part upload.

2. **Security**
   Enjoy the same security and authentication mechanisms provided by your S3 provider.

3. **Disaster recovery**
   Since your files are replicated and stored with redundancy, this offers the capability for disaster recovery.

4. **Support any S3 compliant protocol**
   Artifactory's support is based on the S3 protocol. Any provider that uses S3, such as Ceph, Swift (through the S3 API) and others will also be supported by Artifactory. With support for AWS S3 version 4, you can sign AWS requests using Signature Version 4.

Backup your system. Your current filestore will be deleted.

Setting up Artifactory to use S3 will delete all files in your current filestore.

If you already have a running installation of Artifactory, then before you setup Artifactory to use S3 and migrate your filestore to the cloud, we strongly recommend that you do a complete system backup.

For more information on older S3 filestore implementations, see Other S3 Binary Providers.

Filestore Fundamentals

This page provides you with the information about a specific binary provider. For more information on filestores and the various filestores that you can use, see Configuring the Filestore.

Page Contents

- S3 Object Storage Overview
- Setting up Artifactory to Use S3
  - Setting Your License
  - Configuring Artifactory to Use S3
    - Direct (Eventual-less) versus Eventual Upload Mechanism
    - Direct Upload Mechanism (Recommended)
    - Eventual Upload Mechanism
- Amazon S3 Official SDK Template
  - Authentication Mechanism
  - S3 Binary Storage Template Configuration
    - S3 Direct Upload Template (Recommended)
- Amazon S3 SDK Cluster Binary Provider
  - cluster-s3-storage-v3 template configuration
  - What's in the template?
- S3 Sharding
  - S3-sharding Template
  - State-Aware-S3 Binary Provider
  - S3 Sharding Example 1
  - S3 Sharding Example 2
  - S3 Sharding Example 3
- Migrating Your Filestore to S3
  - Automatic Filestore Migration (Recommended)
  - Manual Filestore Migration

Setting up Artifactory to Use S3

First time installation or upgrade

If you are moving your filestore to S3 in the context of upgrading Artifactory, or a first time installation, we recommend that you first do a standard installation of Artifactory using the default settings, or a standard upgrade using your current settings.

To move your Artifactory filestore to the cloud, you need to execute the following steps:

- Shut down Artifactory.
- Set your license
- Configure Artifactory to use your S3 object storage provider
- Migrate your files to the cloud manually or automatically
- Start up Artifactory

Setting Your License

To use an S3 object store, your Artifactory installation needs to be activated with the appropriate JFrog license.

To do so, make sure your `$JFROG_HOME/artifactory/var/etc/artifactory/artifactory.lic` file contains your license that supports S3.

Configuring Artifactory to Use S3

Artifactory's filestore is configured through the `binarystore.xml` file. The `binarystore.xml` configuration file is located in the `$JFROG_HOME/artifactory/var/etc/artifactory` folder.

Direct (Eventual-less) versus Eventual Upload Mechanism

Migrating from eventual to direct?

If you are migrating from any eventual mechanism to the direct upload mechanism, make sure your eventual directory is empty or you could experience data loss.

The default S3 chain templates rely on an eventual upload mechanism, whereby an upload from a client is considered successful when the full binary has been uploaded to Artifactory. From Artifactory 7.9.0, the direct upload mechanism serves as an alternative mechanism whereby an upload is not considered successful until it reaches S3. There are advantages to both which we cover below.

Direct Upload Mechanism (Recommended)

The Direct Upload mechanism enables you to upload without the need to maintain persistent storage for the eventual directory. This mechanism may also allow for faster uploads, since there is less contention for disk IO, particularly when Artifactory is hosted on AWS.

1. Client uploads an artifact to Artifactory.
2. Artifactory receives and simultaneously uploads to S3.
3. Artifactory finishes uploading the binary to S3
    a. Artifactory returns 201 success to the client.
    b. A database entry for the artifact is created.

Eventual Upload Mechanism

If you are uploading on a system where the S3 upload speed is slow (for example, when Artifactory is hosted on-prem), you may want to use the Eventual Upload mechanism. The Eventual Upload mechanism also allows you to upload when S3 is down or experiencing network issues.

1. Client uploads an artifact to Artifactory.
2. Artifactory receives the full upload.
    a. Artifactory returns a 201 success message to the client.
    b. A database entry for the artifact is created.
    c. The binary is stored in an eventual directory on the local disk.
3. Artifactory uploads the binary to S3.
4. The binary is deleted from the eventual directory.

