Hortonworks Data Platform

Cloud Data Access

(June 1, 2017)

docs.hortonworks.com

Hortonworks Data Platform: Cloud Data Access

The Hortonworks Data Platform, powered by Apache Hadoop, is a massively scalable and 100% open source platform for storing, processing and analyzing large volumes of data. It is designed to deal with data from many sources and formats in a very quick, easy and cost-effective manner. The Hortonworks Data Platform consists of the essential set of Apache Hadoop projects including MapReduce, Hadoop Distributed File System (HDFS), HCatalog, Pig, Hive, HBase, ZooKeeper and Ambari. Hortonworks is the major contributor of code and patches to many of these projects. These projects have been integrated and tested as part of the Hortonworks Data Platform release process and installation and configuration tools have also been included.

Unlike other providers of platforms built using Apache Hadoop, Hortonworks contributes 100% of our code back to the Apache Software Foundation. The Hortonworks Data Platform is Apache-licensed and completely open source. We sell only expert technical support, training and partner-enablement services. All of our technology is, and will remain free and open source. Please visit the Hortonworks Data Platform page for more information on Hortonworks technology. For more information on Hortonworks services, please visit either the Support or Training page. Feel free to Contact Us directly to discuss your specific needs.

Table of Contents

List of Figures

List of Tables

1. About This Guide

The goal of this guide is to provide information and steps required for configuring, using, securing, tuning performance, and troubleshooting access to the cloud storage services using HDP cloud storage connectors available for Amazon Web Services (Amazon S3) and Microsoft Azure (ADLS, WASB).

The primary audience of this guide are the administrators and users of HDP deployed on cloud Infrastructure-as-a-Service (IaaS) such as Amazon Web Services (AWS) or Microsoft Azure. You may also use this guide if your HDP is deployed in your own data center and you plan to access cloud storage via the connectors; however, your experience and performance may vary based on the network bandwidth between your data center and the cloud storage service.

To learn about the architecture of the cloud connectors, refer to Introducing the Cloud Storage Connectors.

In order to start working with data stored in a cloud storage service, you must configure authentication with the service. In addition, you can optionally configure other features where available. To get started with your chosen cloud storage service, refer to:

- Getting Started with Amazon S3 [5]
- Getting Started with ADLS [45]
- Getting Started with WASB [51]

Once you have configured authentication with the chosen cloud storage service, you can start working with the data. To get started, refer to:

- Accessing Cloud Data in Hive [58]
- Accessing Cloud Data in Spark [62]
- Copying Cloud Data with Hadoop [65]

2. Introducing the Cloud Storage Connectors

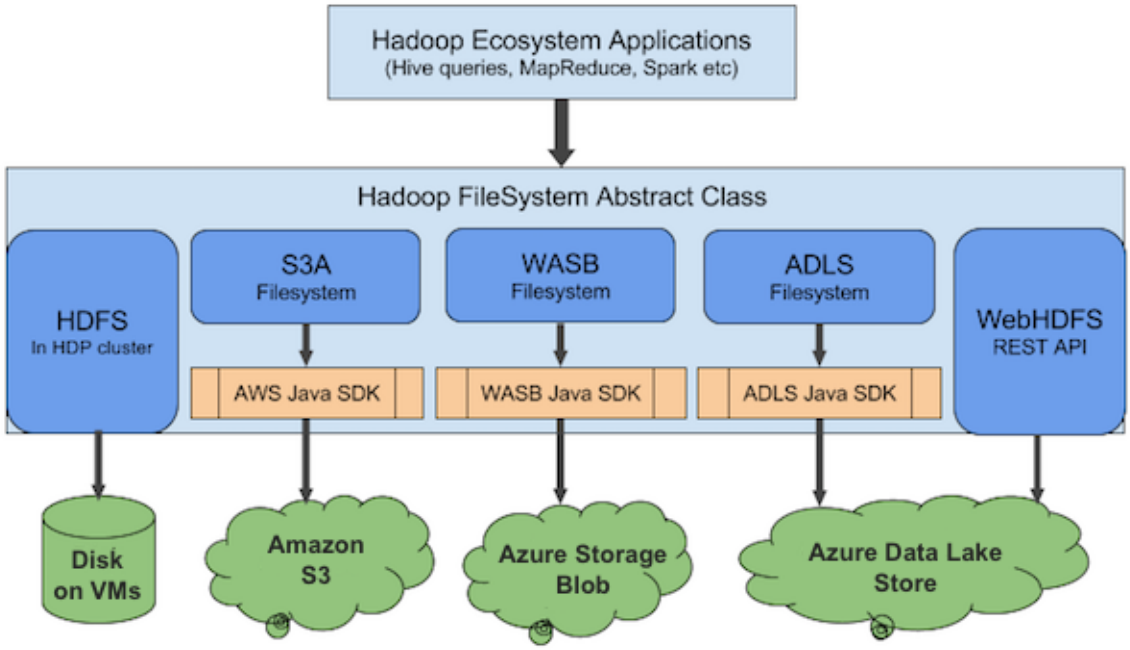
When deploying HDP clusters on AWS or Azure cloud IaaS, you can take advantage of the native integration with the cloud object storage services available on each of the cloud platforms: **Amazon S3 on AWS**, and **ADLS and WASB on Azure**. This integration is via cloud storage connectors included with HDP. Their primary function is to help you connect to, access, and work with data the cloud storage services. These connectors are not a replacement for HDFS and cannot be used as a replacement for HDFS `defaultFS`.

The cloud connectors allow you to seamlessly access and work with data stored in Amazon S3, Azure ADLS and Azure WASB storage services, including, but not limited to, the following use cases:

- Collect data for analysis and then load it into Hadoop ecosystem applications such as Hive or Spark directly from cloud storage services.
- Persist data to cloud storage services for use outside of HDP clusters.
- Copy data stored in cloud storage services to HDFS for analysis and then copy back to the cloud when done.
- Share data between multiple HDP clusters – and between various external non-HDP systems – by pointing at the same data sets in the cloud object stores.

The S3A, ADLS and WASB connectors are implemented as individual Hadoop modules. The libraries and their dependencies are automatically placed on the classpath.

Figure 2.1. HDP Cloud Storage Connector Architecture



Amazon S3 is an object store. The S3A connector implements the Hadoop filesystem interface using WWS Java SDK to access the web service, and provides Hadoop applications with a filesystem view of the buckets. Applications can seamlessly manipulate data stored in Amazon S3 buckets with an URL starting with the *s3a://* prefix.

Azure WASB is an object store with a flat name architecture (flat name space). The WASB connector implements the Hadoop filesystem interface using WASB Java SDK to access the web service, and provides Hadoop applications with a filesystem view of the blobs. Applications can seamlessly manipulate data stored in WASB with an URL starting with the *wasb://* prefix.

Azure ADLS is a WebHDFS-compatible hierarchical file system. Applications can access the data in ADLS directly using WebHDFS REST API. Meanwhile, the ADLS connector implements the Hadoop filesystem interface using ADLS Java SDK to access the web service. Applications can manipulate data stored in ADLS with the URL starting with the *adl://* prefix.

Table 2.1. Cloud Storage Connectors

Cloud Storage Service	Connector Description	URL Prefix
Amazon Simple Storage Service (S3)	The S3A connector enables reading and writing files stored in the Amazon S3 object store.	s3a://
Azure Data Lake Store (ADLS)	The ADLS connector enables reading and writing files stored in the ADLS file system.	adl://
Windows Azure Storage Blob (WASB)	The WASB connector enables reading and writing both block blobs and page blobs from and to WASB object store.	wasb://

The cluster's default filesystem HDFS is defined in the configuration property `fs.defaultFS` in `core-site.xml`. As a result, when running FS shell commands or DistCp against HDFS, you can but do not need to specify the `hdfs://` URL prefix:

```
hadoop distcp hdfs://source-folder s3a://destination-bucket
```

```
hadoop distcp /source-folder s3a://destination-bucket
```

When working with the cloud using cloud URIs **do not change** the value of `fs.defaultFS` to use a cloud storage connector as the filesystem for HDFS. This is **not recommended or supported**. Instead, when working with data stored in S3, ADLS, or WASB, use a fully qualified URL for that connector.

To get started with your chosen cloud storage service, refer to:

- Getting Started with Amazon S3 [5]
- Getting Started with ADLS [45]
- Getting Started with WASB [51]

3. Getting Started with Amazon S3

The following table provides an overview of tasks related to configuring and using HDP with S3. Click on the linked topics to get more information about specific tasks.

Note

If you are looking for data sets to play around, you can use Landsat 8 data sets made available by AWS in a public Amazon S3 bucket called "landsat-pds". For more information, refer to Landsat on AWS.

Task	Description
Meet the prerequisites	To use S3 storage, you must have: 1. An AWS account. 2. One or more S3 buckets on your AWS account. For instructions on how to create a bucket on S3, refer to AWS documentation.
Configure authentication	In order for Hadoop applications to access data stored in your private S3 buckets, you must configure authentication with your Amazon S3 account.
Configure optional features: • Configuring Per-Bucket Settings [14] • A List of S3A Configuration Properties [18]	You can optionally configure additional features such as bucket-specific settings.
Work with S3 data: • Referencing S3 in the URLs [14] • Access data with Hive or Spark • Copy data with DistCp	Once you've configured authentication with your S3 bucket(s), you can access S3 data from Hive (via external tables) and Spark, and perform related tasks such as copying data between HDFS and S3 when needed.
Encrypting Data on S3 [22]	You can optionally work with S3 data that is protected with server-side encryption: SSE-S3, SSE-KMS, or SSE-C.
Improving Performance for S3 [27]	You can optionally configure and fine-tune performance-related features to optimize HDP performance for specific tasks including accessing S3 data from Hive, Spark, and copying data with DistCp.
Troubleshoot	Refer to this section if you experience issues while configuring or using S3 with HDP.

3.1. About Amazon S3

Amazon S3 object store is the standard mechanism to store, retrieve, and share large quantities of data in AWS.

The features of Amazon S3 include:

- Object store model for storing, listing, and retrieving data.
- Support for objects up to 5 terabytes, with many petabytes of data allowed in a single bucket.
- Data is stored in Amazon S3 in buckets which are stored in different AWS regions.
- Buckets can restricted to different users or IAM roles.

- Data stored in an Amazon S3 bucket is billed based on the size of data and based on how long it is stored. In addition, you are billed when you transfer data between regions:
  - Data transfers between an Amazon S3 bucket and a cluster running in the same region are free of download charges (except in the special case of buckets in which data is served on a user-pays basis).
  - Data downloaded from an Amazon S3 bucket located outside the region in which the bucket is hosted is billed per megabyte.
- Data stored in Amazon S3 can be backed up with Amazon Glacier.

The Hadoop client to S3, called "S3A", makes the contents of a bucket appear like a filesystem, with directories, files in the directories, and operations on directories and files. As a result applications which can work with data stored in HDFS can also work with data stored in S3. However, since S3 is an object store, it has certain limitations that you should be aware of.

3.1.1. Limitations of Amazon S3

Even though Hadoop's S3A client can make an S3 bucket appear to be a Hadoop-compatible filesystem, it is still an object store, and has some limitations when acting as a Hadoop-compatible filesystem. The key things to be aware of are:

- Operations on directories are potentially slow and nonatomic.
- Not all file operations are supported. In particular, some file operations needed by Apache HBase are not available — so HBase cannot be run on top of Amazon S3.
- Data is not visible in the object store until the entire output stream has been written.
- Amazon S3 is *eventually consistent*. Objects are replicated across servers for availability, but changes to a replica take time to propagate to the other replicas; the object store is *inconsistent* during this process. The inconsistency issues surface when listing, reading, updating, or deleting files.
- Neither the per-file and per-directory permissions supported by HDFS nor its more sophisticated ACL mechanism are supported.
- Bandwidth between your workload clusters and Amazon S3 is limited and can vary significantly depending on network and VM load.

For these reasons, while Amazon S3 can be used as the source and store for persistent data, it cannot be used as a direct replacement for a cluster-wide filesystem such as HDFS, or be used as `defaultFS`.

3.2. Configuring Authentication with S3

For Apache Hadoop applications to be able to interact with Amazon S3, they must know the AWS access key and the secret key. This can be achieved in three different ways: through configuration properties, environment variables, or instance metadata. While the first two options can be used when accessing S3 from a cluster running in your own data

center. IAM roles, which use instance metadata should be used to control access to AWS resources if your cluster is running on EC2.

Table 3.1. Authentication Options for Different Deployment Scenarios

Deployment Scenario	Authentication Options
Cluster runs on EC2	Use IAM roles to control access to your AWS resources. If you configure role-based access, instance metadata will automatically be used to authenticate.
Cluster runs in your own data center	Use configuration properties to authenticate. You can set the configuration properties globally or per-bucket.

Temporary security credentials, also known as "session credentials", can be issued. These consist of a secret key with a limited lifespan, along with a session token, another secret which must be known and used alongside the access key. The secret key is never passed to AWS services directly. Instead it is used to sign the URL and headers of the HTTP request.

By default, the S3A filesystem client follows the following authentication chain:

1. If login details were provided in the filesystem URI, a warning is printed and then the username and password are extracted for the AWS key and secret respectively. However, authenticating via embedding credentials in the URL is not recommended. Instead, you may authenticate using per-bucket authentication credentials.

2. The `fs.s3a.access.key` and `fs.s3a.secret.key` are looked for in the Hadoop configuration properties.

3. The AWS environment variables are then looked for.

4. An attempt is made to query the Amazon EC2 Instance Metadata Service to retrieve credentials published to EC2 VMs.

3.2.1. Using Instance Metadata to Authenticate

If your cluster is running on EC2, the standard way to manage access is via Amazon Identity and Access Management (IAM), which allows you to create users, groups, and roles to control access to services such as Amazon S3 via attached policies. A role does not have any credentials such as password or access keys associated with it. Instead, if a user is assigned to a role, access keys are generated dynamically and provided to the user when needed. For more information, refer to IAM Roles for Amazon EC2 in Amazon documentation.

When launching your cluster on EC2, specify an IAM role that you want to use; if you are planning to use S3 with your cluster, make sure that the role associated with the cluster includes a policy that grants access to S3. For more information, refer to Using an IAM Role to Grant Permissions to Applications Running on Amazon EC2 Instances in Amazon documentation. No additional configuration is required.

Note

You can use IAM Roles to control access to keys stored in Amazon's KMS Key Management service. For more information, refer to Overview of Managing Access to Your AWS KMS Resources in Amazon documentation.

3.2.2. Using Configuration Properties to Authenticate

To configure authentication with S3, explicitly declare the credentials in a configuration file such as `core-site.xml`:

```
<property>
  <name>fs.s3a.access.key</name>
  <value>ACCESS-KEY</value>
</property>

<property>
  <name>fs.s3a.secret.key</name>
  <value>SECRET-KEY</value>
</property>
```

If using AWS Session credentials for authentication, the secret key must be that of the session, and the `fs.s3a.session.token` option set to your session token.

```
<property>
  <name>fs.s3a.session.token</name>
  <value>SESSION-TOKEN</value>
</property>
```

This configuration can be added for a specific bucket. For more information, refer to Using Per-Bucket Credentials to Authenticate.

Next Steps

To protect these credentials, we recommend that you use the credential provider framework to securely store and access your credentials.

To validate that you can successfully authenticate with S3, try referencing S3 in a URL.

3.2.2.1. Using Per-Bucket Credentials to Authenticate

S3A supports per-bucket configuration, which can be used to declare different authentication credentials and authentication mechanisms for different buckets.

For example, a bucket `s3a://nightly/` used for nightly data can be configured with a session key:

```
<property>
  <name>fs.s3a.bucket.nightly.access.key</name>
  <value>AKAACCESSKEY-2</value>
</property>

<property>
  <name>fs.s3a.bucket.nightly.secret.key</name>
  <value>SESSIONSECRETKEY</value>
</property>
```

Similarly, you can set a session token for a specific bucket:

```
<property>
  <name>fs.s3a.bucket.nightly.session.token</name>
  <value>SESSION-TOKEN</value>
</property>
```

This technique is useful for working with external sources of data, or when copying data between buckets belonging to different accounts.

Related Links

Customizing Per-Bucket Secrets Held in Credential Files [13]

3.2.3. Using Environment Variables to Authenticate

AWS CLI supports authentication through environment variables. These same environment variables will be used by Hadoop if no configuration properties are set.

The environment variables are:

Environment Variable	Description
`AWS_ACCESS_KEY_ID`	Access key
`AWS_SECRET_ACCESS_KEY`	Secret key
`AWS_SESSION_TOKEN`	Session token (only if using session authentication)

3.2.4. Embedding Credentials in the URL to Authenticate

Note

Embedding credentials in the URL is dangerous and deprecated. Due to the security risk it represents, future versions of Hadoop may remove this feature entirely. Use per-bucket configuration options instead.

Hadoop supports embedding credentials within the S3 URL:

```
s3a://key:secret@bucket-name/
```

In general, we strongly discourage using this mechanism, as it invariably results in the secret credentials being logged in many places in the cluster. However, embedding credentials in the URL is sometimes useful when troubleshooting authentication problems; consult the troubleshooting documentation for details.

Before S3A supported per-bucket credentials, this was the sole mechanism for supporting different credentials for different buckets. Now that buckets can be individually configured, this mechanism should no longer be needed. You should use per-bucket configuration options instead.

3.2.5. Defining Authentication Providers

The S3A connector can be configured to obtain client authentication providers from classes which integrate with the AWS SDK by implementing the `com.amazonaws.auth.AWSCredentialsProvider` interface. This is done by listing the implementation classes in the configuration option `fs.s3a.aws.credentials.provider`.

Note

AWS credential providers are distinct from Hadoop credential providers. *Hadoop credential providers* allow passwords and other secrets to be stored

and transferred more securely than in XML configuration files. In contrast, *AWS credential providers* are classes which can be used by the Amazon AWS SDK to obtain an AWS login from a different source in the system, including environment variables, JVM properties, and configuration files.

There are a number of AWS credential provider classes specified in the `hadoop-aws` JAR:

Classname	Description
`org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider`	Standard credential support through configuration properties. It does not support in-URL authentication.
`org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider`	Session authentication
`org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider`	Anonymous login

Furthermore, there are many AWS credential provider classes specified in the Amazon JARs. In particular, there are two which are commonly used:

Classname	Description
`com.amazonaws.auth.EnvironmentVariableCredentialsProvider`	AWS Environment Variables
`com.amazonaws.auth.InstanceProfileCredentialsProvider`	EC2 Metadata Credentials

The order of listing credential providers in the configuration option `fs.s3a.aws.credentials.provider`defines the order of evaluation of credential providers.

The standard authentication mechanism for Hadoop S3A authentication is the following list of providers:

```
<property>
  <name>fs.s3a.aws.credentials.provider</name>
  <value>
  org.apache.hadoop.fs.s3a.SimpleAWSCredentialsProvider,
  com.amazonaws.auth.EnvironmentVariableCredentialsProvider,
  com.amazonaws.auth.InstanceProfileCredentialsProvider</value>
</property>
```

Note

Retrieving credentials with the `InstanceProfileCredentialsProvider` is a slower operation than looking up configuration operations or environment variables. It is best to list it after all other authentication providers — excluding the `AnonymousAWSCredentialsProvider`, which must come last.

3.2.5.1. Using Temporary Session Credentials

Temporary Security Credentials can be obtained from the AWS Security Token Service. These credentials consist of an access key, a secret key, and a session token.

To authenticate with these credentials:

1. Declare `org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider` as the provider.

2. Set the session key in the property `fs.s3a.session.token`, and set the access and secret key properties to those of this temporary session.

```
<property>
  <name>fs.s3a.aws.credentials.provider</name>
  <value>org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider</value>
</property>

<property>
  <name>fs.s3a.access.key</name>
  <value>SESSION-ACCESS-KEY</value>
</property>

<property>
  <name>fs.s3a.secret.key</name>
  <value>SESSION-SECRET-KEY</value>
</property>

<property>
  <name>fs.s3a.session.token</name>
  <value>SECRET-SESSION-TOKEN</value>
</property>
```

The lifetime of session credentials is determined when the credentials are issued; once they expire the application will no longer be able to authenticate to AWS.

3.2.5.2. Using Anonymous Login

You can configure anonymous access to a publicly accessible Amazon S3 bucket without using any credentials. This can be useful for accessing public data sets.

Note

Allowing anonymous access to an Amazon S3 bucket compromises security and therefore is unsuitable for most use cases.

To use anonymous login, specify `org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider`:

```
<property>
  <name>fs.s3a.aws.credentials.provider</name>
  <value>org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider</value>
</property>
```

Once this is done, there is no need to supply any credentials in the Hadoop configuration or via environment variables.

This option can be used to verify that an object store does not permit unauthenticated access; that is, if an attempt to list a bucket is made using the anonymous credentials, it should fail — unless explicitly opened up for broader access.

```
hadoop fs -ls \
  -D fs.s3a.aws.credentials.provider=org.apache.hadoop.fs.s3a.
AnonymousAWSCredentialsProvider \
  s3a://landsat-pds/
```

S3A may be configured to always access specific buckets anonymously. For example, the following configuration defines anonymous access to the public `landsat-pds` bucket accessed via `s3a://landsat-pds/` URI:

```
<property>
  <name>fs.s3a.bucket.landsat-pds.aws.credentials.provider</name>
  <value>org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider</value>
</property>
```

Note

If a list of credential providers is given in `fs.s3a.aws.credentials.provider`, then the anonymous credential provider *must* come last. If not, credential providers listed after it will be ignored.

3.2.5.3. Protecting S3 Credentials with Credential Providers

The Hadoop credential provider framework allows secure credential providers to keep secrets outside Hadoop configuration files, storing them in encrypted files in local or Hadoop filesystems, and including them in requests.

The S3A configuration options with sensitive data (`fs.s3a.secret.key`, `fs.s3a.access.key`, and `fs.s3a.session.token`) can have their data saved to a binary file, with the values being read in when the S3A filesystem URL is used for data access. The reference to this credential provider is all that is passed as a direct configuration option.

To protect your credentials with credential providers:

1. Creating a Credential File [12]

2. Configuring the Hadoop Security Credential Provider Path Property [13]

In addition, if you are using per-bucket credentials, refer to Customizing Per-Bucket Secrets Held in Credential Files.

3.2.5.3.1. Creating a Credential File

You can create a credential file on any Hadoop filesystem. When you create one on HDFS or a UNIX filesystem, the permissions are automatically set to keep the file private to the reader — though as directory permissions are not touched, you should verify that the directory containing the file is readable only by the current user. For example:

```
hadoop credential create fs.s3a.access.key -value 123 \
    -provider jceks://hdfs@nn1.example.com:9001/user/backup/s3.jceks

hadoop credential create fs.s3a.secret.key -value 456 \
    -provider jceks://hdfs@nn1.example.com:9001/user/backup/s3.jceks
```

After creating the credential file, you can list it to see what entries are kept inside it. For example:

```
hadoop credential list -provider jceks://hdfs@nn1.example.com:9001/user/
backup/s3.jceks

Listing aliases for CredentialProvider: jceks://hdfs@nn1.example.com:9001/
user/backup/s3.jceks
fs.s3a.secret.key
fs.s3a.access.key
```

After performing these steps. credentials are ready for use.

3.2.5.3.2. Configuring the Hadoop Security Credential Provider Path Property

The URL to the provider must be set in the configuration property `hadoop.security.credential.provider.path`, either in the `core-site.xml` configuration file or on the command line:

Example: Setting via Configuration File

```
<property>
  <name>hadoop.security.credential.provider.path</name>
  <value>jceks://hdfs@nn1.example.com:9001/user/backup/s3.jceks</value>
</property>
```

Because this property only supplies the path to the secrets file, the configuration option itself is no longer a sensitive item.

Example: Setting via Command Line

```
hadoop distcp \
  -D hadoop.security.credential.provider.path=jceks://hdfs@nn1.example.
com:9001/user/backup/s3.jceks \
  hdfs://nn1.example.com:9001/user/backup/007020615 s3a://glacier1/

hadoop fs \
  -D hadoop.security.credential.provider.path=jceks://hdfs@nn1.example.
com:9001/user/backup/s3.jceks \
  -ls s3a://glacier1/
```

Because the provider path is not itself a sensitive secret, there is no risk from placing its declaration on the command line.

Once the provider is set in the Hadoop configuration, hadoop commands work exactly as if the secrets were in an XML file. For example:

```
hadoop distcp hdfs://nn1.example.com:9001/user/backup/007020615 s3a://
glacier1/
hadoop fs -ls s3a://glacier1/
```

3.2.5.3.3. Customizing Per-Bucket Secrets Held in Credential Files

Although most properties which are set per-bucket are automatically propagated from their `fs.s3a.bucket.` custom entry to that of the base `fs.s3a.` option, supporting secrets kept in Hadoop credential files is slightly more complex: property values are kept in these files, and they cannot be dynamically patched.

Instead, callers need to create different configuration files for each bucket, setting the base secrets, then declare the path to the appropriate credential file in a bucket-specific version of the property `fs.s3a.security.credential.provider.path`.

Example

1. Set *base* properties for `fs.s3a.secret.key` and `fs.s3a.access.key` in `core-site.xml` or in your job submission.

2. Set similar properties *per-bucket* for a bucket called "frankfurt-1". These will override the base properties when talking to the bucket "frankfurt-1".

3. When setting properties in a JCEKS file, you must use the *base* property names — even if you only intend to use them *for a specific bucket*.

   For example, in the JCEKS file called `hdfs://users/steve/frankfurt.jceks`, set the *base* parameters fs.s3a.secret.key, fs.s3a.access.key to your "frankfurt-1" values from step 2.

4. Next, set the path to the JCEKS file as a *per-bucket* option.

   For example, fs.s3a.bucket.frankfurt-1.security.credential.provider.path should be set to `hdfs://users/steve/frankfurt.jceks`.

5. When the credentials for "frankfurt-1" are set up, the property `fs.s3a.bucket.frankfurt-1.security.credential.provider.path` will be read, and the secrets from that file used to set the options to access the bucket.

Related Links

Using Per-Bucket Credentials to Authenticate [8]

Credential Provider API

3.3. Referencing S3 in the URLs

Regardless of which specific Hadoop ecosystem application you are using, you can access data stored in Amazon S3 using the URL starting with the *s3a://* prefix followed by bucket name and path to file or directory.

The URL structure is:

```
s3a://<bucket>/<dir>/<file>
```

For example, to access a file called "mytestfile" in a directory called "mytestdir", which is stored in a bucket called "mytestbucket", the URL is:

```
s3a://mytestbucket/mytestdir/mytestfile
```

The following FileSystem shell commands demonstrate access to a bucket named `mytestbucket`:

```
hadoop fs -ls s3a://mytestbucket/

hadoop fs -mkdir s3a://mytestbucket/testDir

hadoop fs -put testFile s3a://mytestbucket/testFile

hadoop fs -cat s3a://mytestbucket/testFile
test file content
```

3.4. Configuring Per-Bucket Settings

You can specify bucket-specific configuration values which override the common configuration values.

This allows for:

- Different authentication mechanisms and credentials on different buckets
- Different encryption policies on different buckets
- Different S3 endpoints to use for different buckets. This is essential when working with S3 regions which only support the "V4 authentication API", in case of which callers must always declare the explicit region

All `fs.s3a` options other than a small set of unmodifiable values (currently `fs.s3a.impl`) can be set on a per-bucket basis.

To set a bucket-specific option:

1. Add a new configuration, replacing the `fs.s3a.` prefix on an option with `fs.s3a.bucket.BUCKETNAME.`, where BUCKETNAME is the name of the bucket.

   For example, if you are configuring access key for a bucket called "nightly", instead of using *`fs.s3a.access.key`* property name, use *`fs.s3a.bucket.nightly.access.key`*.

2. When connecting to a bucket, all options explicitly set for that bucket will override the base `fs.s3a.` values, but they will not be picked up by other buckets.

Example

You may have a base configuration to use the IAM role information available when deployed in Amazon EC2:

```
<property>
  <name>fs.s3a.aws.credentials.provider</name>
  <value>org.apache.hadoop.fs.s3a.SharedInstanceProfileCredentialsProvider</
value>
</property>
```

This will be the default authentication mechanism for S3A buckets.

A bucket `s3a://nightly/` used for nightly data uses a session key, so its bucket-specific configuration is:

```
<property>
  <name>fs.s3a.bucket.nightly.access.key</name>
  <value>AKAACCES-SKEY-2</value>
</property>

<property>
  <name>fs.s3a.bucket.nightly.secret.key</name>
  <value>SESSION-SECRET-KEY</value>
</property>

<property>
  <name>fs.s3a.bucket.nightly.session.token</name>
  <value>SHORT-LIVED-SESSION-TOKEN</value>
</property>

<property>
  <name>fs.s3a.bucket.nightly.aws.credentials.provider</name>
  <value>org.apache.hadoop.fs.s3a.TemporaryAWSCredentialsProvider</value>
```

```
</property>
```

Finally, the public `s3a://landsat-pds/` bucket could be accessed anonymously, so its bucket-specific configuration is:

```
<property>
   <name>fs.s3a.bucket.landsat-pds.aws.credentials.provider</name>
   <value>org.apache.hadoop.fs.s3a.AnonymousAWSCredentialsProvider</value>
</property>
```

For all other buckets, the base configuration is used.

Related Links

Configuring Per-Bucket Settings to Access Data Around the World [16]

Using Per-Bucket Credentials to Authenticate [8]

Customizing Per-Bucket Secrets Held in Credential Files [13]

3.4.1. Configuring Per-Bucket Settings to Access Data Around the World

S3 buckets are hosted in different AWS regions, the default being "US-East". The S3A client talks to this region by default, issuing HTTP requests to the server `s3.amazonaws.com`. This **central endpoint** can be used for accessing any bucket in any region which supports using the V2 Authentication API, albeit possibly at a reduced performance.

Each region has its own S3 endpoint, documented by Amazon. The S3A client supports these endpoints. While it is generally simpler to use the default endpoint, direct connections to specific regions (i.e. connections via region's own endpoint) may deliver performance and availability improvements, and are mandatory when working with the most recently deployed regions, such as Frankfurt and Seoul.

When deciding which endpoint to use, consider the following:

- Applications running in EC2 infrastructure do not pay for data transfers to or from local S3 buckets. In contrast, they will be billed for access to remote buckets. Therefore, wherever possible, always use local buckets and local copies of data.
- When the V1 request signing protocol is used, the default S3 endpoint can support data transfer with any bucket.
- When the V4 request signing protocol is used, AWS requires the explicit region endpoint to be used — hence S3A must be configured to use the specific endpoint. This is done in the configuration option `fs.s3a.endpoint`.
- All endpoints other than the default endpoint only support interaction with buckets local to that S3 instance.

If the wrong endpoint is used, the request may fail. This may be reported as a 301 redirect error, or as a 400 Bad Request. Take these failures as cues to check the endpoint setting of a bucket.

Here is a list of properties defining all Amazon S3 regions, as of March 2017.

These parameters can be used to specify endpoints for individual buckets. You can add these properties to your `core-site.xml`:

```
<!-- This is the default endpoint, which can be used to interact with any v2
 region. -->
<property>
  <name>central.endpoint</name>
  <value>s3.amazonaws.com</value>
</property>

<property>
  <name>canada.endpoint</name>
  <value>s3.ca-central-1.amazonaws.com</value>
</property>

<property>
  <name>frankfurt.endpoint</name>
  <value>s3.eu-central-1.amazonaws.com</value>
</property>

<property>
  <name>ireland.endpoint</name>
  <value>s3-eu-west-1.amazonaws.com</value>
</property>

<property>
  <name>london.endpoint</name>
  <value>s3.eu-west-2.amazonaws.com</value>
</property>

<property>
  <name>mumbai.endpoint</name>
  <value>s3.ap-south-1.amazonaws.com</value>
</property>

<property>
  <name>ohio.endpoint</name>
  <value>s3.us-east-2.amazonaws.com</value>
</property>

<property>
  <name>oregon.endpoint</name>
  <value>s3-us-west-2.amazonaws.com</value>
</property>

<property>
  <name>sao-paolo.endpoint</name>
  <value>s3-sa-east-1.amazonaws.com</value>
</property>

<property>
  <name>seoul.endpoint</name>
  <value>s3.ap-northeast-2.amazonaws.com</value>
</property>

<property>
  <name>singapore.endpoint</name>
  <value>s3-ap-southeast-1.amazonaws.com</value>
```

```
</property>

<property>
  <name>sydney.endpoint</name>
  <value>s3-ap-southeast-2.amazonaws.com</value>
</property>

<property>
  <name>tokyo.endpoint</name>
  <value>s3-ap-northeast-1.amazonaws.com</value>
</property>

<property>
  <name>virginia.endpoint</name>
  <value>${central.endpoint}</value>
</property>
```

The list above can be used to specify the endpoint of individual buckets. If you add these to your`core-site.xml`, you can then define per-bucket endpoints.

Example

The following examples show per-bucket endpoints set for the "landsat-pds" and "eu-dataset" buckets, with the endpoints set to central and EU/Ireland, respectively:

```
<property>
  <name>fs.s3a.bucket.landsat-pds.endpoint</name>
  <value>${central.endpoint}</value>
  <description>The endpoint for s3a://landsat-pds URLs</description>
</property>

<property>
  <name>fs.s3a.bucket.eu-dataset.endpoint</name>
  <value>${ireland.endpoint}</value>
  <description>The endpoint for s3a://eu-dataset URLs</description>
</property>
```

Explicitly declaring a bucket bound to the central endpoint ensures that if the default endpoint is changed to a new region, data stored in US-east is still reachable.