Amazon S3 Official SDK Template

Artifactory provides the `s3-storage-v3` template to configure your S3 Cloud Storage using the official Amazon SDK.

High Availability clusters will need to use a shared mount for the S3 provider

Using the Amazon S3 Official SDK template requires a shared mount for cluster. This is because an arbitrary node will handle the object store interaction, while all of the nodes will be able to add files to the shared mount eventual directory.

Authentication Mechanism

The following authentication methods are supported:

- Basic credentials - these are the identity and credential parameters in the table below
- Default Amazon Provider Chain credentials - these are the default Amazon Provider chain searches for credentials, which follow this order:

1. System property based credentials
2. Credentials profiles file.
3. Credentials delivered through the Amazon ECS container service.
4. Instance profile credentials delivered through the Amazon ECS metadata service.

The Amazon S3 Official Amazon SDK template is used for configuring S3 Cloud Storage and supports the following set of parameters.

type	s3-storage-v3
testConnection	Default: true When set to 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.
identity	Your cloud storage provider identity.
credential	Your cloud storage provider authentication credential.
Port	The cloud storage providers port. When a Port is not assigned and the useHttp parameter is set to true, the default port will be set to 80. Otherwise, the default port is set as 443. Note that If you have defined a port, the port will take precedence and will be applied regardless of the value set for the useHttp parameter.
region	The region offered by your cloud storage provider with which you want to work.
bucketName	Your globally unique bucket name.
path	Default: filestore The path relative to the bucket where binary files are stored.
rootFoldersNameLength	Default: 2 The number of initial characters in the object's checksum that should be used to name the folder in storage. This can take any value between 0 - 5.0 means that checksum files will be stored at the root of the object store bucket. For example, if the object's checksum is 8c335149... and `rootFoldersNameLength` is set to 4, the folder under which the object would be stored would be named 8c33.
proxyIdentity	Corresponding parameters if you are accessing the cloud storage provider through a proxy server.
proxyCredential	
proxyPort	
proxyHost	
endPoint	The cloud storage provider's URL. **Amazon Endpoints: Supported JFrog Subscriptions** The AWS S3 `s3.amazonaws.com` endpoint is supported in all the JFrog subscriptions. Additional endpoints are supported in the JFrog Enterprise/ Enterprise+ subscriptions.
useHttp	Default: false Defines the connection schema. When set to true, you can set a non-secure HTTP connection.

kmsClientSideEncryptionKeyId (named kmsServerSideEncryptionKeyId prior to Artifactory version 7.31.10)	Default is N/A. Use KMS Encryption client with given KMS encryption key ID or alias. The name, `kmsServerSideEncryptionKeyId`, is deprecated. But you can continue to use the name without causing any errors.
server-side-encryption-aws-kms	Default is N/A. If set to true, S3 encrypts artifacts on the server based on the default KMS key. You can also set to true with the encryption key ID or alias instead of true.
kmsCryptoMode	Default: EncryptionOnly. Only applies to the s3-storage-v3 template. Use KMS encryption with one of the following crypto policies: • EncryptionOnly • AuthenticatedEncryption • StrictAuthenticatedEncryption For more information, see https://docs.aws.amazon.com/sdk-for-java/v1/developer-guide/examples-crypto-kms.html.
useInstanceCredentials	Default: false. Set to true to use the AWS S3 as your default provider chain according to the Authentication mechanism.
usePresigning	Default: false. When set to true, applies to signed URLs for adding, deleting, getting client methods on s3 objects. Note: Enabling Presigning restricts the maximum size of objects that are uploaded to under 5GB. For more information, see AWS Documentation.
multiPartLimit	Default: 100,000,000 bytes When *usePresigning* is set to false, or for the s3old and s3 templates File size threshold (in bytes) over which file uploads are chunked and multi-threaded.
transferManagerThreads	Default: 10. Only applies to the s3-storage-v3 template. Applies when *usePresigning* is set to false Applies to multipart uploads, configured by the multiPartLimit.
multipartElementSize	Default: A 5 MB chunk. If a tag is not set, the AWS client default of 5MB will be applied. Specify the chunk size (similar to the functionality in Azure BP).
signatureExpirySeconds	Default: 300 Sets the validity period in seconds for signed URLs used internally for uploads and downloads.
maxConnections	Default: 50. Sets the maximum HTTP client connections for the AWS client
connectionTimeout	Default: 10x1000. Sets the connection timeout (in milliseconds) for the AWS client.
socketTimeout	Default: 50 * 1000. Sets the socket timeout (in milliseconds) for the AWS client.
enablePathStyleAccess	Default: false Amazon S3 supports *virtual-hosted-style* and *path-style* access in all regions. The *path-style* syntax requires that using the region-specific endpoint when attempting to access a bucket. For non-AWS users, this property may need to be set to true.