3.5. A List of S3A Configuration Properties

The following `fs.s3a` configuration properties are available. To override these default s3a settings, add your configuration to your `core-site.xml`.

```
<property>
  <name>fs.s3a.connection.maximum</name>
  <value>15</value>
  <description>Controls the maximum number of simultaneous connections to S3.
</description>
</property>

<property>
  <name>fs.s3a.connection.ssl.enabled</name>
  <value>true</value>
  <description>Enables or disables SSL connections to S3.</description>
</property>

<property>
```

```
  <name>fs.s3a.endpoint</name>
  <description>AWS S3 endpoint to connect to. An up-to-date list is
    provided in the AWS Documentation: regions and endpoints. Without this
    property, the standard region (s3.amazonaws.com) is assumed.
  </description>
</property>

<property>
  <name>fs.s3a.path.style.access</name>
  <value>false</value>
  <description>Enable S3 path style access ie disabling the default virtual
 hosting behaviour.
    Useful for S3A-compliant storage providers as it removes the need to set
 up DNS for virtual hosting.
  </description>
</property>

<property>
  <name>fs.s3a.proxy.host</name>
  <description>Hostname of the (optional) proxy server for S3 connections.</
description>
</property>

<property>
  <name>fs.s3a.proxy.port</name>
  <description>Proxy server port. If this property is not set
    but fs.s3a.proxy.host is, port 80 or 443 is assumed (consistent with
    the value of fs.s3a.connection.ssl.enabled).</description>
</property>

<property>
  <name>fs.s3a.proxy.username</name>
  <description>Username for authenticating with proxy server.</description>
</property>

<property>
  <name>fs.s3a.proxy.password</name>
  <description>Password for authenticating with proxy server.</description>
</property>

<property>
  <name>fs.s3a.proxy.domain</name>
  <description>Domain for authenticating with proxy server.</description>
</property>

<property>
  <name>fs.s3a.proxy.workstation</name>
  <description>Workstation for authenticating with proxy server.</description>
</property>

<property>
  <name>fs.s3a.attempts.maximum</name>
  <value>20</value>
  <description>How many times we should retry commands on transient errors.</
description>
</property>

<property>
  <name>fs.s3a.connection.establish.timeout</name>
  <value>5000</value>
```

```
  <description>Socket connection setup timeout in milliseconds.</description>
</property>

<property>
  <name>fs.s3a.connection.timeout</name>
  <value>200000</value>
  <description>Socket connection timeout in milliseconds.</description>
</property>

<property>
  <name>fs.s3a.paging.maximum</name>
  <value>5000</value>
  <description>How many keys to request from S3 when doing
     directory listings at a time.</description>
</property>

<property>
  <name>fs.s3a.threads.max</name>
  <value>10</value>
  <description> Maximum number of concurrent active (part)uploads,
  which each use a thread from the threadpool.</description>
</property>

<property>
  <name>fs.s3a.socket.send.buffer</name>
  <value>8192</value>
  <description>Socket send buffer hint to amazon connector. Represented in
 bytes.</description>
</property>

<property>
  <name>fs.s3a.socket.recv.buffer</name>
  <value>8192</value>
  <description>Socket receive buffer hint to amazon connector. Represented in
 bytes.</description>
</property>

<property>
  <name>fs.s3a.threads.keepalivetime</name>
  <value>60</value>
  <description>Number of seconds a thread can be idle before being
    terminated.</description>
</property>

<property>
  <name>fs.s3a.max.total.tasks</name>
  <value>5</value>
  <description>Number of (part)uploads allowed to the queue before
  blocking additional uploads.</description>
</property>

<property>
  <name>fs.s3a.multipart.size</name>
  <value>100M</value>
  <description>How big (in bytes) to split upload or copy operations up into.
    A suffix from the set {K,M,G,T,P} may be used to scale the numeric value.
  </description>
</property>

<property>
```

```
  <name>fs.s3a.multipart.threshold</name>
  <value>2147483647</value>
  <description>How big (in bytes) to split upload or copy operations up into.
    This also controls the partition size in renamed files, as rename()
 involves
    copying the source file(s).
    A suffix from the set {K,M,G,T,P} may be used to scale the numeric value.
  </description>
</property>

<property>
  <name>fs.s3a.multiobjectdelete.enable</name>
  <value>true</value>
  <description>When enabled, multiple single-object delete requests are
 replaced by
    a single 'delete multiple objects'-request, reducing the number of
 requests.
    Beware: legacy S3-compatible object stores might not support this request.
  </description>
</property>

<property>
  <name>fs.s3a.acl.default</name>
  <description>Set a canned ACL for newly created and copied objects. Value
 may be Private,
    PublicRead, PublicReadWrite, AuthenticatedRead, LogDeliveryWrite,
 BucketOwnerRead,
    or BucketOwnerFullControl.</description>
</property>

<property>
  <name>fs.s3a.multipart.purge</name>
  <value>false</value>
  <description>True if you want to purge existing multipart uploads that may
 not have been
     completed/aborted correctly</description>
</property>

<property>
  <name>fs.s3a.multipart.purge.age</name>
  <value>86400</value>
  <description>Minimum age in seconds of multipart uploads to purge</
description>
</property>

<property>
  <name>fs.s3a.signing-algorithm</name>
  <description>Override the default signing algorithm so legacy
    implementations can still be used</description>
</property>

<property>
  <name>fs.s3a.server-side-encryption-algorithm</name>
  <description>Specify a server-side encryption algorithm for s3a: file
 system.
    Unset by default, and the only other currently allowable value is AES256.
  </description>
</property>

<property>
```

```
  <name>fs.s3a.buffer.dir</name>
  <value>${hadoop.tmp.dir}/s3a</value>
  <description>Comma separated list of directories that will be used to buffer
 file
    uploads to. No effect if fs.s3a.fast.upload is true.</description>
</property>

<property>
  <name>fs.s3a.block.size</name>
  <value>32M</value>
  <description>Block size to use when reading files using s3a: file system.
  </description>
</property>

<property>
  <name>fs.s3a.user.agent.prefix</name>
  <value></value>
  <description>
    Sets a custom value that will be prepended to the User-Agent header sent
 in
    HTTP requests to the S3 back-end by S3AFileSystem.  The User-Agent header
    always includes the Hadoop version number followed by a string generated
 by
    the AWS SDK.  An example is "User-Agent: Hadoop 2.8.0, aws-sdk-java/1.10.
6".
    If this optional property is set, then its value is prepended to create a
    customized User-Agent.  For example, if this configuration property was
 set
    to "MyApp", then an example of the resulting User-Agent would be
    "User-Agent: MyApp, Hadoop 2.8.0, aws-sdk-java/1.10.6".
  </description>
</property>

<property>
  <name>fs.s3a.readahead.range</name>
  <value>64K</value>
  <description>Bytes to read ahead during a seek() before closing and
  re-opening the S3 HTTP connection. This option will be overridden if
  any call to setReadahead() is made to an open stream.</description>
</property>
```

3.6. Encrypting Data on S3

Amazon S3 supports a number of encryption mechanisms to better secure the data in S3:

- In Server-Side Encryption (SSE), the data is encrypted before it is saved to disk in S3, and decrypted when it is read. This encryption and decryption takes place in the S3 infrastructure, and is transparent to (authenticated) clients.

- In Client-Side Encryption (CSE), the data is encrypted and decrypted on the client, that is, inside the AWS S3 SDK. This mechanism isn't supported in Hadoop due to incompatibilities with most applications. Specifically, the amount of decrypted data is often less than the file length, breaking all the code which assumes that the the content of a file is the same size as that stated in directory listings.

Note

HDP **only** supports Server-Side Encryption ("SSE") and does **not** support Client-Side Encryption ("CSE").

For this server-side encryption to work, the S3 servers require secret keys to encrypt data, and the same secret keys to decrypt it. These keys can be managed in three ways:

- SSE-S3: By using Amazon S3-Managed Keys
- SSE-KMS: By using AWS Key Management Service
- SSE-C: By using customer-supplied keys

In general, the specific configuration mechanism can be set via the property `fs.s3a.server-side-encryption-algorithm` in `core-site.xml`. However, some encryption options require extra settings. Server Side encryption slightly slows down performance when reading data from S3.

It is possible to configure encryption for specific buckets and to mandate encryption for a specific S3 bucket.

Related Links

Troubleshooting S3-SSE [42]

3.6.1. SSE-S3: Amazon S3-Managed Encryption Keys

In SSE-S3, all keys and secrets are managed inside S3. This is the simplest encryption mechanism.

3.6.1.1. Enabling SSE-S3

To write S3-SSE encrypted files, the value of `fs.s3a.server-side-encryption-algorithm` must be set to that of the encryption mechanism used in `core-site.xml`; currently only AES256 is supported.

```
<property>
  <name>fs.s3a.server-side-encryption-algorithm</name>
  <value>AES256</value>
</property>
```

Once set, all new data will be uploaded encrypted. There is no need to set this property when downloading data — the data will be automatically decrypted when read using the Amazon S3-managed key.

To learn more, refer to Protecting Data Using Server-Side Encryption with Amazon S3-Managed Encryption Keys (SSE-S3) in AWS documentation.

3.6.2. SSE-KMS: Amazon S3-KMS Managed Encryption Keys

Amazon offers a pay-per-use key management service, AWS KMS. This service can be used to encrypt data on S3 using keys which can be centrally managed and assigned to specific roles and IAM accounts.

The AWS KMS can be used by S3 to encrypt uploaded data. When uploading data encrypted with SSE-KMS, the named key that was used to encrypt the data is retrieved from the KMS service, and used to encode the per-object secret which encrypts the uploaded data. To decode the data, the same key must be retrieved from KMS and used to unencrypt the per-object secret key, which is then used to decode the actual file.

KMS keys can be managed by an organization's administrators in AWS, including having access permissions assigned and removed from specific users, groups, and IAM roles. Only those "principals" with granted rights to a key may access it, hence only they may encrypt data with the key, *and decrypt data encrypted with it*. This allows KMS to be used to provide a cryptographically secure access control mechanism for data stores on S3.

Note

AWS KMS service is **not** related to the Key Management Service built into Hadoop (*Hadoop KMS*). The *Hadoop KMS* primarily focuses on managing keys for *HDFS Transparent Encryption*. Similarly, HDFS encryption is unrelated to S3 data encryption.

3.6.2.1. Enabling SSE-KMS

To enable SSE-KMS, the property `fs.s3a.server-side-encryption-algorithm` must be set to SSE-KMS in `core-site.xml`:

```
<property>
  <name>fs.s3a.server-side-encryption-algorithm</name>
  <value>SSE-KMS</value>
</property>
```

The ID of the specific key used to encrypt the data should also be set in the property `fs.s3a.server-side-encryption.key`:

```
<property>
  <name>fs.s3a.server-side-encryption.key</name>
  <value>arn:aws:kms:us-west-2:360379543683:key/
071a86ff-8881-4ba0-9230-95af6d01ca01</value>
</property>
```

If your account is set up set up with a default KMS key and `fs.s3a.server-side-encryption.key` is unset, the default key will be used.

Alternatively, organizations may define a default key in the Amazon KMS; if a default key is set, then it will be used whenever SSE-KMS encryption is chosen and the value of `fs.s3a.server-side-encryption.key` is empty.

Note

AWS Key Management Service (KMS) is pay-per-use, working with data encrypted via KMS keys incurs extra charges during data I/O.

To learn more, refer to Protecting Data Using Server-Side Encryption with AWS KMS-Managed Keys (SSE-KMS) in AWS documentation.

3.6.3. SSE-C: Server-Side Encryption with Customer-Provided Encryption Keys

In SSE-C, the client supplies the secret key needed to read and write data.

Note

SSE-C integration with Hadoop is still stabilizing; issues related to it are still surfacing. It is already clear that SSE-C with a common key **must** be used exclusively within a bucket if it is to be used at all. This is the only way to ensure that path and directory listings do not fail with "Bad Request" errors.

3.6.3.1. Enabling SSE-C

To use SSE-C, the configuration option `fs.s3a.server-side-encryption-algorithm` must be set to `SSE-C`, and a base-64 encoding of the key placed in `fs.s3a.server-side-encryption.key`.

```
<property>
  <name>fs.s3a.server-side-encryption-algorithm</name>
  <value>SSE-C</value>
</property>

<property>
  <name>fs.s3a.server-side-encryption.key</name>
  <value>RG8gbm90IGV2ZXIgbG9nIHRoaXMga2V5IG9yIG90aGVyd2lzZSBzaGFyZSBpdA==</
value>
</property>
```

This property can be set in a Hadoop JCEKS credential file, which is significantly more secure than embedding secrets in the XML configuration file.

3.6.4. Configuring Encryption for Specific Buckets

S3A's per-bucket configuration mechanism can be used to configure the encryption mechanism and credentials for specific buckets. For example, to access the bucket called "production" using SSE-KMS with the key ID `arn:aws:kms:us-west-2:360379543683:key/071a86ff-8881-4ba0-9230-95af6d01ca01`, the settings are:

```
<property>
  <name>fs.s3a.bucket.production.server-side-encryption-algorithm</name>
  <value>SSE-KMS</value>
</property>

<property>
  <name>fs.s3a.bucket.production.server-side-encryption.key</name>
  <value>arn:aws:kms:us-west-2:360379543683:key/
071a86ff-8881-4ba0-9230-95af6d01ca01</value>
</property>
```

Per-bucket configuration does not apply to secrets kept in JCEKS files; the core configuration properties must be used (for example `fs.s3a.server-side-encryption.key`), with the path to the JCEKS file instead configured for the bucket:

```
<property>
  <name>fs.s3a.bucket.production.server-side-encryption-algorithm</name>
  <value>SSE-KMS</value>
</property>

<property>`
  <name>fs.s3a.bucket.production.security.credential.provider.path</name>
  <value>hdfs://common/production.jceks</value>
</property>
```

To learn more, refer to Protecting Data Using Server-Side Encryption with Customer-Provided Encryption Keys (SSE-C) in AWS documentation.

3.6.5. Mandating Encryption for an S3 Bucket

To mandate that all data uploaded to a bucket is encrypted, it is possible to set a bucket policy declaring that clients must provide encryption information with all data uploaded.

Mandating encryption across a bucket offers significant benefits:

1. It guarantees that all clients uploading data have encryption enabled; there is no need (or indeed, easy mechanism) to test this within a client.

2. It guarantees that the same encryption mechanism is used by all clients.

3. If applied to an empty bucket, it guarantees that all data in the bucket is encrypted.

We recommend selecting an encryption policy for a bucket when the bucket is created, and setting it in the bucket policy. This stops misconfigured clients from unintentionally uploading unencrypted data.

Note

Mandating an encryption mechanism on newly uploaded data does not encrypt existing data; existing data will retain whatever encryption (if any) applied at the time of creation.

Here is a policy to mandate SSE-S3/AES265 encryption on all data uploaded to a bucket. This covers uploads as well as the copy operations which take place when file/directory rename operations are mimicked.

```
{
  "Version": "2012-10-17",
  "Id": "EncryptionPolicy",
  "Statement": [ { "Sid": "RequireEncryptionHeaderOnPut", "Effect":
 "Deny", "Principal": "*", "Action": [ "s3:PutObject" ], "Resource":
 "arn:aws:s3:::BUCKET/*", "Condition": { "Null": { "s3:x-amz-server-side-
encryption": true } } }, { "Sid": "RequireAESEncryptionOnPut", "Effect":
 "Deny", "Principal": "*", "Action": [ "s3:PutObject" ], "Resource":
 "arn:aws:s3:::BUCKET/*", "Condition": { "StringNotEquals": { "s3:x-amz-
server-side-encryption": "AES256" } } } ] }
```

To use SSE-KMS, a different restriction must be defined:

```
{
  "Version": "2012-10-17",
  "Id": "EncryptionPolicy",
  "Statement": [ { "Sid": "RequireEncryptionHeaderOnPut", "Effect":
 "Deny", "Principal": "*", "Action": [ "s3:PutObject" ], "Resource":
 "arn:aws:s3:::BUCKET/*", "Condition": { "Null": { "s3:x-amz-server-side-
encryption": true } } }, { "Sid": "RequireKMSEncryptionOnPut", "Effect":
 "Deny", "Principal": "*", "Action": [ "s3:PutObject" ], "Resource":
 "arn:aws:s3:::BUCKET/*", "Condition": { "StringNotEquals": { "s3:x-amz-
server-side-encryption": "SSE-KMS" } } } ] }
```

To use one of these policies:

1. Replace BUCKET with the specific name of the bucket being secured.
2. Locate the bucket in the AWS console S3 section.
3. Select the "Permissions" tab.
4. Select the "Bucket Policy" tab in the permissions section.
5. Paste the edited policy into the editor.
6. Save the policy.

3.6.6. Performance Impact of Encryption

Server Side encryption slightly slows down performance when reading data from S3, both in the reading of data during the execution of a query, and in scanning the files prior to the actual scheduling of work.

Amazon throttles reads and writes of S3-SSE data, which results in a significantly lower throughput than normal S3 IO requests. The default rate, 600 requests/minute, means that at most ten objects per second can be read or written using SSE-KMS per second — across an entire hadoop cluster (or indeed, the entire customer account). The default limits may be suitable during development — but in large scale production applications the limits may rapidly be reached. Contact Amazon to increase capacity.

3.7. Improving Performance for S3

Use this checklist to ensure optimal performance when working with data in S3.

Checklist for Data

- [ ] Amazon S3 bucket is in same region as the EC2-hosted cluster. Learn more
- [ ] The directory layout is "shallow". For directory listing performance, the directory layout prefers "shallow" directory trees with many files over deep directory trees with only a few files per directory.
- [ ] The "pseudo" block size set in `fs.s3a.block.size` is appropriate for the work to be performed on the data.
- [ ] Copy to HDFS any data that needs to be repeatedly read to HDFS.

Checklist for Cluster Configs

- [ ] Set `yarn.scheduler.capacity.node-locality-delay` to 0 to improve container launch times. Learn more
- [ ] When copying data using DistCp, use the following performance optimizations.
- [ ] When reading ORC data, set `fs.s3a.experimental.input.fadvise` to `random`. Learn more
- [ ] If planning to use Hive with S3, review Improving Hive Performance with S3/ADLS/WASB.
- [ ] If planning to use Spark with S3, review Improving Spark Performance with S3/ADLS/WASB.

Checklist for Code

- [ ] Application does not make `rename()` calls. Where it does, it does not assume the operation is immediate.
- [ ] Application does not assume that `delete()` is near-instantaneous.
- [ ] Application uses `FileSystem.listFiles(path, recursive=true)` to list a directory tree.
- [ ] Application prefers forward seeks through files, rather than full random IO.
- [ ] If making "random" IO through `seek()` and `read()` sequences or and Hadoop's `PositionedReadable` API, `fs.s3a.experimental.input.fadvise` is set to `random`. Learn more

More

- [ ] Improve Load-Balancing Behavior for S3.

3.7.1. Improving DistCp Performance with S3

This section includes tips for improving performance when copying large volumes of data between Amazon S3 and HDFS.

The bandwidth between the Hadoop cluster and Amazon S3 is the upper limit to how fast data can be copied into S3. The further the Hadoop cluster is from the Amazon S3 installation, or the narrower the network connection is, the longer the operation will take. Even a Hadoop cluster deployed within Amazon's own infrastructure may encounter network delays from throttled VM network connections.

Network bandwidth limits notwithstanding, there are some options which can be used to tune the performance of an upload:

- Working with Local S3 Buckets [29]
- Accelerating File Listing [29]
- Configuring and Tuning S3A Fast Upload [29]

- Controlling the Number of Mappers and Their Bandwidth [33]

3.7.1.1. Working with Local S3 Buckets

A foundational step to getting good performance is working with buckets close to the Hadoop cluster, where "close" is measured in network terms.

Maximum performance is achieved from working with S3 buckets in the same AWS region as the cluster. For example, if your cluster is in North Virginia ("US East"), you will achieve best performance if your S3 bucket is in the same region.

In addition to improving performance, working with local buckets ensures that no bills are incurred for reading from the bucket.

3.7.1.2. Accelerating File Listing

When data is copied between buckets, listing all the files to copy can take a long time. In such cases, you can increase `-numListstatusThreads` from 1 (default) to 15. With this setting, multiple threads will be used for listing the contents of the source folder.

3.7.1.3. Configuring and Tuning S3A Fast Upload

Note

These tuning recommendations are experimental and may change in the future.

Because of the nature of the S3 object store, data written to an S3A `OutputStream` is not written incrementally — instead, by default, it is buffered to disk until the stream is closed in its `close()` method. This can make output slow because the execution time for `OutputStream.close()` is proportional to the amount of data buffered and inversely proportional to the bandwidth between the host to S3; that is `O(data/bandwidth)`. Other work in the same process, server, or network at the time of upload may increase the upload time.

In summary, the further the process is from the S3 endpoint, or the smaller the EC2 VM is, the longer it will take complete the work. This can create problems in application code:

- Code often assumes that the `close()` call is fast; the delays can create bottlenecks in operations.
- Very slow uploads sometimes cause applications to time out - generally, threads blocking during the upload stop reporting progress, triggering timeouts.
- Streaming very large amounts of data may consume all disk space before the upload begins.

3.7.1.3.1. Enabling S3A Fast Upload

To enable the fast upload mechanism, set the `fs.s3a.fast.upload` property (it is disabled by default).

When this is set, the incremental block upload mechanism is used, with the buffering mechanism set in `fs.s3a.fast.upload.buffer`. The number of threads performing

uploads in the filesystem is defined by `fs.s3a.threads.max`; the queue of waiting uploads limited by `fs.s3a.max.total.tasks`. The size of each buffer is set by `fs.s3a.multipart.size`.

3.7.1.3.2. Configuring S3A Fast Upload Options

The following major configuration options are available for the S3A fast upload:

Table 3.2. S3A Fast Upload Configuration Options

Parameter	Default Value	Description
`fs.s3a.fast.upload.buffer`	disk	The `fs.s3a.fast.upload.buffer` determines the buffering mechanism to use when `fs.s3a.fast.upload` is set to "true"; it has no effect when `fs.s3a.fast.upload` is false. Allowed values are: disk, array, bytebuffer: • (default) "disk" will use the directories listed in `fs.s3a.buffer.dir` as the location(s) to save data prior to being uploaded. • "array" uses arrays in the JVM heap. • "bytebuffer" uses off-heap memory within the JVM. Both "array" and "bytebuffer" will consume memory in a single stream up to the number of blocks set by: `fs.s3a.multipart.size` * `fs.s3a.fast.upload.active.blocks`. If using either of these mechanisms, keep this value low. The total number of threads performing work across all threads is set by `fs.s3a.threads.max`, with `fs.s3a.max.total.tasks` values setting the number of queued work items.
`fs.s3a.multipart.size`	100M	Defines the size (in bytes) of the chunks into which the upload or copy operations will be split up. A suffix from the set {K,M,G,T,P} may be used to scale the numeric value.
`fs.s3a.fast.upload.active.block`	8	Defines the maximum number of blocks a single output stream can have active uploading, or queued to the central FileSystem instance's pool of queued operations. This stops a single stream overloading the shared thread pool.
`fs.s3a.buffer.dir`	Empty value	Allows you to add a comma separated list of temporary directories use for storing blocks of data prior to their being uploaded to S3. When unset (by default), the Hadoop temporary directory `hadoop.tmp.dir` is used.

Note that:

- If the amount of data written to a stream is below that set in `fs.s3a.multipart.size`, the upload is performed in the `OutputStream.close()` operation —as with the original output stream.

- The published Hadoop metrics monitor includes live queue length and upload operation counts, so identifying when there is a backlog of work or a mismatch between data generation rates and network bandwidth. Per-stream statistics can also be logged by calling `toString()` on the current stream.
- Incremental writes are not visible; the object can only be listed or read when the multipart operation completes in the `close()` call, which will block until the upload is completed.

Fast Upload with Disk Buffers

This is the default buffer mechanism. The amount of data which can be buffered is limited by the amount of available disk space.

When `fs.s3a.fast.upload.buffer` is set to "disk", all data is buffered to local hard disks prior to upload. This minimizes the amount of memory consumed, and so eliminates heap size as the limiting factor in queued uploads — exactly as the original "direct to disk" buffering used when `fs.s3a.fast.upload=false`.

Fast Upload with Byte Buffers

When `fs.s3a.fast.upload.buffer` is set to "bytebuffer", all data is buffered in "direct" ByteBuffers prior to upload. This *may* be faster than buffering to disk in cases such as when disk space is small there may not be much disk space to buffer with (for example, when using tiny EC2 VMs).

The ByteBuffers are created in the memory of the JVM, but not in the Java Heap itself. The amount of data which can be buffered is limited by the Java runtime, the operating system, and, for YARN applications, the amount of memory requested for each container.

The slower the upload bandwidth to S3, the greater the risk of running out of memory — and so the more care is needed in tuning the upload thread settings to reduce the maximum amount of data which can be buffered awaiting upload (see below).

Fast Upload with Array Buffers

When `fs.s3a.fast.upload.buffer` is set to "array", all data is buffered in byte arrays in the JVM's heap prior to upload. This *may* be faster than buffering to disk.

The amount of data which can be buffered is limited by the available size of the JVM heap heap. The slower the write bandwidth to S3, the greater the risk of heap overflows. This risk can be mitigated by tuning the upload thread settings (see below).

3.7.1.1.3.3. Thread Tuning for S3A Fast Upload

Both the array and bytebuffer buffer mechanisms can consume very large amounts of memory, on-heap or off-heap respectively. The disk buffer mechanism does not use much memory up, but it consumes hard disk capacity.

If there are many output streams being written to in a single process, the amount of memory or disk used is the multiple of all stream's active memory and disk use.

You may need to perform careful tuning to reduce the risk of running out memory, especially if the data is buffered in memory. There are a number parameters which can be tuned:

1. The total number of threads available in the filesystem for data uploads or any other queued filesystem operation. This is set in `fs.s3a.threads.max`.

2. The number of operations which can be queued for execution, awaiting a thread. This is set in `fs.s3a.max.total.tasks`.

3. The number of blocks which a single output stream can have active (that is, being uploaded by a thread or queued in the filesystem thread queue). This is set in `fs.s3a.fast.upload.active.blocks`.

4. The length of time that an idle thread can stay in the thread pool before it is retired. This is set in `fs.s3a.threads.keepalivetime`.

Table 3.3. S3A Fast Upload Tuning Options

Parameter	Default Value	Description
`fs.s3a.fast.upload.active.blocks`	4	Maximum number of blocks a single output stream can have active (uploading, or queued to the central FileSystem instance's pool of queued operations). This stops a single stream overloading the shared thread pool.
`fs.s3a.threads.max`	10	The total number of threads available in the filesystem for data uploads or any other queued filesystem operation.
`fs.s3a.max.total.tasks`	5	The number of operations which can be queued for execution
`fs.s3a.threads.keepalivetime`	60	The number of seconds a thread can be idle before being terminated.

When the maximum allowed number of active blocks of a single stream is reached, no more blocks can be uploaded from that stream until one or more of those active block uploads completes. That is, a `write()` call which would trigger an upload of a now full datablock will instead block until there is capacity in the queue.

Consider the following:

- As the pool of threads set in `fs.s3a.threads.max` is shared (and intended to be used across all threads), a larger number here can allow for more parallel operations. However, as uploads require network bandwidth, adding more threads does not guarantee speedup.

- The extra queue of tasks for the thread pool (`fs.s3a.max.total.tasks`) covers all ongoing background S3A operations.

- When using memory buffering, a small value of `fs.s3a.fast.upload.active.blocks` limits the amount of memory which can be consumed per stream.

- When using disk buffering, a larger value of `fs.s3a.fast.upload.active.blocks` does not consume much memory. But it may result in a large number of blocks to compete with other filesystem operations.

We recommend a low value of `fs.s3a.fast.upload.active.blocks` — enough to start background upload without overloading other parts of the system. Then experiment to see if higher values deliver more throughput — especially from VMs running on EC2.

3.7.1.4. Controlling the Number of Mappers and Their Bandwidth

If you want to control the number of mappers launched for DistCp, you can add the `-m` option and set it to the desired number of mappers.

When using DistCp from a Hadoop cluster running in Amazon's infrastructure, increasing the number of mappers may speed up the operation.

Similarly, if copying to S3 from a cluster in a different region, it is possible that the bandwidth from the Hadoop cluster to Amazon S3 is the bottleneck. In such a situation, because the bandwidth is shared across all mappers, adding more mappers will not accelerate the upload: it will merely slow all the mappers down.

The `-bandwidth` option sets the approximate maximum bandwidth for each mapper in Megabytes per second. This a floating point number, so a value such as `-bandwidth 0.5` allocates 0.5 MB/s to each mapper.

3.7.2. Improving Container Allocation Performance for S3

As AWS does not have the concept of rack locality, set `yarn.scheduler.capacity.node-locality-delay = 0` to have faster container launches when using the Capacity Scheduler.

For more information, refer to the Apache documentation.

3.7.3. Optimizing HTTP Get Requests for S3

Note

This feature is experimental and its behavior may change in the future.

The S3A filesystem client supports the notion of input policies, similar to that of the POSIX `fadvise()` API call. This tunes the behavior of the S3A client to optimize HTTP GET requests for various use cases. To optimize HTTP GET requests, you can take advantage of the S3A experimental input policy `fs.s3a.experimental.input.fadvise`:

Policy	Description
"sequential" (default)	Read through the file, possibly with some short forward seeks. The whole document is requested in a single HTTP request; forward seeks within the readahead range are supported by skipping over the intermediate data. This leads to maximum read throughput, but with very expensive backward seeks.
"normal"	This is currently the same as "sequential".
"random"	Optimized for random IO, specifically the Hadoop `PositionedReadable` operations — though `seek(offset); read(byte_buffer)` also benefits. Rather than ask for the whole file, the range of the HTTP request is set to that of the length of data desired in the `read` operation - rounded up to the readahead value set in `setReadahead()` if necessary.

Policy	Description
	By reducing the cost of closing existing HTTP requests, this is highly efficient for file IO accessing a binary file through a series of `PositionedReadable.read()` and `PositionedReadable.readFully()` calls. Sequential reading of a file is expensive, as now many HTTP requests must be made to read through the file.

For operations simply reading through a file (copying, DistCp, reading gzip or other compressed formats, parsing .csv files, and so on) the `sequential` policy is appropriate. This is the default, so you don't need to configure it.

For the specific case of high-performance random access IO (for example, accessing ORC files), you may consider using the `random` policy in the following circumstances:

- Data is read using the `PositionedReadable` API.
- There are long distance (many MB) forward seeks.
- Backward seeks are as likely as forward seeks.
- There is little or no use of single character `read()` calls or small `read(buffer)` calls.
- Applications are running close to the Amazon S3 data store; that is, the EC2 VMs on which the applications run are in the same region as the Amazon S3 bucket.

You must set the desired fadvise policy in the configuration option `fs.s3a.experimental.input.fadvise` when the filesystem instance is created. It can only be set on a per-filesystem basis, not on a per-file-read basis. You can set it in `core-site.xml`:

```
<property>
  <name>fs.s3a.experimental.input.fadvise</name>
  <value>random</value>
</property>
```

Or, you can set it in the `spark-defaults.conf` configuration of Spark:

```
spark.hadoop.fs.s3a.experimental.input.fadvise random
```

Be aware that this random access performance comes at the expense of sequential IO — which includes reading files compressed with gzip.

3.7.4. Improving Load-Balancing Behavior for S3

S3 uses a set of front-end servers to provide access to the underlying data. The decision about which front-end server to use is handled via load-balancing DNS service. When the IP address of an S3 bucket is looked up, the choice of which IP address to return to the client is made based on the current load of the front-end servers.

Over time, the load across the front-end changes, so those servers that are considered "lightly loaded" change. This means that if the DNS value is cached for any length of time, applications may end up talking to an overloaded server; or, in the case of failures, they may end up trying to talk to a server that is no longer there.

And, for historical security reasons, in the era of applets, the DNS TTL of a JVM is set to "infinity" by default.

To improve AWS load-balancing, set the DNS time-to-live of an application which works with Amazon S3 to something lower than the default. Refer to Setting the JVM TTL for DNS Name Lookups in the AWS documentation.

3.8. Troubleshooting S3

Use these tips to troubleshoot errors. Common problems that you may encounter while working with Amazon S3 include:

- Authentication Failures [35]
- Classpath Related Errors [37]
- Connectivity Problems [38]
- Errors During Delete or Rename of Files [41]
- Errors Related to Visible S3A Inconsistency [41]
- Troubleshooting S3-SSE [42]

3.8.1. Authentication Failures

You may encounter the following S3 authentication issues.

3.8.1.1. Authentication Failure Due to Signature Mismatch

If Hadoop cannot authenticate with the S3 service endpoint, the client retries a number of times before eventually failing. When it finally gives up, it will report a message about signature mismatch:

```
com.amazonaws.services.s3.model.AmazonS3Exception:
 The request signature we calculated does not match the signature you
 provided.
 Check your key and signing method.
  (Service: AmazonS3; StatusCode: 403; ErrorCode: SignatureDoesNotMatch,
```

The likely cause is that you either have the wrong credentials for any of the current authentication mechanism(s) — or somehow the credentials were not readable on the host attempting to read or write the S3 bucket.

Enabling debug logging for the package `org.apache.hadoop.fs.s3a` can help provide more information.

1. The standard first step is: try to use the AWS command line tools with the same credentials, through a command such as:

   ```
   hdfs fs -ls s3a://my-bucket/
   ```

   Note the trailing "/" here; without that the shell thinks you are trying to list your home directory under the bucket, which will only exist if explicitly created.

Attempting to list a bucket using inline credentials is a means of verifying that the key and secret can access a bucket:

```
hdfs fs -ls s3a://key:secret@my-bucket/
```

Do escape any + or / symbols in the secret, as discussed below, and never share the URL, logs generated using it, or use such an inline authentication mechanism in production.

Finally, if you set the environment variables, you can take advantage of S3A's support of environment-variable authentication by attempting the same ls operation; that is, unset the `fs.s3a` secrets and rely on the environment variables.

2. Make sure that the name of the bucket is the correct one. That is, check the URL.
3. Make sure the property names are correct. For S3A, they are `fs.s3a.access.key` and `fs.s3a.secret.key`. You cannot just copy the S3N properties and replace `s3n` with `s3a`.
4. Make sure that the properties are visible to the process attempting to talk to the object store. Placing them in `core-site.xml` is the standard mechanism.
5. If using session authentication, the session may have expired. Generate a new session token and secret.
6. If using environment variable-based authentication, make sure that the relevant variables are set in the environment in which the process is running.
7. There are a couple of system configuration problems (JVM version, system clock) that you should check.

3.8.1.2. Authentication Failure Due to Clock Skew

The timestamp is used in signing to S3, so as to defend against replay attacks. If the system clock is too far behind *or ahead* of Amazon's, requests will be rejected.

This can surface as the situation where read requests are allowed, but operations which write to the bucket are denied.

Solution: Check the system clock.

3.8.1.3. Authentication Failure When Using URLs with Embedded Secrets

If you are using the strongly discouraged mechanism of including the AWS key and secret in a URL, make sure that both "+" and "/" symbols are encoded in the URL. As many AWS secrets include these characters, encoding problems are not uncommon.

Use this table for conversion:

Symbol	Encoded Value
+	%2B
/	%2F

For example, a URL for an S3 `bucket` with AWS ID `user1` and secret `a+b/c` will be represented as

```
s3a://user1:a%2Bb%2Fc@bucket
```

You only need to use this technique when placing secrets in the URL.

3.8.1.4. Authentication Failures When Running on Java 8u60+

A change in the Java 8 JVM broke some of the `toString()` string generation of Joda Time 2.8.0, which stopped the Amazon S3 client from being able to generate authentication headers suitable for validation by S3.