disableChunkedEncoding	Default: false The default behavior is to enable chunked encoding automatically for *PutObjectRequest* and *UploadPartRequest*. Setting this flag will result in disabling chunked encoding for all requests, which may impact performance. Using this option is recommended only if your endpoint does not implement chunked uploading.
useDeprecatedBucketExistenceCheck	Default: false Setting this property will force checking bucket existence based on a HEAD request to the bucket. (Deprecated in AWS)
enableSignedUrlRedirect	Enables direct cloud storage download.
signedUrlExpirySeconds	Default: 30 (optional) Specifies the number of seconds that a signed URL provided to a requesting client for direct download from cloud storage is valid.

The following snippets show the basic template configuration and examples that use the S3 binary provider to support several configurations (CEPH, CleverSafe and more).

S3 Binary Storage Template Configuration

You need to configure the S3 provider with parameters specific to your account (but can leave all other parameters with the recommended values).

S3 Direct Upload Template (Recommended)

The S3 direct (or eventual-less) template allows directly uploading to S3, bypassing the eventual upload mechanism. For more information, see Direct (Eventual-less) versus Eventual Upload Mechanism.

```
<config version="2">
    <chain>
        <provider id="cache-fs" type="cache-fs">
            <provider id="s3-storage-v3" type="s3-storage-v3"/>
        </provider>
    </chain>
    <provider id="s3-storage-v3" type="s3-storage-v3">
       <endpoint>s3.amazonaws.com</endpoint>
       <bucketName>bucketName</bucketName>
       <path>pathPrefix</path>
       <region>s3Region</region>
       <identity>yourIdentity</identity>
       <credential>yourCredentials</credential>
    </provider>
</config>
```

While you don't need to configure anything else in your binarystore.xml, this is what the `s3-storage-v3` template may look like under the hood.

This example sets S3 with your credentials, use pre-signing for object manipulations and sets the pre-signed URL for 10 minutes (600 seconds)

```
<config version="2">
   <chain template="s3-storage-v3-direct"/>
   <provider id="s3-storage-v3" type="s3-storage-v3">
       <endpoint>s3.amazonaws.com</endpoint>
       <bucketName>bucketName</bucketName>
       <path>pathPrefix</path>
       <region>s3Region</region>
       <identity>yourIdentity</identity>
       <credential>yourCredentials</credential>
       <usePresigning>true</usePresigning>
       <signatureExpirySeconds>600</signatureExpirySeconds>
   </provider>
</config>
```

Amazon S3 SDK Cluster Binary Provider

This is the setting used for Amazon S3 Official SDK Template library when configuring filestore sharding for an HA cluster. It is based on the sharding and dynamic provider logic that synchronizes the cluster-file-system.

When using the *cluster-s3-storage-v3* template , data is temporarily stored on the file system of each node using the Eventual-Cluster Binary Provider , and is then passed on to your S3 object storage for persistent storage.

Each node has its own local filestore (just like in the file-system binary provider) and is connected to all other cluster nodes via dynamically allocated Remote Binary Providers using the Sharding-Cluster Binary Provider .

cluster-s3-storage-v3 template configuration

Because you must configure the *s3* provider with parameters specific to your account (but can leave all other parameters with the recommended values), if you choose to use the *cluster-s3-storage-v3* template, your *binarystore.xml* configuration file should look like the example below.

Authentication Methods

Amazon S3 supports two authentication methods.

- Basic credentials - If you use the basic credentials, you will need to enter the identity and credential parameters
- Default Amazon Provider Chain credentials - If you use the default Amazon Provider Chain credentials, you will need to provide the default Amazon Provider chain searches for credentials.