Solution: Make sure that the version of Joda Time is 2.8.1 or later, or use a new version of Java 8.

3.8.2. Classpath Related Errors

The Hadoop S3 filesystem clients need the Hadoop-specific filesystem clients and third party S3 client libraries to be compatible with the Hadoop code, and any dependent libraries to be compatible with Hadoop and the specific JVM.

The classpath must be set up for the process talking to S3. If this is code running in the Hadoop cluster, then the JARs must be on that classpath. This includes `distcp`.

3.8.2.1. ClassNotFoundException Errors

```
ClassNotFoundException: org.apache.hadoop.fs.s3a.S3AFileSystem
ClassNotFoundException: org.apache.hadoop.fs.s3native.NativeS3FileSystem
ClassNotFoundException: org.apache.hadoop.fs.s3.S3FileSystem
```

These are the Hadoop classes, found in the `hadoop-aws` JAR. An exception reporting that one of these classes is missing means that this JAR is not on the classpath.

Similarly, this error

```
ClassNotFoundException: com.amazonaws.services.s3.AmazonS3Client
```

or similar errors related to another `com.amazonaws` class mean that one or more of the `aws-*-sdk` JARs are missing.

To solve the issue, add the missing JARs to the classpath.

3.8.2.2. Missing Method in com.amazonaws Class

This can be triggered by incompatibilities between the AWS SDK on the classpath and the version with which Hadoop was compiled.

The AWS SDK JARs change their signature between releases often, so the only way to safely update the AWS SDK version is to recompile Hadoop against the later version.

There is nothing the Hadoop team can do here; if you get this problem, then you are on your own. The Hadoop developer team did look at using reflection to bind to the SDK,

but there were too many changes between versions for this to work reliably. All it did was postpone version compatibility problems until the specific codepaths were executed at runtime. This was actually a backward step in terms of fast detection of compatibility problems.

3.8.2.3. Missing Method in a Jackson Class

This is usually caused by version mismatches between Jackson JARs on the classpath. All Jackson JARs on the classpath must be of the same version.

3.8.3. Connectivity Problems

You may encounter the following S3 connectivity issues.

3.8.3.1. Unable to Execute HTTP Request: Read Timed Out

A read timeout means that the S3A client could not talk to the S3 service, and eventually gave up trying:

```
Unable to execute HTTP request: Read timed out
java.net.SocketTimeoutException: Read timed out
    at java.net.SocketInputStream.socketRead0(Native Method)
    at java.net.SocketInputStream.socketRead(SocketInputStream.java:116)
    at java.net.SocketInputStream.read(SocketInputStream.java:170)
    at java.net.SocketInputStream.read(SocketInputStream.java:141)
    at org.apache.http.impl.io.AbstractSessionInputBuffer.
fillBuffer(AbstractSessionInputBuffer.java:166)
    at org.apache.http.impl.io.SocketInputBuffer.fillBuffer(SocketInputBuffer.
java:90)
    at org.apache.http.impl.io.AbstractSessionInputBuffer.
readLine(AbstractSessionInputBuffer.java:281)
    at org.apache.http.impl.conn.DefaultHttpResponseParser.
parseHead(DefaultHttpResponseParser.java:92)
    at org.apache.http.impl.conn.DefaultHttpResponseParser.
parseHead(DefaultHttpResponseParser.java:62)
    at org.apache.http.impl.io.AbstractMessageParser.
parse(AbstractMessageParser.java:254)
    at org.apache.http.impl.AbstractHttpClientConnection.
receiveResponseHeader(AbstractHttpClientConnection.java:289)
    at org.apache.http.impl.conn.DefaultClientConnection.
receiveResponseHeader(DefaultClientConnection.java:252)
    at org.apache.http.impl.conn.ManagedClientConnectionImpl.
receiveResponseHeader(ManagedClientConnectionImpl.java:191)
    at org.apache.http.protocol.HttpRequestExecutor.
doReceiveResponse(HttpRequestExecutor.java:300)
    at com.amazonaws.http.protocol.SdkHttpRequestExecutor.
doReceiveResponse(SdkHttpRequestExecutor.java:66)
    at org.apache.http.protocol.HttpRequestExecutor.
execute(HttpRequestExecutor.java:127)
    at org.apache.http.impl.client.DefaultRequestDirector.
createTunnelToTarget(DefaultRequestDirector.java:902)
    at org.apache.http.impl.client.DefaultRequestDirector.
establishRoute(DefaultRequestDirector.java:821)
    at org.apache.http.impl.client.DefaultRequestDirector.
tryConnect(DefaultRequestDirector.java:647)
    at org.apache.http.impl.client.DefaultRequestDirector.
execute(DefaultRequestDirector.java:479)
```

```
    at org.apache.http.impl.client.AbstractHttpClient.
execute(AbstractHttpClient.java:906)
    at org.apache.http.impl.client.AbstractHttpClient.
execute(AbstractHttpClient.java:805)
    at com.amazonaws.http.AmazonHttpClient.executeHelper(AmazonHttpClient.
java:384)
    at com.amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:232)
    at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.
java:3528)
    at com.amazonaws.services.s3.AmazonS3Client.getObject(AmazonS3Client.
java:1111)
    at org.apache.hadoop.fs.s3a.S3AInputStream.reopen(S3AInputStream.java:91)
```

This is not uncommon in Hadoop client applications — there is a whole wiki entry dedicated to possible causes of the error.

For S3 connections, key causes are:

- The S3 endpoint property `fs.s3a.endpoint` for the target bucket is invalid.
- There's a proxy setting for the S3 client, and the proxy is not reachable or is on a different port.
- The caller is on a host with fundamental connectivity problems. If a VM is on EC2, consider releasing it and requesting a new one.

3.8.3.2. Bad Request Exception When Working with S3 Frankfurt, Seoul, or Elsewhere

S3 Frankfurt and Seoul only support the V4 authentication API. Consequently, any requests using the V2 API will be rejected with 400 `Bad Request`:

```
$ bin/hadoop fs -ls s3a://frankfurt/
WARN s3a.S3AFileSystem:Client: Amazon S3 error 400: 400 Bad Request; Bad
 Request (retryable)

com.amazonaws.services.s3.model.AmazonS3Exception: Bad Request
 (Service: Amazon S3; Status Code:400; Error Code:400 Bad Request;
 Request ID:923C5D9E75E44C06), S3 Extended Request ID: HDwje6k
+ANEeDsM6aJ8+D5gUmNAMguOk2BvZ8PH3g9z0gpH+IuwT7N19oQOnIr5CIx7Vqb/uThE=
    at com.amazonaws.http.AmazonHttpClient.
handleErrorResponse(AmazonHttpClient.java:1182)
    at com.amazonaws.http.AmazonHttpClient.executeOneRequest(AmazonHttpClient.
java:770)
    at com.amazonaws.http.AmazonHttpClient.executeHelper(AmazonHttpClient.
java:489)
    at com.amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:310)
    at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.
java:3785)
    at com.amazonaws.services.s3.AmazonS3Client.headBucket(AmazonS3Client.
java:1107)
    at com.amazonaws.services.s3.AmazonS3Client.
doesBucketExist(AmazonS3Client.java:1070)
    at org.apache.hadoop.fs.s3a.S3AFileSystem.
verifyBucketExists(S3AFileSystem.java:307)
    at org.apache.hadoop.fs.s3a.S3AFileSystem.initialize(S3AFileSystem.
java:284)
    at org.apache.hadoop.fs.FileSystem.createFileSystem(FileSystem.java:2793)
```

```
    at org.apache.hadoop.fs.FileSystem.access$200(FileSystem.java:101)
    at org.apache.hadoop.fs.FileSystem$Cache.getInternal(FileSystem.java:2830)
    at org.apache.hadoop.fs.FileSystem$Cache.get(FileSystem.java:2812)
    at org.apache.hadoop.fs.FileSystem.get(FileSystem.java:389)
    at org.apache.hadoop.fs.Path.getFileSystem(Path.java:356)
    at org.apache.hadoop.fs.shell.PathData.expandAsGlob(PathData.java:325)
    at org.apache.hadoop.fs.shell.Command.expandArgument(Command.java:235)
    at org.apache.hadoop.fs.shell.Command.expandArguments(Command.java:218)
    at org.apache.hadoop.fs.shell.FsCommand.processRawArguments(FsCommand.
java:103)
    at org.apache.hadoop.fs.shell.Command.run(Command.java:165)
    at org.apache.hadoop.fs.FsShell.run(FsShell.java:315)
    at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:76)
    at org.apache.hadoop.util.ToolRunner.run(ToolRunner.java:90)
    at org.apache.hadoop.fs.FsShell.main(FsShell.java:373)
ls: doesBucketExist on frankfurt-new: com.amazonaws.services.s3.model.
AmazonS3Exception:
  Bad Request (Service: Amazon S3; Status Code:400; Error Code:400 Bad
 Request;
```

This happens when you are trying to work with any S3 service which only supports the "V4" signing API — and the client is configured to use the default S3A service endpoint.

To avoid this error, set the specific endpoint to use via the `fs.s3a.endpoint` property. For more information, refer to Configuring Per-Bucket Settings to Access Data Around the World.

3.8.3.3. Error Message "The bucket you are attempting to access must be addressed using the specified endpoint"

This surfaces when `fs.s3a.endpoint` is configured to use S3 service endpoint which is neither the original AWS one (`s3.amazonaws.com`) nor the one where the bucket is hosted.

```
org.apache.hadoop.fs.s3a.AWSS3IOException: purging multipart uploads on
 landsat-pds:
 com.amazonaws.services.s3.model.AmazonS3Exception:
  The bucket you are attempting to access must be addressed usingthe specified
 endpoint.
  Please send all future requests to this endpoint.
   (Service: Amazon S3; Status Code: 301; Error Code: PermanentRedirect;
 Request ID: 5B7A5D18BE596E4B),
    S3 Extended Request ID: uE4pbbmpxi8Nh7rycS6GfIEi9UH/SWmJfGtM9IeKvRyBPZp/
hN7DbPyz272eynz3PEMM2azlhjE=:

    at com.amazonaws.http.AmazonHttpClient.
handleErrorResponse(AmazonHttpClient.java:1182)
    at com.amazonaws.http.AmazonHttpClient.executeOneRequest(AmazonHttpClient.
java:770)
    at com.amazonaws.http.AmazonHttpClient.executeHelper(AmazonHttpClient.
java:489)
    at com.amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:310)
    at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.
java:3785)
    at com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.
java:3738)
    at com.amazonaws.services.s3.AmazonS3Client.
listMultipartUploads(AmazonS3Client.java:2796)
```

```
    at com.amazonaws.services.s3.transfer.TransferManager.
abortMultipartUploads(TransferManager.java:1217)
    at org.apache.hadoop.fs.s3a.S3AFileSystem.
initMultipartUploads(S3AFileSystem.java:454)
    at org.apache.hadoop.fs.s3a.S3AFileSystem.initialize(S3AFileSystem.
java:289)
    at org.apache.hadoop.fs.FileSystem.createFileSystem(FileSystem.java:2715)
    at org.apache.hadoop.fs.FileSystem.access$200(FileSystem.java:96)
    at org.apache.hadoop.fs.FileSystem$Cache.getInternal(FileSystem.java:2749)
    at org.apache.hadoop.fs.FileSystem$Cache.getUnique(FileSystem.java:2737)
    at org.apache.hadoop.fs.FileSystem.newInstance(FileSystem.java:430)
```

To resolve the issue, use the specific endpoint of the bucket's S3 service. Using the explicit endpoint for the region is recommended for speed and the ability to use the V4 signing API.

If not using "V4" authentication, you can use the original S3 endpoint:

```
<property>
  <name>fs.s3a.endpoint</name>
  <value>s3.amazonaws.com</value>
</property>
```

3.8.4. Errors During Delete or Rename of Files

The `MultiObjectDeleteException` error may occur when deleting or renaming files:

```
Exception in thread "main" com.amazonaws.services.s3.model.
MultiObjectDeleteException:
    Status Code: 0, AWS Service: null, AWS Request ID: null, AWS Error Code:
 null,
    AWS Error Message: One or more objects could not be deleted, S3 Extended
 Request ID: null
  at com.amazonaws.services.s3.AmazonS3Client.deleteObjects(AmazonS3Client.
java:1745)
```

This error happens when you are trying to delete multiple objects but one of the objects cannot be deleted. Typically, this is due to the fact that the caller lacks the permission to delete these objects. This error should not occur just because the objects are missing.

To resolve the issue, consult the log to see the specifics of which objects could not be deleted. Determine whether or not you have the permission to do so.

If the operation is failing for reasons other than the caller lacking permissions:

1. Try setting `fs.s3a.multiobjectdelete.enable` to "false".

2. Consult HADOOP-11572 for up-to-date advice.

3.8.5. Errors Related to Visible S3A Inconsistency

S3 is *an eventually consistent object store*. That is, it is not a filesystem. It offers read-after-create consistency, which means that a newly created file is immediately visible. Except, there is a small quirk: a negative GET may be cached, such that even if an object is immediately created, the fact that there "wasn't" an object is still remembered.

That means the following sequence on its own will be consistent:

```
touch(path) -> getFileStatus(path)
```

But this sequence may be inconsistent:

```
getFileStatus(path) -> touch(path) -> getFileStatus(path)
```

A common source of visible inconsistencies is that the S3 metadata database — the part of S3 which serves list requests — is updated asynchronously. Newly added or deleted files may not be visible in the index, even though direct operations on the object (HEAD and GET) succeed.

In S3A, that means that the `getFileStatus()` and `open()` operations are more likely to be consistent with the state of the object store than any directory list operations (`listStatus()`, `listFiles()`, `listLocatedStatus()`, `listStatusIterator()`).

The following errors may be related to eventual consistency of S3.

`FileNotFoundException`**, Even Though the File Was Just Written**

This can be a sign of consistency problems. It may also surface if there is some asynchronous file write operation still in progress in the client: the operation has returned, but the write has not yet completed. While the S3A client code does block during the `close()` operation, we suspect that asynchronous writes may be taking place somewhere in the stack; This could explain why parallel tests fail more often than serialized tests.

File Not Found in a Directory Listing, Even Though `getFileStatus()` **Finds It**

File was not found in a directory listing, even though `getFileStatus()` finds it — or a deleted file is found in listing, even though `getFileStatus()` reports that it is not there.

This is a visible sign of updates to the metadata server, which is lagging behind the state of the underlying filesystem.

File Not Visible/Saved

The files in an object store are not visible until the write has been completed. In-progress writes are simply saved to a local file/cached in RAM and only uploaded at the end of a write operation. If a process terminated unexpectedly, or failed to call the `close()` method on an output stream, the pending data will have been lost.

File `flush()` **and** `hflush()` **Calls Do Not Save Data to S3A**

Again, this is due to the fact that the data is cached locally until the `close()` operation. The S3A filesystem cannot be used as a store of data if it is required that the data is persisted durably after every`flush()/hflush()` call. This includes resilient logging, HBase-style journaling and the like. The standard strategy here is to save to HDFS and then copy to S3.

3.8.6. Troubleshooting S3-SSE

Refer to this section when troubleshooting S3-SSE.

3.8.6.1. AccessDeniedException When Creating Directories and Files

Operations such as creating directories (`mkdir()`/`innerMkdirs()`) or files fail when trying to create a file or directory in an object store where the bucket permission requires encryption of a specific type, and the client is not configured to use this specific encryption mechanism.

To resolve the issue, you must configure the client to use the encryption mechanism that you specified in the bucket permissions.

```
java.nio.file.AccessDeniedException: /test: innerMkdirs on /test: com.
amazonaws.services.s3.model.AmazonS3Exception: Access Denied (Service:
 Amazon S3; StatusCode: 403; ErrorCode: AccessDenied; RequestID:
 398EB3738450B416), S3 Extended Request ID: oOcNg+RvbS5YaJ7EQNXVZnHOF/
7fwwhCzyRCjFF+UKLRi3slkobphLt/M+n4KPw5cljSSt2f6/E= at org.apache.hadoop.fs.
s3a.S3AUtils.translateException(S3AUtils.java:158) at org.apache.hadoop.fs.
s3a.S3AUtils.translateException(S3AUtils.java:101) at org.apache.hadoop.fs.
s3a.S3AFileSystem.mkdirs(S3AFileSystem.java:1528) at org.apache.hadoop.fs.
FileSystem.mkdirs(FileSystem.java:2216) Caused by: com.amazonaws.services.
s3.model.AmazonS3Exception: Access Denied (Service: Amazon S3; StatusCode:
 403; ErrorCode: AccessDenied; RequestID: 398EB3738450B416) at com.amazonaws.
http.AmazonHttpClient$RequestExecutor.handleErrorResponse(AmazonHttpClient.
java:1586) at com.amazonaws.http.AmazonHttpClient$RequestExecutor.
executeOneRequest(AmazonHttpClient.java:1254) at com.amazonaws.http.
AmazonHttpClient$RequestExecutor.executeHelper(AmazonHttpClient.
java:1035) at com.amazonaws.http.AmazonHttpClient$RequestExecutor.
doExecute(AmazonHttpClient.java:747) at com.amazonaws.http.AmazonHttpClient
$RequestExecutor.executeWithTimer(AmazonHttpClient.java:721) at com.
amazonaws.http.AmazonHttpClient$RequestExecutor.execute(AmazonHttpClient.
java:704) at com.amazonaws.http.AmazonHttpClient$RequestExecutor.access
$500(AmazonHttpClient.java:672) at com.amazonaws.http.AmazonHttpClient
$RequestExecutionBuilderImpl.execute(AmazonHttpClient.java:654) at com.
amazonaws.http.AmazonHttpClient.execute(AmazonHttpClient.java:518) at com.
amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4185) at
 com.amazonaws.services.s3.AmazonS3Client.invoke(AmazonS3Client.java:4132)
 at com.amazonaws.services.s3.AmazonS3Client.putObject(AmazonS3Client.
java:1712) at com.amazonaws.services.s3.transfer.internal.UploadCallable.
uploadInOneChunk(UploadCallable.java:133) at com.amazonaws.services.s3.
transfer.internal.UploadCallable.call(UploadCallable.java:125) at com.
amazonaws.services.s3.transfer.internal.UploadMonitor.call(UploadMonitor.
java:139) at com.amazonaws.services.s3.transfer.internal.UploadMonitor.
call(UploadMonitor.java:47) at java.util.concurrent.FutureTask.
run(FutureTask.java:266) at java.util.concurrent.ThreadPoolExecutor.
runWorker(ThreadPoolExecutor.java:1142) at java.util.concurrent.
ThreadPoolExecutor$Worker.run(ThreadPoolExecutor.java:617) at java.lang.
Thread.run(Thread.java:745)
```

3.8.6.2. AES256 Is Enabled but an Encryption Key Was Set in `fs.s3a.server-side-encryption.key`

You will see this error when the encryption mechanism is set to SSE-S3/AES-256 but the configuration also declares an encryption key. The error happens because user-supplied keys are not supported in SSE-S3. Remove the `fs.s3a.server-side-encryption.key` setting or switch to SSE-KMS encryption.

```
testEncryption(org.apache.hadoop.fs.s3a.
ITestS3AEncryptionSSECBlockOutputStream)  Time elapsed: 0.103 sec  <<< ERROR!
```

```
java.io.IOException: AES256 is enabled but an encryption key was set in fs.
s3a.server-side-encryption.key (key length = 44)
    at org.apache.hadoop.fs.s3a.S3AUtils.getEncryptionAlgorithm(S3AUtils.
java:758)
    at org.apache.hadoop.fs.s3a.S3AFileSystem.initialize(S3AFileSystem.
java:260)
    at org.apache.hadoop.fs.FileSystem.createFileSystem(FileSystem.java:3242)
    at org.apache.hadoop.fs.FileSystem.get(FileSystem.java:467)
    at org.apache.hadoop.fs.contract.AbstractBondedFSContract.
init(AbstractBondedFSContract.java:72)
    at org.apache.hadoop.fs.contract.AbstractFSContractTestBase.
setup(AbstractFSContractTestBase.java:177)
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAccessorImpl.
java:62)
    at sun.reflect.DelegatingMethodAccessorImpl.
invoke(DelegatingMethodAccessorImpl.java:43)
    at java.lang.reflect.Method.invoke(Method.java:498)
    at org.junit.runners.model.FrameworkMethod$1.
runReflectiveCall(FrameworkMethod.java:47)
    at org.junit.internal.runners.model.ReflectiveCallable.
run(ReflectiveCallable.java:12)
    at org.junit.runners.model.FrameworkMethod.
invokeExplosively(FrameworkMethod.java:44)
    at org.junit.internal.runners.statements.RunBefores.evaluate(RunBefores.
java:24)
    at org.junit.internal.runners.statements.RunAfters.evaluate(RunAfters.
java:27)
    at org.junit.rules.TestWatcher$1.evaluate(TestWatcher.java:55)
    at org.junit.internal.runners.statements.FailOnTimeout$StatementThread.
run(FailOnTimeout.java:74)
```

3.8.6.3. Unknown Server Side Encryption Algorithm

This error means that the algorithm is unknown or mistyped; here "SSE_C" was used rather than "SSE-C":

```
java.io.IOException: Unknown Server Side Encryption algorithm SSE_C
    at org.apache.hadoop.fs.s3a.S3AEncryptionMethods.
getMethod(S3AEncryptionMethods.java:67)
    at org.apache.hadoop.fs.s3a.S3AUtils.getSSEncryptionAlgorithm(S3AUtils.
java:760)
```

Make sure to enter the correct algorithm name.

4. Getting Started with ADLS

Azure Data Lake Store (ADLS) is a file system designed for use as a hyper-scale repository for big data analytic workloads.

The features of ADLS include:

- Hierarchical filesystem containing folders, which in turn contain data stored as files.
- Provides unlimited storage without imposing any limits on account sizes, file sizes, or the amount of data that can be stored in a data lake.
- Compatible with Hadoop Distributed File System (HDFS).
- Can be accessed by Hadoop application via the WebHDFS-compatible REST APIs or the ADLS connector.
- Uses Azure Active Directory (AAD) for identity and access management.

For more general information on ADLS, refer to Get Started with Azure Data Lake Store Using the Azure Portal in Azure documentation.

Overview of Configuring and Using ADLS with HDP

The following table provides an overview of tasks related to configuring and using HDP with ADLS. Click on the linked topics to get more information about specific tasks.

Task	Description
Meet the prerequisites	To use ADLS storage, you must have: 1. An Azure subscription for Data Lake Store. 2. An ADLS account. For instructions on how to create one, refer to Microsoft Azure documentation.
Configure authentication	In order for Hadoop applications to access data stored in your ADLS account, you must configure authentication with the ADLS account using either a client credential (analogous to a service principal) or a refresh token (associated with a user). We recommend that you use the simpler client credential method.
Configure optional features: • Configuring User and Group Representation [49]	You can optionally configure how user and group information is represented during `getFileStatus()`, `listStatus()`, and `getAclStatus()` calls.
Work with ADLS data: • Referencing ADLS in the URLs [49] • Access data with Hive or Spark • Copy data with DistCp	Once you've configured authentication with your data lake, you can access ADLS data from Hive (via external tables) and Spark, and perform other related tasks such as copying data between HDFS and ADLS when needed.

4.1. Configuring Authentication with ADLS

In order to access data in your data lake store, you must configure authentication with ADLS via Azure Active Directory (Azure AD). Azure uses Azure AD as a multi-tenant cloud-based directory and identity management service. For more information, refer to What is Active Directory.

You can configure authentication with ADLS by using either a client credential (analogous to a service principal) or a refresh token (associated with a user). We recommend that you use the simpler client credential method.

4.1.1. Using Client Credential

To configure authentication with ADLS using the client credential, you must register a new application with Active Directory service and then give your application access to your ADL account. After you've performed these steps, you can configure your `core-site.xml`.

Note

For more detailed instructions including screenshots refer to How to Configure Authentication with ADLS blog post.

Register an Application

If you already have your application registered with Active Directory, simply obtain the parameters listed in step 7 below. If you are starting from scratch, perform all the steps:

1. Log in to the Azure Portal.
2. Navigate to your **Active Directory** and then select **App Registrations**.
3. Create a new web application by clicking on **+New application registration**.
4. Specify an application name, type (Web app/API), and sign-on URLs.

   Remember the application name: you will later add it to your ADLS account as an authorized user.
5. Once an application is created, navigate to the application configuration and find the **Keys** in the application's settings.
6. Create a key by entering key description, selecting a key duration, and then clicking **Save**.

   Make sure to copy and save the key value. You won't be able to retrieve it after you leave the page.
7. Note down the properties that you will need to authenticate:

Parameter	How to obtain it
Application ID	You can find it in your application's settings. This will be your *`fs.adl.oauth2.client.id`*
Key	This is the key that you generated for your application. If you did not copy the it, you must create a new key from the **Keys** page in your application's settings. This will be your *`fs.adl.oauth2.credential`*
Token endpoint	You can obtain this from the **App Registrations**>**Endpoints** page by copying the *`OAUTH 2.0 TOKEN ENDPOINT`* value. This will be your *`fs.adl.oauth2.refresh.url`*

Add the Application to your Data Lake Store Account

If you are planning to use multiple Data Lake Store accounts, perform these steps for each account.

1. Log in to the Azure Portal.
2. If you don't have a **Data Lake Store** account, create one.
3. Navigate to your Data Lake Store account and then select **Access Control (IAM)**.
4. Click on **+Add** to add role-based permissions.
5. Under **Role** select the "Owner". Under **Select**, select your application.

   This will grant the "Owner" role for this ADL account to your application.

Configure core-site.xml

Add the following four properties to your `core-site.xml`. While *`fs.adl.oauth2.access.token.provider.type`* must be set to “ClientCredential”, you can obtain the remaining three parameters from step 7 above.

```
<property>
  <name>fs.adl.oauth2.access.token.provider.type</name>
  <value>ClientCredential</value>
</property>

<property>
  <name>fs.adl.oauth2.client.id</name>
  <value>APPLICATION-ID</value>
</property>

<property>
  <name>fs.adl.oauth2.credential</name>
  <value>KEY</value>
</property>

<property>
  <name>fs.adl.oauth2.refresh.url</name>
  <value>TOKEN-ENDPOINT</value>
</property>
```

Next Steps

To protect these credentials, we recommend that you use the credential provider framework to securely store and access your credentials.

To make sure that authentication works, try referencing ADLS in the URLs.

4.1.2. Using Token-Based Authentication

To use token-based authentication:

1. Obtain a valid OAuth2 bearer token from the Azure Active Directory service for those valid users who have access to Azure Data Lake Storage account. The token must be

obtained for a specific client ID in the application code. For more information, refer to Active Directory Library For Java.

2. Add the following properties to your `core-site.xml`:

```
<property>
  <name>fs.adl.oauth2.access.token.provider.type</name>
  <value>RefreshToken</value>
</property>

<property>
  <name>fs.adl.oauth2.client.id</name>
  <value>CLIENT-ID</value>
</property>

<property>
  <name>fs.adl.oauth2.refresh.token</name>
  <value>REFRESH-TOKEN</value>
</property>
```

Se the value of *`fs.adl.oauth2.access.token.provider.type`* to "RefreshToken" and set the other two parameters.

Note

Do not share the client ID or the refresh token. They must be kept secret.

Next Steps

To make sure that authentication works, try referencing ADLS in the URLs.

4.1.3. Protecting the Azure Credentials for ADLS with Credential Providers

All ADLS credential properties can be protected by credential providers.

To provision the credentials:

```
hadoop credential create fs.adl.oauth2.client.id -value 123
    -provider localjceks://file/home/foo/adls.jceks
hadoop credential create fs.adl.oauth2.refresh.token -value 123
    -provider localjceks://file/home/foo/adls.jceks
```

Next, configure the following configuration properties, either on the command line or in the `core-site.xml` configuration file:

```
<property>
  <name>fs.adl.oauth2.access.token.provider.type</name>
  <value>RefreshToken</value>
</property>
<property>
  <name>hadoop.security.credential.provider.path</name>
  <value>localjceks://file/home/foo/adls.jceks</value>
</property>
```

The *`hadoop.security.credential.provider.path`* should indicate the path to interrogate for protected credentials.

You may optionally add the provider path property to the `distcp` command line instead of adding a job-specific configuration to a generic `core-site.xml`. The options enclosed in square brackets illustrate this capability.

```
hadoop distcp
    [-D fs.adl.oauth2.access.token.provider.type=RefreshToken
     -D hadoop.security.credential.provider.path=localjceks://file/home/user/
adls.jceks]
    hdfs://<NameNode Hostname>:9001/user/foo/srcDir
    adl://<Account Name>.azuredatalakestore.net/tgtDir/
```

Related Links

Credential Provider API

4.2. Referencing ADLS in the URLs

Regardless of which specific Hadoop ecosystem application you are using, you can access data stored in ADLS using the URI starting with the *adl://* prefix.

The URL structure is:

```
adl://<data_lake_store_name>.azuredatalakestore.net/dir/file
```

For example, to access "testfile" located in a directory called "testdir", stored in a data lake store called "mytest", the URL is:

```
adl://mytest.azuredatalakestore.net/testdir/testfile
```

The following FileSystem shell commands demonstrate access to a data lake store named `mytest`:

```
hadoop fs -ls adl://mytest.azuredatalakestore.net/

hadoop fs -mkdir adl://mytest.azuredatalakestore.net/testDir

hadoop fs -put testFile adl://mytest.azuredatalakestore.net/testDir/testFile

hadoop fs -cat adl://mytest.azuredatalakestore.net/testDir/testFile
test file content
```

4.3. Configuring User and Group Representation

HDP allows you to configure how user and group information is represented during `getFileStatus()`, `listStatus()`, and `getAclStatus()` calls. You can configure this by adding the following property to `core-site.xml`:

```
<property>
  <name>adl.feature.ownerandgroup.enableupn</name>
  <value>true</value>
</property>
```

When set to "true", user and group in the FileStatus/AclStatus response is represented as a user-friendly name as per Azure AD profile. When set to "false" (default), user and group in the FileStatus/AclStatus response is represented by the unique identifier from Azure AD profile (Object ID as GUID).

For best performance we recommended using the default value.

5. Getting Started with WASB

Windows Azure Storage Blob (WASB) is a general-purpose object store.

The features of WASB include:

- Object store with flat namespace.
- Storage account consists of containers, which in turn have data in the form of blobs.
- Authentication based on shared secrets - Account Access Keys (for account-level authorization) and Shared Access Signature Keys (for account, container, or blob authorization).

Overview of Configuring and Using WASB with HDP

The following table provides an overview of tasks related to configuring and using HDP with WASB. Click on the linked topics to get more information about specific tasks.

Task	Description
Meet the prerequisites	To use Azure Blob Storage, you must have: 1. An Azure subscription for Storage Blobs. 2. A storage account. For instructions on how to create one, refer to Microsoft Azure documentation.
Configure authentication	In order to access data in an WASB account, you must configure authentication with the WASB account by providing the access key.
Configure optional features: • Configuring Page Blob Support [54] • Configuring Atomic Folder Rename [54] • Configuring Support for Append API [55] • Configuring Multithread Support [55] • Configuring WASB Secure Mode [56] • Configuring Authorization Support in WASB [57]	You can optionally configure these additional features.
Work with WASB data: • Referencing WASB in the URLs [53] • Access data with Hive or Spark • Copy data with DistCp	Once you've configured authentication with your blob storage account, you can access data stored in this account from Hive (via external tables) and Spark, and perform other related tasks such as copying data between HDFS and WASB when needed.

5.1. Configuring Authentication with WASB

In order to access data stored in your Azure blob storage account, you must configure your storage account access key in `core-site.xml`. The configuration property that you must use is `fs.azure.account.key.<account name>.blob.core.windows.net` and the value is the access key.

For example the following property should be used for a storage account called "testaccount":

```
<property>
  <name>fs.azure.account.key.testaccount.blob.core.windows.net</name>
  <value>TESTACCOUNT-ACCESS-KEY</value>
</property>
```

You can obtain your access key from the **Access keys** in your storage account settings.

Note

For more detailed instructions including screenshots refer to How to Configure Authentication with WASB blog post.

Next Steps

To protect your access key, we recommend that you use the credential provider framework to securely store and access your credentials. You may also protect the Azure credentials within an encrypted file.

To make sure that authentication works, try referencing WASB in the URLs.

5.1.1. Protecting the Azure Credentials for WASB with Credential Providers

To protect your credentials from unauthorized users, we recommend that you use the credential provider framework which securely stores your credentials and allows you to securely access them.

To provision the credentials:

```
% hadoop credential create fs.azure.account.key.youraccount.blob.core.windows.
net -value 123
    -provider localjceks://file/home/lmccay/wasb.jceks
```

Next, configure the following configuration properties, either on the command line or in the `core-site.xml` configuration file:

```
<property>
  <name>hadoop.security.credential.provider.path</name>
  <value>localjceks://file/home/lmccay/wasb.jceks</value>
  <description>Path to interrogate for protected credentials.</description>
</property>
```

You may optionally add the provider path property to the `distcp` command line instead of adding a job-specific configuration to a generic `core-site.xml`. The options enclosed in square brackets illustrate this capability.

```
% hadoop distcp
    [-D hadoop.security.credential.provider.path=localjceks://file/home/
lmccay/wasb.jceks]
    hdfs://hostname:9001/user/lmccay/007020615 wasb://
yourcontainer@youraccount.blob.core.windows.net/testDir/
```

You may also protect the Azure credentials within an encrypted file.

Related Links

Credential Provider API

5.1.1.1. Protecting the Azure Credentials for WASB within an Encrypted File

In addition to using the credential provider framework to protect your credentials, it is also possible to configure it in an encrypted form. An additional configuration property `fs.azure.shellkeyprovider.script` specifies an external program to be invoked by Hadoop processes to decrypt the key. The encrypted key value is passed to this external program as a command line argument:

```
<property>
  <name>fs.azure.account.keyprovider.youraccount</name>
  <value>org.apache.hadoop.fs.azure.ShellDecryptionKeyProvider</value>
</property>

<property>
  <name>fs.azure.account.key.youraccount.blob.core.windows.net</name>
  <value>YOUR ENCRYPTED ACCESS KEY</value>
</property>

<property>
  <name>fs.azure.shellkeyprovider.script</name>
  <value>PATH TO DECRYPTION PROGRAM</value>
</property>
```

5.2. Referencing WASB in the URLs

Regardless of which specific Hadoop ecosystem application you are using, you can access data stored in WASB using the URL starting with the *wasb://* prefix.