To learn more, and for complete list of parameters used for configuring the S3 Cloud Storage, see Amazon S3 Official SDK Template.

```
<config version="2">
    <chain template="cluster-s3-storage-v3"/>
    <provider id="s3-storage-v3" type="s3-storage-v3">
       <endpoint>s3.amazonaws.com</endpoint>
       <bucketName>bucketName</bucketName>
       <path>pathPrefix</path>
       <region>s3Region</region>
       <identity>yourIdentity</identity>
       <credential>yourCredentials</credential>
       <usePresigning>true</usePresigning>
       <signatureExpirySeconds>600</signatureExpirySeconds>
   </provider>
</config>
```

What's in the template?

While you don't need to configure anything else in your binarystore.xml, this is what the s3-storage-v3 template looks like under the hood.

```
<config version="2">
   <chain>
    <provider id="cache-fs-eventual-s3" type="cache-fs">
       <provider id="sharding-cluster-eventual-s3" type="sharding-cluster">
          <sub-provider id="eventual-cluster-s3" type="eventual-cluster">
             <provider id="retry-s3" type="retry">
                <provider id="s3-storage-v3" type="s3-storage-v3"/>
             </provider>
          </sub-provider>
          <dynamic-provider id="remote-s3" type="remote"/>
     </provider>
   </provider>
  </chain>

    <provider id="cache-fs-eventual-s3" type="cache-fs">
         <maxCacheSize>100000000000</maxCacheSize>
    </provider>

    <provider id="sharding-cluster-eventual-s3" type="sharding-cluster">
       <redundancy>3</redundancy>
       <lenientLimit>2</lenientLimit>
       <property name="zones" value="local,remote"/>
  </provider> <provider id="eventual-cluster-s3" type="eventual-cluster">
       <maxWorkers>10</maxWorkers>
       <dispatcherInterval>1000</dispatcherInterval>
       <checkPeriod>15000</checkPeriod>
       <addStalePeriod>300000</addStalePeriod>
       <zone>local</zone>
  </provider>
  <provider id="remote-s3" type="remote">
       <checkPeriod>15000</checkPeriod>
       <connectionTimeout>5000</connectionTimeout>
       <socketTimeout>15000</socketTimeout>
       <maxConnections>200</maxConnections>
       <connectionRetry>2</connectionRetry>
       <zone>remote</zone>
  </provider>
  <provider id="s3-storage-v3" type="s3-storage-v3">
       <endpoint>s3.amazonaws.com</endpoint>;
       <identity>[ENTER IDENTITY HERE]</identity>
       <credential>[ENTER CREDENTIALS HERE]</credential>
       <path>[ENTER PATH HERE]</path>
       <bucketName>[ENTER BUCKET NAME HERE]</bucketName>
  </provider>
</config>
```

S3 Sharding

You can implement sharding with multiple S3 buckets. The `s3-sharding` template is available with Artifactory to configure sharding with S3. A sub-binary provider, `state-aware-s3`, template is used with the `s3-sharding` template to implelement sharding across multiple S3 buckets.

S3-sharding Template

The `s3-sharding` template uses the same parameters as Amazon S3 official template except for `UrlPresigning`, which is not supported.

```
<chain template="s3-sharding">
    <provider type="sharding" id="sharding">
        <redundancy>2</redundancy>
        <sub-provider type="state-aware-s3" id="s3-shard1"/>
        <sub-provider type="state-aware-s3" id="s3-shard2"/>
    </provider>
</chain>
```

State-Aware-S3 Binary Provider

This binary provider is not independent and will always be used with S3 sharding. The provider is aware if its underlying S3 bucket is functioning or not. It can also recover from errors (the parent provider is responsible for recovery) with the addition of the `checkPeriod` field.

type	state-aware-s3
checkPeriod	Default: 15000 ms The minimum time to wait between trying to re-activate the provider if it had fatal errors at any point.
writeEnabled	Default: true Enables/disables the write operations for the binary provider. If set to false, the state-aware-s3 provider will continue to serve read requests, so Artifactory can continue to read binaries from that provider. In addition, the garbage collection can continue to clean the deleted binaries from the provider. (Only applicable under a sharding provider.)
zone	The name of the sharding zone the provider is part of (only applicable under a sharding provider).