The URL structure is:

```
wasb://<container_name>@<storage_account_name>.blob.core.windows.net/dir/file
```

For example, to access a file called "testfile" located in a directory called "testdir", stored in the container called "testcontainer" on the account called "hortonworks", the URL is:

```
wasb://testcontainer@hortonworks.blob.core.windows.net/testdir/testfile
```

You can also use *wasbs* prefix to utilize SSL-encrypted HTTPS access:

```
wasbs://<container_name>@<storage_account_name>.blob.core.windows.net/dir/file
```

For example, the following Hadoop FileSystem shell commands demonstrate access to a storage account named `myaccount` and a container named `mycontainer`:

```
hadoop fs -ls wasb://mycontainer@myaccount.blob.core.windows.net/

hadoop fs -mkdir wasb://mycontainer@myaccount.blob.core.windows.net/testDir

hadoop fs -put testFile wasb://mycontainer@myaccount.blob.core.windows.net/
testDir/testFile

hadoop fs -cat wasb://mycontainer@myaccount.blob.core.windows.net/testDir/
testFile
test file content
```

It is also possible to configure `fs.defaultFS` to use a `wasb` or `wasbs` URL. This causes all bare paths, such as `/testDir/testFile` to resolve automatically to that file system.

5.3. Configuring Page Blob Support

The Azure Blob Storage interface for Hadoop supports two kinds of blobs, block blobs and page blobs.

Block blobs, which are used by default, are suitable for most big-data use cases such as input data for Hive, Pig, analytical map-reduce jobs, and so on.

Page blobs can be up to 1TB in size, larger than the maximum 200GB size for block blobs. Their primary use case is in the context of HBase write-ahead logs. This is because page blobs can be written any number of times, whereas block blobs can only be appended up to 50,000 times, at which point you run out of blocks and your writes fail. This wouldn't work for HBase logs, so page blob support was introduced to overcome this limitation.

1. In order to have the files that you create be page blobs, you must set the configuration variable `fs.azure.page.blob.dir` in `core-site.xml` to a comma-separated list of folder names. For example:

   ```
   <property>
     <name>fs.azure.page.blob.dir</name>
     <value>/hbase/WALs,/hbase/oldWALs,/data/mypageblobfiles</value>
   </property>
   ```

   To make all files page blobs, you can simply set this to `/`.

2. You can set two additional configuration properties related to page blobs. You can also set them in `core-site.xml`:

- The configuration option `fs.azure.page.blob.size` defines the default initial size for a page blob. The parameter value is an integer specifying the number of bytes. It must be 128MB or greater, but no more than 1TB.

- The configuration option `fs.azure.page.blob.extension.size` defines the page blob extension size. This determines the amount by which to extend a page blob when it becomes full. The parameter value is an integer specifying the number of bytes. It must be 128MB or greater, specified as an integer number of bytes.

5.4. Configuring Atomic Folder Rename

The Azure Blob Storage stores files in a flat key/value store without formal support for folders. The hadoop-azure file system layer simulates folders on top of Azure storage. By default, folder rename in the hadoop-azure file system layer is not atomic. This means that a failure during a folder rename could potentially leave some folders in the original directory and some in the new one.

Since HBase depends on atomic folder rename, a configuration setting called `fs.azure.atomic.rename.dir` can be set in `core-site.xml` to specify a comma-separated list of directories where folder rename is made atomic. If a folder rename fails, a redo will be applied to finish. A file `<folderName>-renamePending.json` may appear

temporarily and is the record of the intention of the rename operation, to allow redo in event of a failure.

The default value of this setting is just `/hbase`. To list multiple directories, separate them with a comma. For example:

```
<property>
  <name>fs.azure.atomic.rename.dir</name>
  <value>/hbase,/data</value>
</property>
```

5.5. Configuring Support for Append API

The Azure Blob Storage interface for Hadoop includes optional support for Append API for single writer.

To enable it, set the configuration `fs.azure.enable.append.support` to "true" in `core-site.xml`:

```
<property>
  <name>fs.azure.enable.append.support</name>
  <value>true</value>
</property>
```

Note

Append support in Azure Blob Storage interface differs from HDFS semantics: it does not enforce single writer internally but requires applications to guarantee this semantic. It becomes a responsibility of the application either to ensure single-threaded handling for a particular file path, or to rely on some external locking mechanism of its own. Failure to do so will result in an unexpected behavior.

5.6. Configuring Multithread Support

Rename and delete blob operations on directories with a large number of files and sub directories are currently very slow as these operations are done serially, one blob at a time. These files and sub-folders can be deleted or renamed parallel. The following configurations can be used to enable threads to do parallel processing:

Delete

To enable 10 threads for delete operation, set the following configuration value in `core-site.xml`:

```
<property>
  <name>fs.azure.delete.threads</name>
  <value>10</value>
</property>
```

To disable threads, set it to 0 or 1. The default behavior is threads disabled.

Rename

To enable 20 threads for "rename" operation, set the following configuration value in `core-site.xml`:

```
<property>
  <name>fs.azure.rename.threads</name>
  <value>20</value>
</property>
```

To disable threads, set it to 0 or 1. The default behavior is threads disabled.

5.7. Configuring WASB Secure Mode

WASB can operate in secure mode, where the storage access keys required to communicate with Azure storage do not have to be in the same address space as the process using WASB. In this mode, all interactions with Azure storage are performed using SAS URIs. There are two sub-modes within the secure mode:

- (Option 1) The remote SAS key mode, where the SAS keys are generated from a remote process
- (Option 2) The local mode, where SAS keys are generated within WASB.

By default, the SAS key mode is expected to run in the remote mode; however, for testing purposes the local mode can be enabled to generate SAS keys in the same process as WASB.

To enable the secure mode, set the following property in `core-site.xml`:

```
<property>
  <name>fs.azure.secure.mode</name>
  <value>true</value>
</property>
```

Next, do one of the following, depending on the sub-mode that you are using:

To enable SAS key generation locally (Option 1), set the following property in `core-site.xml`:

```
<property>
  <name>fs.azure.local.sas.key.mode</name>
  <value>true</value>
</property>
```

To use the remote SAS key generation mode (Option 2), an external REST service is expected to provided required SAS keys. The following property can set in `core-site.xml` to provide the end point to use for remote SAS key generation:

```
<property>
  <name>fs.azure.cred.service.url</name>
  <value>{URL}</value>
 </property>
```

The remote service is expected to provide support for two REST calls `{URL}/GET_CONTAINER_SAS` and `{URL}/GET_RELATIVE_BLOB_SAS`, for generating container and relative blob SAS keys.

Example requests:

```
{URL}/GET_CONTAINER_SAS?
storage_account=<account_name>&container=<container>&sas_expiry=<expiry
period>&delegation_token=<delegation token> {URL}/
GET_CONTAINER_SAS?
storage_account=<account_name>&container=<container>&relative_path=<relative
path>&sas_expiry=<expiry period>&delegation_token=<delegation
token>
```

The service is expected to return a response in JSON format:

```
{
  "responseCode" : 0 or non-zero <int>,
  "responseMessage" : relavant message on failure <String>,
  "sasKey" : Requested SAS Key <String>
}
```

5.8. Configuring Authorization Support in WASB

To enable authorization support in WASB, set the following property in `core-site.xml`:

```
<property>
   <name>fs.azure.authorization</name>
   <value>true</value>
</property>
```

The current implementation of authorization relies on the presence of an external service that can enforce the authorization. The service is expected to be running on a URL provided by the following configuration, which should also be set in `core-site.xml`:

```
<property>
   <name>fs.azure.authorization.remote.service.url</name>
   <value>{URL}</value>
</property>
```

The remote service is expected to provide support for the following REST call: `{URL}/CHECK_AUTHORIZATION`

An example request: `{URL}/CHECK_AUTHORIZATION?wasb_absolute_path=<absolute_path>&operation_type=<operation type>&delegation_token=<delegation token>`

The service is expected to return a response in JSON format:

```
{
  "responseCode" : 0 or non-zero <int>,
  "responseMessage" : relevant message on failure <String>,
  "authorizationResult" : true/false <boolean>
}
```

6. Accessing Cloud Data in Hive

Datasets stored in S3, ADLS or WASB can be made available in Hive via external tables:

1. Exposing Cloud Data as Hive Tables [58]
2. Populating Partition-Related Information [59]
3. Analyzing Tables [59]

If you are using Ranger to manage Hive authorization, refer to Create a Hive Policy in the Security Guide to learn how to create Hive policies that include S3 URLs.

To improve performance for Hive with S3, ADLS and WASB, refer to Improving Hive Performance.

6.1. Exposing Cloud Data as Hive Tables

Datasets stored in S3, ADLS or WASB can be easily made available in Hive as managed or external tables. The main difference between these two table types is that data linked in an external table does not get deleted when the table is deleted.

Therefore, external tables are optimal when the data is already present in a cloud storage service such as S3, ADLS or WASB, which provides longer-term persistence at a lower cost than attached storage.

External Tables

External tables operate in a similar manner as references. If you create an external table called "inventory"

```
CREATE EXTERNAL TABLE `inventory`(
  `inv_item_sk` int,
  `inv_warehouse_sk` int,
  `inv_quantity_on_hand` int)
PARTITIONED BY (
  `inv_date_sk` int) STORED AS ORC
LOCATION
  's3a://BUCKET_NAME/tpcds_bin_partitioned_orc_200.db/inventory';
```

and then you drop the "inventory" table, the contents of `s3a://BUCKET_NAME/tpcds_bin_partitioned_orc_200.db/inventory` will not be deleted. Only the table definition will be removed from the metastore.

Note that data from a partitioned table is not automatically loaded upon table creation. To load data, use the MSCK. For more information, refer to Populating Partition-Related Information.

Managed Tables

Note

The following actions require you to have write access to the S3 bucket.

If you create a managed table called “inventory”

```
CREATE TABLE `inventory`(
  `inv_item_sk` int,
  `inv_warehouse_sk` int,
  `inv_quantity_on_hand` int)
PARTITIONED BY (
  `inv_date_sk` int) STORED AS ORC
LOCATION
  's3a://BUCKET_NAME/tpcds_bin_partitioned_orc_200.db/inventory';
```

and then you drop the “inventory” table, the contents of `s3a://BUCKET_NAME/tpcds_bin_partitioned_orc_200.db/inventory` will be deleted.

6.2. Populating Partition-Related Information

When working with data stored in S3, ADLS or WASB, the steps for populating partition-related information are the same as when working with data in HDFS.

Creating table definitions does not by itself auto-populate partition-related information to the metastore. When a dataset available in Amazon S3 is already partitioned, you must run the MSCK command in order to populate the partition-related information into the metastore.

For example, consider the following statement:

```
CREATE EXTERNAL TABLE `inventory`(
  `inv_item_sk` int,
  `inv_warehouse_sk` int,
  `inv_quantity_on_hand` int)
PARTITIONED BY (
  `inv_date_sk` int) STORED AS ORC
LOCATION
  's3a://BUCKET_NAME/tpcds_bin_partitioned_orc_200.db/inventory';
```

This statement creates a table definition in the metastore, but does not populate the partition-related information.

To populate the partition-related information, you need to run `MSCK REPAIR TABLE inventory`.

You can increase the value of the `hive.metastore.fshandler.threads` parameter to increase the number of threads used for scanning the partitions in the MSCK phase (defaut is 15). This will speed up load if you have hardware capacity.

6.3. Analyzing Tables

When working with data in S3, ADLS or WASB, the steps for analyzing tables are the same as when working with data in HDFS.

Table statistics can be gathered automatically by setting `hive.stats.autogather=true` or by running `analyze table test compute statistics` command. For example:

```
ANALYZE TABLE Table1 PARTITION(ds='2008-04-09', hr=11) COMPUTE STATISTICS;
```

Column statistics are not automatically created. You must manually gather column statistics by running `analyze table test compute statistics for columns` command. For example:

```
ANALYZE TABLE Table1 PARTITION(ds='2008-04-09', hr=11) COMPUTE STATISTICS FOR
 COLUMNS;
```

6.4. Improving Hive Performance with S3/ADLS/ WASB

Tune the following parameters to improve Hive performance when working with S3, ADLS or WASB.

Table 6.1. Improving General Performance

Parameter	Recommended Setting
yarn.scheduler.capacity.node-locality-delay	Set this to "0".
hive.warehouse.subdir.inherit.perms	Set this to "false" to reduce the number of file permission checks.
hive.metastore.pre.event.listeners	Set this to an empty value to reduce the number of directory permission checks.

You can set these parameters in `hive-site.xml`.

Table 6.2. Accelerating ORC Reads in Hive

Parameter	Recommended Setting
hive.orc.compute.splits.num.threads	If using ORC format and you want improve the split computation time, you can set the value of this parameter to match the number of available processors. By default, this parameter is set to 10. This parameter controls the number of parallel threads involved in computing splits. For Parquet computing splits is still single-threaded, so split computations can take longer with Parquet and S3/ADLS/WASB.
hive.orc.splits.include.file.footer	If using ORC format with ETL file split strategy, you can set this parameter to "true" in order to use existing file footer information in split payload.

You can set these parameters using `--hiveconf` option in Hive CLI or using the `set` command in Beeline.

Table 6.3. Accelerating ETL Jobs

Parameter	Recommended Setting
hive.metastore.fshandler.threads	Query launches can be slightly slower if there are no stats available or when `hive.stats.fetch.partition.stats=false`. In such cases, Hive ends up looking at file sizes for every file that it tries to access. Tuning `hive.metastore.fshandler.threads` helps reduce the overall time taken for the metastore operation.

Parameter	Recommended Setting
`fs.trash.interval`	Drop table can be slow in object stores such as S3 because the action involves moving files to trash (a copy + delete). To remedy this, you can set `fs.trash.interval=0` to completely skip trash.

You can set these parameters using `--hiveconf` option in Hive CLI or using the `set` command in Beeline.

Accelerating Inserts in Hive

When inserting data, Hive moves data from a temporary folder to the final location. This move operation is actually a copy+delete action, which is expensive in object stores such as S3; the more data is being written out to the object store, the more expensive the operation is.

To accelerate the process, you can tune `hive.mv.files.thread`, depending on the size of your dataset (default is 15). You can set it in `hive-site.xml`.

7. Accessing Cloud Data in Spark

Datasets stored in S3, ADLS or WASB can be made available in Spark.

S3, ADLS or WASB are viewed by Spark as filesystems, allowing them to be used as the source and destination of data of data: be it batch, SQL, DataFrame, or Spark Streaming. To load and save data in the cloud, Spark uses the same APIs that is used to load and save data in HDFS or other filesystems.

Provided the relevant libraries are on the classpath, a file stored in S3, ADLS or WASB can be referenced simply via a URL:

```
sparkContext.textFile("s3a://landsat-pds/scene_list.gz").count()
```

Similarly, an RDD can be saved to an object store via `saveAsTextFile()`:

```
val numbers = sparkContext.parallelize(1 to 1000)
// save to Amazon S3 (or compatible implementation)
numbers.saveAsTextFile("s3a://bucket1/counts")
```

Example 1: DataFrames

DataFrames can read from and write to object stores using their `read()` and `write()` methods:

```
import org.apache.spark.SparkConfimport org.apache.spark.sql.
SparkSessionimport org.apache.spark.sql.types.StringTypeval spark =
SparkSession
    .builder
    .appName("DataFrames")
    .config(sparkConf)
    .getOrCreate()
import spark.implicits._
val numRows =1000// generate test dataval sourceData = spark.range(0,
 numRows).select($"id".as("l"), $"id".cast(StringType).as("s"))

// define the destinationval dest ="s3a://bucket1/dataframes"// write the
 dataval orcFile = dest + "/data.orc"
sourceData.write.format("orc").save(orcFile)

// now read it backval orcData = spark.read.format("orc").load(orcFile)

// finally, write the data as Parquet
orcData.write.format("parquet").save(dest + "/data.parquet")

spark.stop()
```

Note

Checkpointing streaming data to an S3 bucket is very slow, as the stream data is (potentially) recalculated, uploaded to S3, and then renamed into the checkpoint file (the rename being a slow copy operation). If S3 is used for checkpointing, the interval between checkpoints must be long enough to allow for this slow checkpoint.

Example 2: Spark Streaming and Cloud Storage

Spark Streaming can monitor files added to object stores by creating a `FileInputDStream` DStream monitoring path under a bucket:

```
import org.apache.spark.SparkConfimport org.apache.spark.sql.
SparkSessionimport org.apache.spark.streaming._

val sparkConf =newSparkConf()
val ssc =newStreamingContext(sparkConf, Milliseconds(5000))
try {
  val lines = ssc.textFileStream("s3a://bucket1/incoming")
  val matches = lines.filter(_.endsWith("3"))
  matches.print()
  ssc.start()
  ssc.awaitTermination()
} finally {
  ssc.stop(true)
}
```

Note

The time to scan for new files is proportional to the number of files under the path — not the number of *new* files — so this can become a slow operation.

Related Links

Committing Output to S3 [63]

Improving Spark Performance with S3/ADLS/WASB [63]

7.1. Committing Output to S3

For the reasons covered in Limitations of Amazon S3, using S3 as the direct destination of work may be slow and unreliable in the presence of failures. Therefore, we recommend that you use HDFS as the destination of work, using DistCp to copy to S3 afterwards if you wish to persist beyond the life of the cluster. HDFS has the behaviors critical to the output committer used by Spark and Hadoop MapReduce to ensure the output is correctly generated (atomic directory renames and consistent directory listings).