S3 Sharding Example 1

In the following sample configuration, the filestore is implemented with two S3 shards, one region, and one redundancy.

```
<config version="2">

<chain>
      <provider type="sharding" id="sharding">
         <sub-provider type="state-aware-s3" id="s3-shard1" />
         <sub-provider type="state-aware-s3" id="s3-shard2" />
      </provider>
   </chain>
   <provider id="sharding" type="sharding">
      <redundancy>1</redundancy>
   </provider>

<provider id="s3-shard1" type="state-aware-s3">
      <endpoint>http://s3.amazonaws.com</endpoint>
      <bucketName>data1212</bucketName>
      <path>yon1220d</path>
      <region>us-east-1</region>
      <provider.id>aws-s3</provider.id>
      <identity>AK...-accessKeyId</identity>
      <credential>ePE...-secretAccessKey</credential>
      <enableSignedUrlRedirect>true</enableSignedUrlRedirect>
      <signedUrlExpirySeconds>3600</signedUrlExpirySeconds>
      <testConnection>false</testConnection>
      <max.retry.number>2</max.retry.number>
   </provider>
   <provider id="s3-shard2" type="state-aware-s3">
      <endpoint>http://s3.amazonaws.com</endpoint>
      <bucketName>test-tomers-bucket</bucketName>
      <path>yon1220t</path>
      <region>us-east-1</region>
      <provider.id>aws-s3</provider.id>
      <identity>AK...-accessKeyId</identity>
      <credential>ePE...-secretAccessKey</credential>
      <enableSignedUrlRedirect>true</enableSignedUrlRedirect>
      <signedUrlExpirySeconds>3600</signedUrlExpirySeconds>
      <testConnection>false</testConnection>
      <max.retry.number>2</max.retry.number>
   </provider>
</config>
```

S3 Sharding Example 2

In the following sample configuration, the filestore is implemented with two S3 shards, one region, and two redundancy.

```
<config version="2">
   <chain>
      <provider type="sharding" id="sharding">
         <sub-provider type="state-aware-s3" id="s3-shard1" />
         <sub-provider type="state-aware-s3" id="s3-shard2" />
      </provider>
   </chain>
   <provider id="sharding" type="sharding">
      <redundancy>2</redundancy>
   </provider>
   <provider id="s3-shard1" type="state-aware-s3">
      <endpoint>http://s3.amazonaws.com</endpoint>
      <bucketName>bucket1</bucketName>
      <path>path1</path>
      <region>us-east-1</region>
      <provider.id>aws-s3</provider.id>
      <identity>AK...-accessKeyId</identity>
      <credential>ePE...-secretAccessKey</credential>
      <enableSignedUrlRedirect>true</enableSignedUrlRedirect>
      <signedUrlExpirySeconds>3600</signedUrlExpirySeconds>
      <testConnection>false</testConnection>
      <max.retry.number>2</max.retry.number>
   </provider>
   <provider id="s3-shard2" type="state-aware-s3">
      <endpoint>http://s3.amazonaws.com</endpoint>
      <bucketName>bucket2</bucketName>
      <path>path2</path>
      <region>us-east-1</region>
      <provider.id>aws-s3</provider.id>
      <identity>AK...-accessKeyId</identity>
      <credential>ePE...-secretAccessKey</credential>
      <enableSignedUrlRedirect>true</enableSignedUrlRedirect>
      <signedUrlExpirySeconds>3600</signedUrlExpirySeconds>
      <testConnection>false</testConnection>
      <max.retry.number>2</max.retry.number>
   </provider>
</config>
```