7.2. Improving Spark Performance with S3/ADLS/WASB

Use the following recommendations to improve Spark performance with cloud data:

- Accelerating ORC and Parquet Reads [63]
- Accelerating Sequential Reads Through Files in S3 [64]

7.2.1. Accelerating ORC and Parquet Reads

Use Random Read Policy

When reading binary ORC and Parquet datasets, you should configure Spark to use the S3A's `random` IO read policy, as described in Optimizing HTTP GET Requests for S3. With

`fs.s3a.experimental.input.fadvise` set to `random`, rather than ask for a the entire file in one HTTPS request (the "normal" operation), the S3A connector only asks for part of a file at a time. If it needs to seek backwards, the remaining data in this part is discarded, and then a new request is made on the same HTTPS connection. This reduces the time wasted on closing and opening up new HTTPS connections.

This setting dramatically speeds up random access, but actually reduces performance on queries performing sequential reads through an entire file — so do not use `random` setting for such jobs.

Minimize Read and Write Operations for ORC

For optimal performance when reading files saved in the ORC format, read and write operations must be minimized. To achieve this, set the following options:

```
spark.sql.orc.filterPushdown true
spark.sql.hive.metastorePartitionPruning true
```

The `spark.sql.orc.filterPushdown` option enables the ORC library to skip unneeded columns and to use index information to filter out parts of the file where it can be determined that no columns match the predicate.

With the `spark.sql.hive.metastorePartitionPruning` option enabled, predicates are pushed down into the Hive metastore to eliminate unmatched partitions.

Minimize Read and Write Operations for Parquet

For optimal performance when reading files saved in the Parquet format, read and write operations must be minimized, including generation of summary metadata, and coalescing metadata from multiple files. The predicate pushdown option enables the Parquet library to skip unneeded columns, saving bandwidth. To achieve this, set the following options:

```
spark.hadoop.parquet.enable.summary-metadata false
spark.sql.parquet.mergeSchema false
spark.sql.parquet.filterPushdown true
spark.sql.hive.metastorePartitionPruning true
```

7.2.2. Accelerating Sequential Reads Through Files in S3

Note

This optimization is meant specifically for Amazon S3.

The most effective way to scan a large file is in a single HTTPS request - which is the default behavior. If the scanning code skips parts of the file using `seek()`, then you can potentially improve the performance of these forward seeks by tuning the option `spark.hadoop.fs.s3a.readahead.range`. For example:

```
spark.hadoop.fs.s3a.readahead.range 512M
```

This option declares the number of bytes to read when seeking forwards in a file before closing and re-opening the HTTPS connection to S3. That close/reopen operation can be so slow that simply reading and discarding the data is actually faster. This is particularly true when working with remote S3 buckets of "long-haul" connections.

8. Copying Cloud Data with Hadoop

To copy and manage datasets stored in S3, ADLS or WASB between the cloud storage and HDFS, you can use DistCp and FS Shell commands.

8.1. Copying Data with DistCp

You can use DistCp to copy data between your cluster's HDFS and your cloud storage. DistCp is a utility for copying large data sets between distributed filesystems. To access DistCp utility, SSH to any node in your cluster.

Copying Data from HDFS to Cloud Storage

To transfer data from HDFS to an Amazon S3 bucket, list the path to HDFS first and the path to the cloud storage second:

```
hadoop distcp hdfs://source-folder s3a://destination-bucket
```

Updating Existing Data

If you would like to transfer only the files that don't already exist in the target folder, add the `update` option to improve the copy speeds:

```
hadoop distcp -update hdfs://source-folder s3a://destination-bucket
```

When copying between Amazon S3 and HDFS, the "update" check only compares file size; it does not use checksums to detect other changes in the data.

Copying Data from Cloud Storage to HDFS

To copy data from your cloud storage container to HDFS, list the path of the cloud storage data first and the path to HDFS second. For example:

```
hadoop distcp s3a://hwdev-examples-ireland/datasets /tmp/datasets2
```

This downloads all files.

You can add the `update` option to only download data which has changed:

```
hadoop distcp -update s3a://hwdev-examples-ireland/datasets /tmp/datasets2
```

Copying Data Between Cloud Storage Containers

You can copy data between cloud storage containers simply by listing the different URLs as the source and destination paths. This includes copying:

- Between two Amazon S3 buckets
- Between two ADLS containers
- Between two WASB containers
- Between ADLS and WASB containers

For example, to copy data from one Amazon S3 bucket to another, use the following syntax:

```
hadoop distcp s3a://hwdev-example-ireland/datasets s3a://hwdev-example-us/
datasets
```

Irrespective of source and destination bucket locations, when copying data between Amazon S3 buckets, all data passes through the Hadoop cluster: once to read, once to write. This means that the time to perform the copy depends on the size of the Hadoop cluster, and the bandwidth between it and the S3 buckets. Furthermore, even when running within Amazon's own infrastructure, you are billed for your accesses to remote Amazon S3 buckets.

Copying Data Within a Cloud Storage Container

Copy operations within a single object store still take place in the Hadoop cluster, even when the object store implements a more efficient copy operation internally. That is, an operation such as

```
hadoop distcp s3a://bucket/datasets/set1 s3a://bucket/datasets/set2
```

copies each byte down to the Hadoop worker nodes and back to the bucket. In addition to the operation being slow, it means that charges may be incurred.

Specifying Per-Bucket DistCp Options for S3

If a bucket has different authentication or endpoint options, then the different options for that bucket can be set with a bucket-specific option. For example, to copy to a remote bucket using Amazon's V4 authentication API requires the explicit S3 endpoint to be declared:

```
hadoop distcp s3a://hwdev-example-us/datasets/set1 s3a://hwdev-example-
frankfurt/datasets/ \
  -D fs.s3a.bucket.hwdev-example-frankfurt.endpoint=s3.eu-central-1.amazonaws.
com
```

Similarly, different credentials may be used when copying between buckets of different accounts. When performing such an operation, consider that secrets on the command line can be visible to other users on the system, so potentially insecure.

```
hadoop distcp s3a://hwdev-example-us/datasets/set1 s3a://hwdev-example-
frankfurt/datasets/ \
  -D fs.s3a.bucket.hwdev-example-frankfurt.endpoint=s3.eu-central-1.amazonaws.
com \
  -D fs.s3a.fs.s3a.bucket.hwdev-example-frankfurt.access.key=AKAACCESSKEY-2 \
  -D fs.s3a.bucket.nightly.secret.key=SECRETKEY
```

Using short-lived session keys can reduce the vulnerabilities here, while storing the secrets in hadoop jceks credential files is potentially significantly more secure.

Related Links

Improving Performance for DistCp [67]

Local Space Requirements for Copying to S3 [67]

Limitations When Using DistCp with S3 [67]

Apache DistCp documentation

8.1.1. Improving Performance for DistCp

ADLS and WASB

You can tune *fs.azure.selfthrottling.read.factor* and *fs.azure.selfthrottling.write.factor*. Refer to Maximizing HDInsight throughput to Azure Blob Storage blog post.

Amazon S3

If you are planning to copy large amounts of data between HDFS and S3, you can accelerate the process by passing `-D fs.s3a.fast.upload=true` while invoking DistCp. For example:

```
hadoop distcp -D fs.s3a.fast.upload=true  s3a://dominika-test/driver-data /
tmp/test2
```

The `fs.s3a.fast.upload` option significantly accelerates data upload by writing the data in blocks, possibly in parallel.

For more tips on how to improve performance for DistCp with S3, refer to Configuring and Tuning S3A Fast Upload.

8.1.2. Local Space Requirements for Copying to S3

When copying files to S3 using the S3A connector, DistCp copies each file to the local temp directory before the final upload, so you need as much space on your disk as your largest file. The location of this intermediate directory is set in the property *fs.s3a.buffer.dir*; if needed, you can change that to a location where you have more space.

When working with S3, you reduce the amount of disk space needed by switching to the S3A fast upload mechanism, which only needs enough disk space to store blocks of data which have not yet been uploaded, or even do it in memory. You can limit the requirements even further by reducing the thread pool size.

8.1.3. Limitations When Using DistCp with S3

When using DistCp with data in S3, consider the following limitations:

- The `-append` option is not supported.
- The `-diff` option is not supported.
- The `-atomic` option causes a rename of the temporary data, so significantly increases the time to commit work at the end of the operation. Furthermore, as S3A does not offer atomic renames of directories, the `-atomic operation doesn't actually deliver what is promised.` *Avoid using this option.*
- All `-p` options, including those to preserve permissions, user and group information, attributes checksums, and replication are ignored.
- CRC checking will not be performed, irrespective of the value of the `-skipCrc` flag.

8.2. Running FS Shell Commands

Many of the standard Hadoop FileSystem shell commands that interact with HDFS also can be used to interact with S3, ADLS, and WASB. They can be useful for a few specific purposes including confirming that the authentication with your cloud service works, debugging, browsing files and creating directories (as an alternative to the cloud service-specific tools), and other management operations.

When running the commands, provide a fully qualified URL. The commands use the following syntax

```
hadoop fs -<operation> URL
```

where <operation> indicates a particular action to be performed against a directory or a file.

For example, the following command lists all files in a directory called "dir1", which resides in an Amazon S3 bucket called "bucket1":

```
hadoop fs -ls s3a://bucket1/dir1
```

Examples

Create directories and create or copy files into them:

```
# Create a directory
hadoop fs -mkdir s3a://bucket1/datasets/

# Upload a file from the cluster filesystem
hadoop fs -put /datasets/example.orc s3a://bucket1/datasets/

# Touch a file
hadoop fs -touchz s3a://bucket1/datasetstouched
```

Download and view objects:

```
# Copy a directory to the local filesystem
hadoop fs -copyToLocal s3a://bucket1/datasets/

# Copy a file from the object store to the local filesystem
hadoop fs -get s3a://bucket1/hello.txt /examples

# Print the object
hadoop fs -cat s3a://bucket1/hello.txt

# Print the object, unzipping it if necessary
hadoop fs -text s3a://bucket1/hello.txt

# Download log files into a local file
hadoop fs -getmerge s3a://s3a://bucket1/logs\* log.txt
```

Related Links

Commands That May Be Slower with S3 [69]

Operations Unsupported for S3 [70]

Deleting Objects on S3 [70]

8.2.1. Commands That May Be Slower with S3

Some commands tend to be significantly slower with Amazon S3 than when invoked against HDFS or other filesystems. This includes renaming files, listing files, `find`, `mv`, `cp`, and `rm`.

Renaming Files

Unlike in a normal filesystem, renaming files and directories in an object store usually takes time proportional to the size of the objects being manipulated. As many of the filesystem shell operations use renaming as the final stage in operations, skipping that stage can avoid long delays.

In particular, we recommend that when using the `put` and `copyFromLocal` commands, you set the `-d`option for a direct upload. For example:

```
# Upload a file from the cluster filesystem
hadoop fs -put -d /datasets/example.orc s3a://bucket1/datasets/

# Upload a file from the local filesystem
hadoop fs -copyFromLocal -d -f ~/datasets/devices.orc s3a://bucket1/datasets/

# Create a file from stdin
echo "hello" | hadoop fs -put -d -f - s3a://bucket1/datasets/hello.txt
```

Listing Files

Commands which list many files tend to be significantly slower with Amazon S3 than when invoked against HDFS or other filesystems. For example:

```
hadoop fs -count s3a://bucket1/
hadoop fs -du s3a://bucket1/
```

Find

The `find` command can be very slow on a large store with many directories under the path supplied.

```
# Enumerate all files in the bucket
hadoop fs -find s3a://bucket1/ -print

# List *.txt in the bucket.
# Remember to escape the wildcard to stop the bash shell trying to expand it
hadoop fs -find s3a://bucket1/datasets/ -name \*.txt -print
```

Rename

The time to rename a file depends on its size. The time to rename a directory depends on the number and size of all files beneath that directory. If the operation is interrupted, the object store will be in an undefined state.

```
hadoop fs -mv s3a://bucket1/datasets s3a://bucket/historical
```

Copy

The copy operation reads each file and then writes it back to the object store; the time to complete depends on the amount of data to copy, and on the bandwidth in both directions between the local computer and the object store.

```
hadoop fs -cp s3a://bucket1/datasets s3a://bucket1/historical
```

Note

The further the VMs are from the object store, the longer the copy process takes.

8.2.2. Operations Unsupported for S3

S3A does not implement the same feature set as HDFS. The following FileSystem shell subcommands are not supported with an S3A URI:

- `-appendToFile`
- `-checksum`
- `-chgrp`
- `-chmod`
- `-chown`
- `-createSnapshot`
- `-deleteSnapshot`
- `-df`
- `-getfacl`
- `-getfattr`
- `-renameSnapshot`
- `-setfacl`
- `-setfattr`
- `-setrep`
- `-truncate`

8.2.3. Deleting Objects on S3

The `rm` command deletes objects and directories full of objects. If the object store is eventually consistent, `fs ls` commands and other accessors may briefly return the details of the now-deleted objects; this is an artifact of object stores which cannot be avoided.

If the filesystem client is configured to copy files to a trash directory, the trash directory is in the bucket. The `rm` operation then takes time proportional to the size of the data. Furthermore, the deleted files continue to incur storage costs.

To make sure that your deleted files are no longer incurring costs, you can do two things:

- Use the the `-skipTrash` option when removing files:`hadoop fs -rm -skipTrash s3a://bucket1/dataset`
- Use the `expunge` command to purge any data that has been previously moved to the `.Trash` directory:`hadoop fs -expunge -D fs.defaultFS=s3a://bucket1/`

  As the `expunge` command only works with the default filesystem, you need to use the `-D` option to make the target object store the default filesystem. This will change the default configuration.

8.2.4. Overwriting Objects on S3

Amazon S3 is eventually consistent, which means that an operation which overwrites existing objects may not be immediately visible to all clients and queries. As a result, later operations which query the same object's status or contents may get the previous object; this can sometimes surface within the same client, while reading a single object.

Avoid having a sequence of commands which overwrite objects and then immediately working on the updated data; there is a risk that the previous data will be used instead.

8.2.5. Timestamps on S3

Timestamps of objects and directories in Amazon S3 do not follow the behavior of files and directories in HDFS:

- The creation time of an object is the time when the object was created in the object store. This is at the end of the write process, not in the beginning.
- If an object is overwritten, the modification time is updated.
- Directories may or may not have valid timestamps.
- The `atime` access time feature is not supported by any of the object stores found in the Apache Hadoop codebase.

8.2.6. Security Model and Operations on S3

The security and permissions model of Amazon S3 is very different from this of a UNIX-style filesystem: on Amazon S3, operations which query or manipulate permissions are generally unsupported. Operations to which this applies include: `chgrp`, `chmod`, `chown`, `getfacl`, and `setfacl`. The related attribute commands `getfattr` and`setfattr` are also unavailable. In addition, operations which try to preserve permissions (for example `fs -put -p`) do not preserve permissions.

Although these operations are unsupported, filesystem commands which list permission and user/group details usually simulate these details. As a consequence, when interacting

with read-only object stores, the permissions found in "list" and "stat" commands may indicate that the user has write access — when in fact he does't.

Amazon S3 has a permissions model of its own. This model can be manipulated through store-specific tooling. Be aware that some of the permissions which can be set — such as write-only paths, or various permissions on the root path — may be incompatible with the S3A client. It expects full read and write access to the entire bucket with trying to write data, and may fail if it does not have these permissions.

As an example of how permissions are simulated, here is a listing of Amazon's public, read-only bucket of Landsat images:

```
$ hadoop fs -ls s3a://landsat-pds/
Found 10 items
drwxrwxrwx   - mapred          0 2016-09-26 12:16 s3a://landsat-pds/L8
-rw-rw-rw-   1 mapred      23764 2015-01-28 18:13 s3a://landsat-pds/index.html
drwxrwxrwx   - mapred          0 2016-09-26 12:16 s3a://landsat-pds/landsat-
pds_stats
-rw-rw-rw-   1 mapred        105 2016-08-19 18:12 s3a://landsat-pds/robots.txt
-rw-rw-rw-   1 mapred         38 2016-09-26 12:16 s3a://landsat-pds/run_info.
json
drwxrwxrwx   - mapred          0 2016-09-26 12:16 s3a://landsat-pds/runs
-rw-rw-rw-   1 mapred   27458808 2016-09-26 12:16 s3a://landsat-pds/
scene_list.gz
drwxrwxrwx   - mapred          0 2016-09-26 12:16 s3a://landsat-pds/tarq
drwxrwxrwx   - mapred          0 2016-09-26 12:16 s3a://landsat-pds/
tarq_corrupt
drwxrwxrwx   - mapred          0 2016-09-26 12:16 s3a://landsat-pds/test
```

As you can see:

- All files are listed as having full read/write permissions.
- All directories appear to have full `rwx` permissions.
- The replication count of all files is "1".
- The owner of all files and directories is declared to be the current user (`mapred`).
- The timestamp of all directories is actually that of the time the `-ls` operation was executed. This is because these directories are not actual objects in the store; they are simulated directories based on the existence of objects under their paths.

When an attempt is made to delete one of the files, the operation fails — despite the permissions shown by the `ls` command:

```
$ hadoop fs -rm s3a://landsat-pds/scene_list.gz
rm: s3a://landsat-pds/scene_list.gz: delete on s3a://landsat-pds/scene_list.
gz:
  com.amazonaws.services.s3.model.AmazonS3Exception: Access Denied (Service:
 Amazon S3;
  Status Code: 403; Error Code: AccessDenied; Request ID: 1EF98D5957BCAB3D),
  S3 Extended Request ID: wi3veOXFuFqWBUCJgV3Z+NQVj9gWgZVdXlPU4KBbYMsw/gA
+hyhRXcaQ+PogOsDgHh31HlTCebQ=
```

This demonstrates that the listed permissions cannot be taken as evidence of write access; only object manipulation can determine this.