S3 Sharding Example 3

In the following sample configuration, the filestore is implemented with five S3 shards, two region, and two redundancy.

```
<?xml version="1.0" encoding="UTF-8"?>
<config version="5">

<chain>
      <provider type="sharding" id="sharding">
         <sub-provider type="state-aware-s3" id="s3-shard1" />
         <sub-provider type="state-aware-s3" id="s3-shard2" />
         <sub-provider type="state-aware-s3" id="s3-shard3" />
         <sub-provider type="state-aware-s3" id="s3-shard4" />
         <sub-provider type="state-aware-s3" id="s3-shard5" />
      </provider>
   </chain>
   <provider id="sharding" type="sharding">
      <redundancy>3</redundancy>
   </provider>
    <provider id="s3-shard1" type="state-aware-s3">
      <endpoint>http://s3.amazonaws.com</endpoint>
      <bucketName>data120</bucketName>
      <path>yond</path>
      <region>us-east-1</region>
      <provider.id>aws-s3</provider.id>
      <identity>AK...-accessKeyId</identity>
      <credential>ePE...-secretAccessKey</credential>
      <enableSignedUrlRedirect>true</enableSignedUrlRedirect>
```

```
      <signedUrlExpirySeconds>3600</signedUrlExpirySeconds>
      <testConnection>false</testConnection>
      <max.retry.number>2</max.retry.number>
   </provider>
   <provider id="s3-shard2" type="state-aware-s3">
      <endpoint>http://s3.amazonaws.com</endpoint>
      <bucketName>data125</bucketName>
      <path>yont</path>
      <region>us-east-1</region>
      <provider.id>aws-s3</provider.id>
      <identity>AK...-accessKeyId</identity>
      <credential>ePE...-secretAccessKey</credential>
      <writeEnabled>false</writeEnabled>
      <enableSignedUrlRedirect>true</enableSignedUrlRedirect>
      <signedUrlExpirySeconds>3600</signedUrlExpirySeconds>
      <testConnection>false</testConnection>
      <max.retry.number>2</max.retry.number>
   </provider>
   <provider id="s3-shard3" type="state-aware-s3">
      <endpoint>http://s3.amazonaws.com</endpoint>
      <bucketName>data121</bucketName>
      <path>yonb</path>
      <region>us-west-1</region>
      <provider.id>aws-s3</provider.id>
      <identity>AK...-accessKeyId</identity>
      <credential>ePE...-secretAccessKey</credential>
      <enableSignedUrlRedirect>true</enableSignedUrlRedirect>
      <signedUrlExpirySeconds>3600</signedUrlExpirySeconds>
      <testConnection>false</testConnection>
      <max.retry.number>2</max.retry.number>
   </provider>
   <provider id="s3-shard4" type="state-aware-s3">
      <endpoint>http://s3.amazonaws.com</endpoint>
      <bucketName>data122</bucketName>
      <path>yonb2</path>
      <region>us-east-1</region>
      <provider.id>aws-s3</provider.id>
      <identity>AK...-accessKeyId</identity>
      <credential>ePE...-secretAccessKey</credential>
      <enableSignedUrlRedirect>true</enableSignedUrlRedirect>
      <signedUrlExpirySeconds>3600</signedUrlExpirySeconds>
      <testConnection>false</testConnection>
      <max.retry.number>2</max.retry.number>
   </provider>
   <provider id="s3-shard5" type="state-aware-s3">
      <endpoint>http://s3.amazonaws.com</endpoint>
      <bucketName>data123</bucketName>
      <path>yonb3</path>
      <region>us-west-1</region>
      <provider.id>aws-s3</provider.id>
      <identity>AK...-accessKeyId</identity>
      <credential>ePE...-secretAccessKey</credential>
      <enableSignedUrlRedirect>true</enableSignedUrlRedirect>
      <signedUrlExpirySeconds>3600</signedUrlExpirySeconds>
      <testConnection>false</testConnection>
      <max.retry.number>2</max.retry.number>
   </provider>
</config>
```

Migrating Your Filestore to S3

There are two ways to migrate your filestore over to your S3 provider.

- Automatically (recommended)
- Manually

Automatic Filestore Migration (Recommended)

To make sure your filestore migration completes successfully without corrupting files, we recommend configuring Artifactory to do this migration for you automatically:

To do so, you need to create the following links in `$JFROG_HOME/artifactory/var/data/artifactory/eventual/`(create it if the *`eventual`* folder does not exist - it is created automatically when the eventual binary provider is applied via an Artifactory restart with an updated binarystore.xml):

- A link with the name **_add** that points to the `$JFROG_HOME/artifactory/var/data/artifactory/`**`filestore`** directory
- A link with the name **_pre** that points to the `$JFROG_HOME/artifactory/var/data/artifactory/filestore/`**`_pre`** directory

With this setting, as soon as Artifactory starts up, it will automatically move your complete filestore over to your S3 provider. Artifactory validates each file uploaded to S3 and removes the file from the NFS if the transfer is successful.

Your current filestore will be deleted

The process of moving your filestore to your S3 provider will delete your current filestore. We strongly recommend you do a complete system backup before doing this migration.

Once the migration is complete, you may delete the `_pre` link and the `$JFROG_HOME/artifactory/var/data/artifactory/`**`_pre`** directory

Manual Filestore Migration

To migrate your filestore manually, you need to execute the following steps:

- Stop Artifactory
- Copy the `$JFROG_HOME/artifactory/var/data/artifactory/filestore` directory to your S3 object storage to the bucket name and path specified when you configured Artifactory to use S3.
- Configure Artifactory to use S3 in the `binarystore.xml` file in the `$JFROG_HOME/artifactory/var/etc/artifactory` folder.
- Start Artifactory